UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO
CIÊNCIA DA COMPUTAÇÃO

Módulo de Segurança de Compilação de Objetos em Ambiente de Desenvolvimento utilizando Banco de Dados Oracle

Pablo Rodriguez de Almeida Gross

Caxias do Sul
2014

Pablo Rodriguez de Almeida Gross

Módulo de Segurança de Compilação de Objetos em Ambiente de Desenvolvimento utilizando Banco de Dados Oracle

Monografia apresentada como exigência para obtenção do grau de Bacharelado em CIÊNCIA DA COMPUTAÇÃO da UNIVERSIDADE DE CAXIAS DO SUL.

Orientador: Daniel Luis Notari

Caxias do Sul
2014

RESUMO

Quando grandes equipes trabalham no desenvolvimento de um sistema, é comum que mais de um desenvolvedor necessite alterar o mesmo código fonte. Para evitar perda ou sobrescrita de código são utilizados softwares de controle de versão, porém quando se trabalha com objetos armazenados dentro dos bancos de dados os controladores de versão não conseguem atuar sobre o código compilado. Este trabalho apresenta um método para integrar o controlador de versão da Microsoft, o Microsoft Visual SourceSafe, com o banco de dados da Oracle.

Palavras-chave: Controle de Versão, Bancos de Dados, Microsoft Visual SourceSafe, Oracle**.**

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BD	Banco de Dados
CVCS	Centralized Version Control System
DCL	Data Control Language
DDL	Data Definition Language
DLL	Dynamic-Link Library
DML	Data Manipulation Language
DQL	Data Query Language
DVCS	Distributed Version Control System
IDE	Integrated Development Enviroment
IUP	Portable User Interface
JIT	Just-In-Time
OO	Orientada (o) a Objetos
PL/SQL	Procedural Language extensions to SQL
RGE	Rio Grande Energia
SCCS	Source Code Control System
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
SVN	Subversion
TCL	Transaction Control Language
TFS	Team Foundation Server
VCS	Version Control System
VSS	Visual SourceSafe

SUMÁRIO

1 INTRODUÇÃO

Segundo BAPTISTA (2013), um banco de dados pode ser definido como "uma coleção de dados relacionados e armazenados em algum dispositivo" e DATE (2004) define um sistema gerenciador de banco de dados (SGBD) como "um sistema computadorizado cujo propósito geral é armazenar informações e permitir ao usuário buscar e atualizar essas informações quando solicitado". KORTH et. al (2006) acrescentam que um BD possui "uma coleção de dados inter-relacionados e um conjunto de programas para acessar esses dados".

Para manipular os dados em um BD foi criada a linguagem SQL (*Structured Query Language*). BEAULIEU e MISHRA (2002) definem a SQL como uma linguagem não-procedural que permite que dados sejam selecionados, alterados e definidos. A linguagem SQL é dividida, conforme Greenberg e Nathan (2001), em Linguagem de Manipulação de Dados (DML), Linguagem de Definição de Dados (DDL), Linguagem de Controle de Dados (DCL), Linguagem de Controle de Transações (TCL) e Linguagem de Consulta de Dados (DQL).

O SGBD ORACLE da Oracle Corporation utiliza a linguagem PL/SQL que é uma derivação da SQL. Nathan e Pataballa (2001) definem PL/SQL como uma extensão procedural da SQL com funcionalidades de linguagens de programação que possui a capacidade de manipular dados e utilizar instruções de consulta. Dentre as funcionalidades de linguagens de programação mencionadas, pode-se destacar a utilização de comandos DDL em objetos como: funções, procedimentos, pacotes, visões e *triggers.*

Equipes de desenvolvimento necessitam de softwares que se conectem com o banco de dados e habilitem uma interface para a criação e alteração de esquemas. Quando se fala em equipe de desenvolvimento, se fala em diversas pessoas alterando objetos, e muitas vezes estes objetos podem ser os mesmos. Desta forma se torna importante controlar estes objetos. Para isso existem os sistemas de controle de versão. De acordo com LACERDA (2013), as principais funções desse tipo de software são: registrar a evolução de um projeto, gerenciando as diferentes versões dos documentos e possibilitar que vários desenvolvedores editem o mesmo arquivo sem que haja perda de código.

Um sistema de controle de versão comum é formado por duas partes, o

repositório central e a(s) área(s) de trabalho (DIAS, 2013). O primeiro guarda todas as versões dos arquivos modificados, o segundo é onde a(s) cópia(s) do(s) arquivo(s) a ser (em) modificado(s) é (são) salva(s), até que as mudanças sejam finalizadas e o(s) arquivo(s) seja(m) devolvido(s) ao repositório (Figura 1).

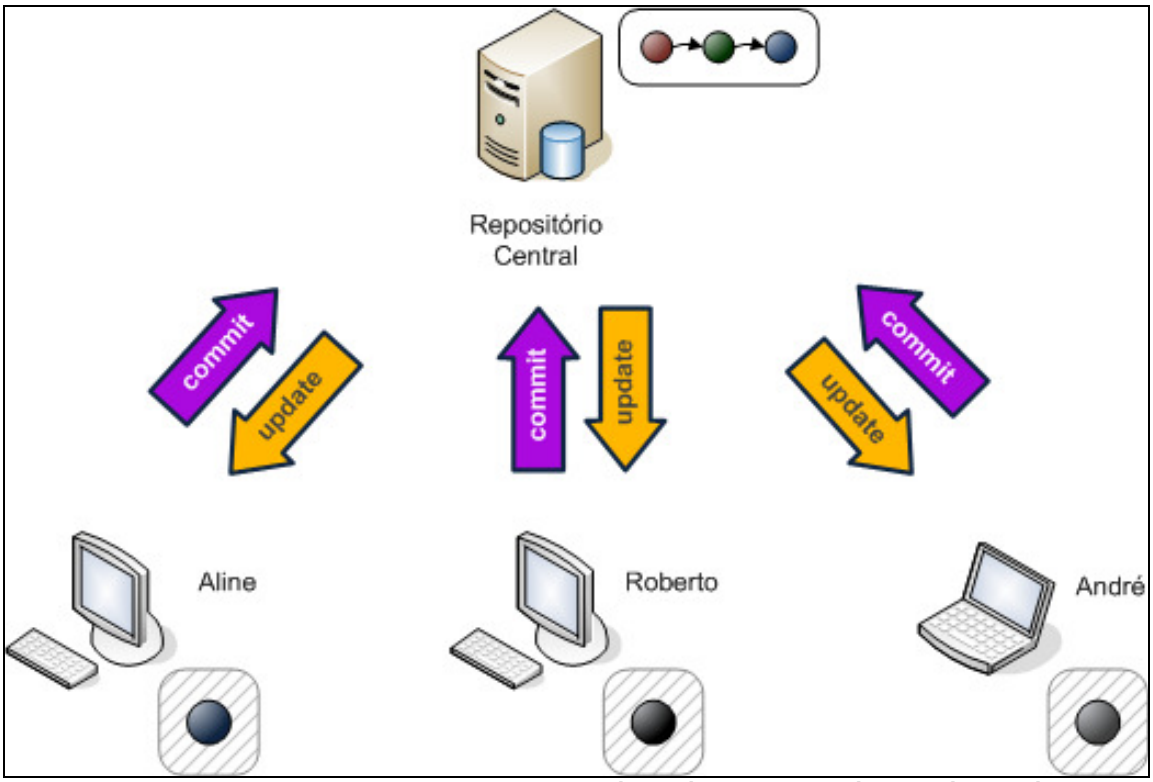

Figura 1 - No controle de versão centralizado há um único repositório e várias cópias de trabalho que se comunicam apenas através do repositório central.
Fonte: (DIAS A.F, 2013).

Nos bancos de dados, um objeto somente pode ser compilado diretamente no servidor, ou seja, não é possível trazer o arquivo para a área de trabalho e fazer uma compilação local. Em ambientes de desenvolvimento existe a possibilidade que mais de um desenvolvedor esteja trabalhando com o mesmo objeto, supondo que todos em uma equipe de desenvolvimento tenham privilégio de compilação de objetos. Toda vez que um desenvolvedor requisitar a compilação de um objeto já existente, este será compilado. Caso algum desenvolvedor já esteja trabalhando com uma versão anterior e solicite uma compilação o código compilado pelo primeiro desenvolvedor será perdido.

A situação acima descrita, pode ser melhor demonstrada na Figura 2. Esta contém a seguinte situação: A Área de Trabalho 1 e a Área de Trabalho 2 já

possuem o código do objeto na versão atual. Área de Trabalho 1 faz uma modificação e solicita uma compilação do objeto, logo em seguida a Área de Trabalho 2 faz uma requisição de compilação do mesmo objeto que a Área de Trabalho 1 havia solicitado. As duas Áreas de Trabalho recebem a confirmação de que o objeto foi compilado, mas o código enviado pela Área de Trabalho 1 foi compilado e logo em seguida sobreposto pelo código enviado pela Área de Trabalho 2. Assim o código enviado pela Área de Trabalho 1 foi perdido.

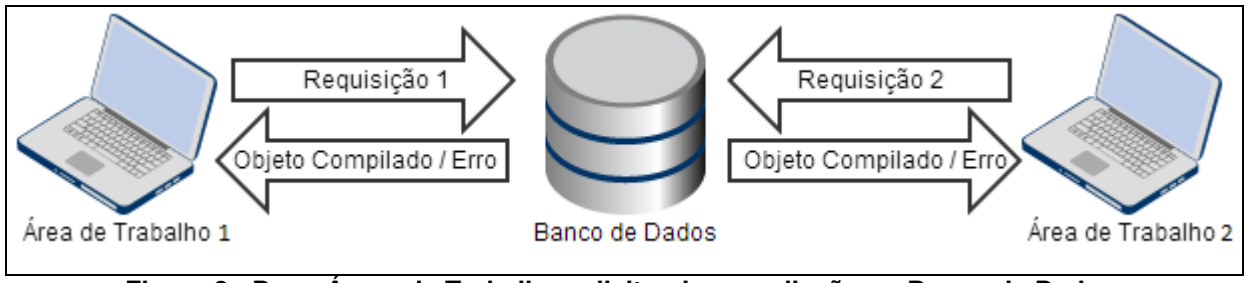

Figura 2 - Duas Áreas de Trabalho solicitando compilação no Banco de Dados.
Fonte: Autoria própria.

1.1 Objetivos

O principal objetivo deste trabalho é projetar e implementar um método para que seja possível utilizar um ambiente de desenvolvimento no SGBD da Oracle, no qual diversos desenvolvedores possam trabalhar sem interferir no código de outro desenvolvedor.

1.1.1 Objetivos Específicos

Dentre os objetivos específicos, podem ser listados:

1. Pesquisar soluções semelhantes
2. Elaborar um método de controle de compilação.
3. Desenvolver um protótipo para o método proposto.
4. Testar e validar o protótipo desenvolvido.

1.2 Estrutura do Trabalho

Neste capítulo foi apresentada uma breve introdução sobre banco de dados, SQL e controle de versão. Além disso, foram descritos os objetivos gerais e específicos deste trabalho e, por fim, a estrutura do texto.

No capítulo 2 são apresentados conceitos de controle de versão e softwares com essa finalidade. O capítulo 3 aprofunda-se na explicação do funcionamento da linguagem PL/SQL. Um problema e uma proposta de solução são apresentados no capítulo 4. A implementação da proposta apresentada no capítulo 4 é apresentada no capítulo 5 e os testes realizados são mostrados no capítulo 6. No capítulo final, o 7, são apresentadas as conclusões e possíveis melhorias futuras.

2 CONTROLE DE VERSÃO

De acordo com CHACON (2009) o controle de versão é "um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo de forma que você possa recuperar versões específicas". Antes do lançamento do *Source Code Control System (SCCS)* na década de 70 (BOLINGER; BRONSON, 1995), que foi considerado a primeira ferramenta de controle de versões (GLASSER, 1978), o único modo de ter as diversas versões de um arquivo, era salvar todas as versões dos arquivos com nomes diferentes. Utilizar o número da versão do arquivo no final do nome, ou utilizar a data e hora eram técnicas utilizadas para manter a organização. Essa técnica ainda hoje é muito comum pela sua simplicidade, porém é altamente vulnerável a erros, pois se trata de um processo manual. Gravar acidentalmente no diretório errado, esquecer o local onde as versões devem ser salvas ou sobrescrever um arquivo são problemas comuns ao utilizar-se desta técnica.

Para evitar erros manuais e automatizar o versionamento, foram criados os Sistemas de Controle de Versão (*Version Control System* - VCS). De maneira simplificada a Figura 3 mostra como funciona um VCS. Atualmente existem dois tipos de VCSs, o centralizado (*Centralized Version Control System* - CVCS) e o distribuído (*Distributed Version Control System* - DVCS) (DIAS, 2013).

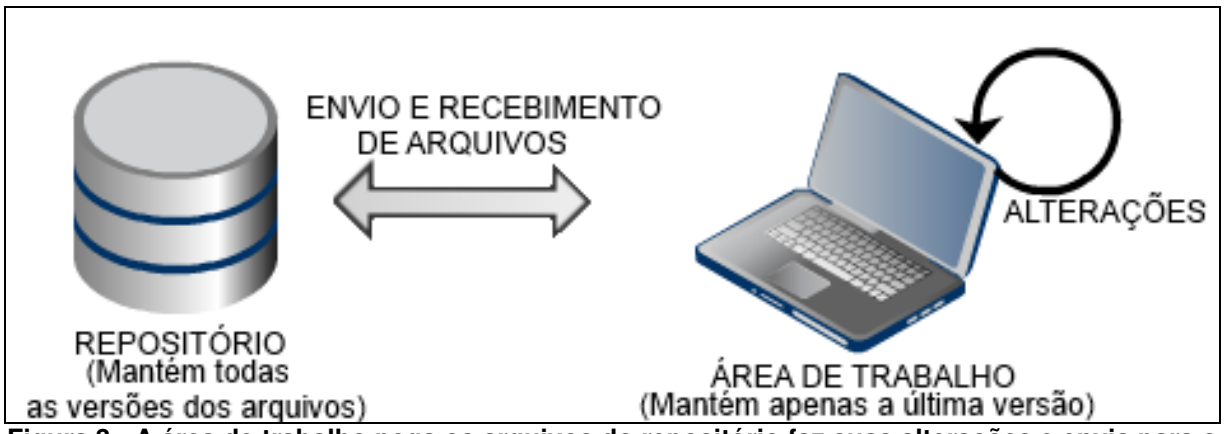

Figura 3 - A área de trabalho pega os arquivos do repositório faz suas alterações e envia para o repositório que grava todas as alterações dos arquivos.
Fonte: Autoria própria.

2.1 Sistema de Controle de Versão Centralizado

DIAS (2013) explica que os CVCSs possuem um servidor central que segue a topologia em estrela, este servidor possui todos os arquivos versionados. Os clientes podem resgatar esses arquivos diretamente do servidor utilizando um comando próprio para busca, geralmente *check out* ou *pull*.

A desvantagem desse modelo é que o servidor central é um ponto único de falha, ou seja, caso o servidor fique fora do ar ou se o disco do servidor tiver algum problema, não é possível buscar novos arquivos. Caso o servidor não possua backup, todas as versões antigas e arquivos que não estiverem nas máquinas dos clientes serão perdidos. Como mostra a Figura 4, em um CVCS toda operação de busca ou alteração de arquivos passa pelo servidor.

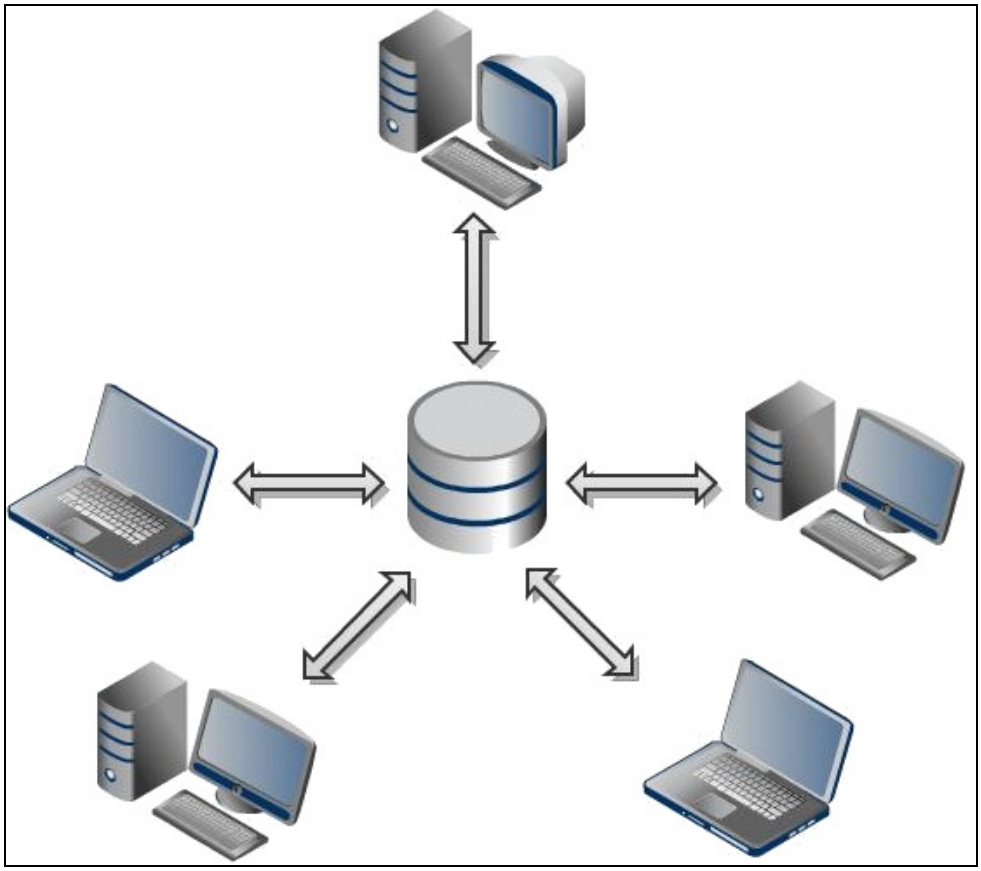

Figura 4 - Funcionamento de CVCS, todas as operações de busca ou entrega de arquivos passa obrigatoriamente pelo servidor.
Fonte: Autoria própria.

2.2 Sistema de Controle de Versão Distribuído

Nos DVCSs os clientes não fazem a cópia apenas das últimas versões, mas sim de todas as versões do histórico (CHACON, 2009). Na prática, todo o *check out* feito é na verdade uma cópia de todo o repositório. A desvantagem da utilização deste método é a grande quantidade de arquivos repetidos que irá existir, um arquivo e todas as suas versões em cada máquina cliente. A Figura 5 mostra que em um DVCS, todos os clientes possuem as versões dos arquivos e podem se comunicar sem necessidade de acesso ao servidor.

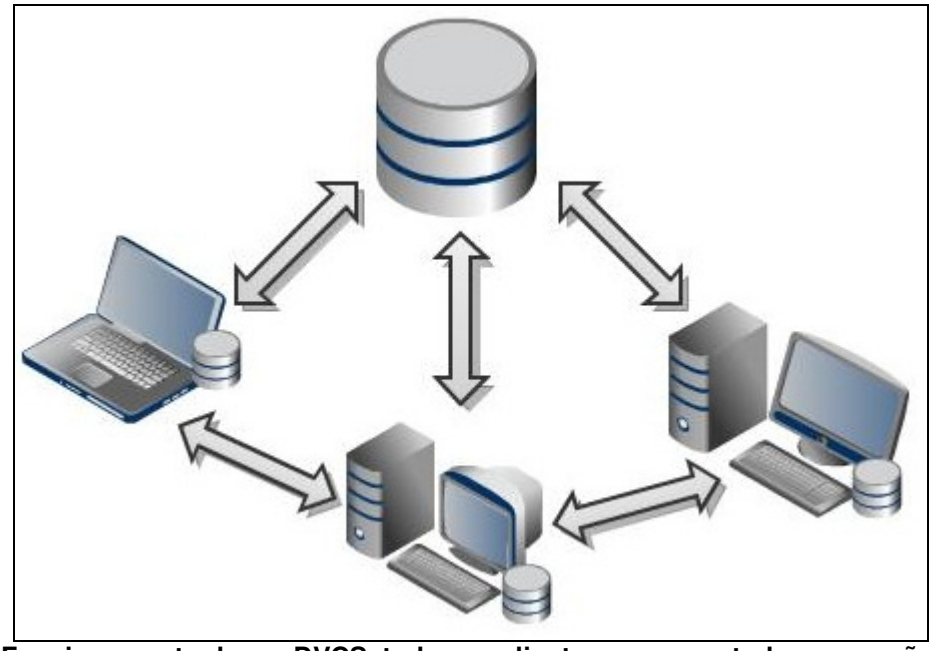

Figura 5 - Funcionamento de um DVCS, todos os clientes possuem todas as versões de todos os arquivos e podem fazer trocas de arquivos sem passar pelo servidor.
Fonte: Autoria própria.

2.3 Controle de Código Fonte

Apesar dos sistemas de controle de versão poderem ser utilizados para praticamente qualquer tipo de arquivo, sua utilização é mais comum no versionamento de código fonte.

Segundo HARPER (2009) "o controle de código fonte é importante em qualquer ambiente de desenvolvimento, se você está trabalhando por conta própria

ou com outros usuários em uma equipe acessando os mesmos arquivos. Às vezes chamado de gerenciamento de código fonte ou controle de versões, controle de código fonte fornece um histórico de revisão do código que você está trabalhando e a capacidade de comparar as alterações e mesclar documentos"

Além do comando *check out*, que serve para recuperar um arquivo do servidor e criar uma cópia local, existem outros comandos que são básicos em todos os controladores de versão como por exemplo o *check in* (também conhecido por *commit*) que serve para enviar uma nova versão para o servidor. Os VCSs podem ser configurados de duas maneiras: a *file locking* e a *version merging, t*ambém conhecidas como *lock-modify-unlock* e *copy-modify-merge* respectivamente (COLLINS-SUSSMAN; FITZPATRICK; PILATOCC, 2011).

Na configuração de *file locking* somente um cliente pode modificar o conteúdo de um arquivo, ou seja, enquanto o arquivo estiver reservado para um cliente nenhum dos outros pode enviar alterações deste arquivo para o servidor. Geralmente se associa a operação de *file lock* com a operação de *check out* para que quando um cliente recupere um arquivo este fique bloqueado. O arquivo que estiver em *lock* só poderá voltar a ser editado por outros clientes quando o cliente que pediu o *lock* fizer manualmente o *unlock (*também conhecido como *undo check out)* ou enviar uma nova versão para o servidor.

A configuração de *version merging* permite que diversos clientes possam alterar o mesmo arquivo. Para isso, no momento da devolução do arquivo modificado (*check in*) o programa de VCS identifica que outro cliente já havia feito uma alteração e avisa ao segundo cliente que ele deve fazer um *merge*, ou seja, mesclar as alterações feitas pelo outro cliente com as dele. A maioria dos VCSs possui uma interface que mostra as diferenças entre a versão que está sendo enviada e a que está atualizada no servidor. As Figuras 6, 7 e 8 mostram como funciona a operação de *merge* em um sistema de controle de versão.

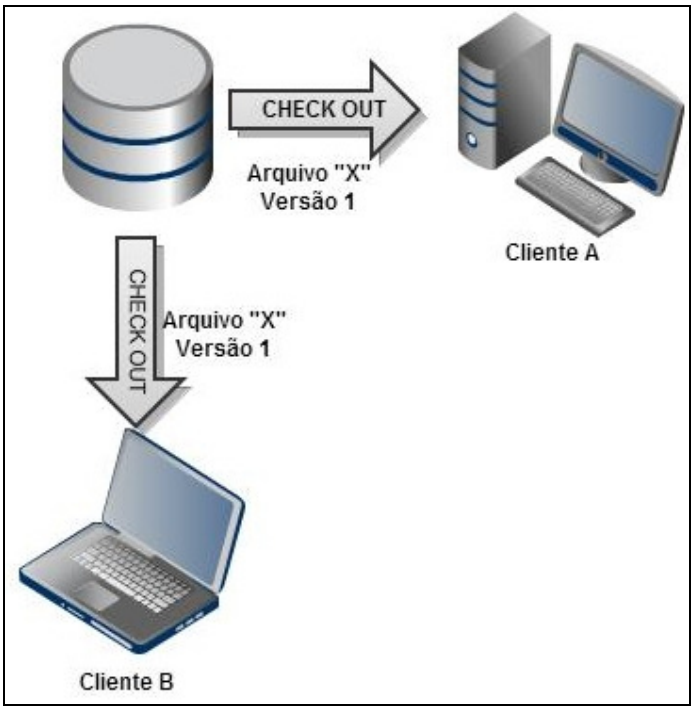

Figura 6 - Cliente A e o Cliente B buscam a versão 1 de um Arquivo "X"
Fonte: Autoria própria.

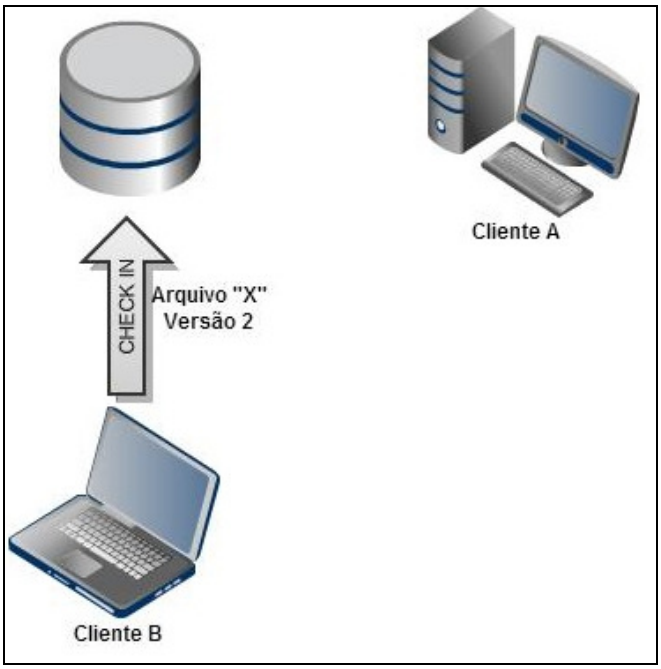

Figura 7 - Cliente B envia uma versão 2 do Arquivo "X" para o servidor.
Fonte: Autoria própria.

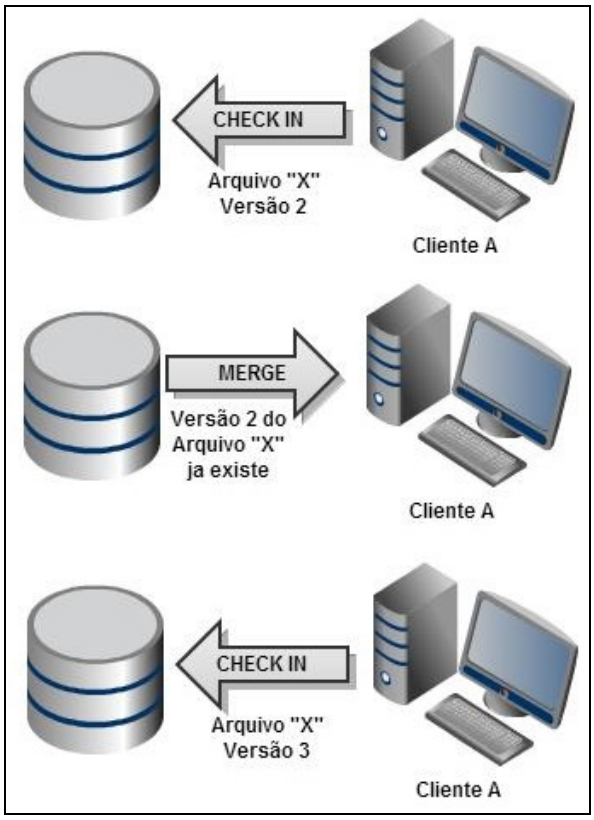

Figura 8 – Servidor avisa sobre versão nova, Cliente A deve realizar o merge.
Fonte: Autoria própria.

2.4 Ferramentas de Segurança de Código Fonte

Serão analisados os softwares Microsoft Visual SourceSafe[1], Subversion[2] e o Git[3]. Os critérios da análise serão: licença, tipo de controlador, compatibilidade com sistemas operacionais, interoperabilidade com IDE's, documentação e implantação.

2.4.1 Microsoft Visual SourceSafe

O SourceSafe foi originalmente criado pela One Tree Software, que foi comprada pela Microsoft em 16 de novembro de 1994[4]. A partir desta data, as novas

[1] http://msdn.microsoft.com/pt-br/library/ms181038(v=vs.80).aspx
[2] http://subversion.apache.org/
[3] http://git-scm.com/

[4] http://www.nytimes.com/1994/11/16 /business/company-news-microsoft-says-it-has-acquired-one-

versões do software passam a ser chamadas de Microsoft Visual SourceSafe (VSS). No final de 2005 a Microsoft lançou a última versão do SourceSafe e descontinuou o desenvolvimento do software para dar lugar a sua nova ferramenta de controle de versão, o Team Foundation Server[5] (TFS). Apesar de ter sido descontinuado, o VSS ainda é amplamente utilizado e tem suporte estendido até 2017[6].

O VSS é conhecido por ser um sistema controlador de versão centralizado e híbrido, ou seja, todas as versões atualizadas dos arquivos estão em um servidor e ele pode ser configurado como *file locking* e *version merging*. O controlador de versão funciona em cima de um banco de dados e possui três programas distintos para acessá-lo: o de administrador (*Visual SourceSafe Administrator*), o de cliente (*Visual SourceSafe Explorer*) e o programa de acesso por linha de comando (*Visual SourceSafe command line utility*). Utilizando a interface de cliente o software traz uma nova funcionalidade, que é abrir uma versão de leitura dos arquivos sem a necessidade de fazer uma cópia local, ou seja, abrir direto do servidor onde está armazenado. Além desta funcionalidade todas as funcionalidades padrão de um controlador de versão estão disponíveis.

2.4.2 Subversion

COLLINS-SUSSMAN, FITZPATRICK e PILATOCC (2011) explicam que o Subversion, popularmente conhecido como SVN, é um CVCS *open source.* Este começou a ser desenvolvido no início do ano 2000 com o objetivo de substituir o CVS devido as suas limitações. Até a data em que esse trabalho foi escrito, a versão mais atual do SVN era a 1.8.4, disponível em http://subversion.apache.org.

O SVN pode ser configurado como *file locking* e *version merging.* Em situações em que um cliente pode destruir o trabalho de outro, como é o caso da compilação de objetos em BD, é recomendado o uso da configuração de *file locking.* Os autores justificam a utilização de *file locking* com exemplos de arquivos binários, como os de imagens e sons, onde geralmente não é possível fazer um *merge* dos

---

tree-software.html
[5] http://msdn.microsoft.com/pt-br/vstudio/ff637362.aspx
[6]http://support.microsoft.com/lifecycle/search/default.aspx?sort=PN&alpha=sourcesafe&Filter=FilterNO

arquivos.

Um dos pontos positivos do uso do SVN é a portabilidade. Tanto o cliente quanto o servidor podem ser utilizados em diversos sistemas operacionais: Unix, Mac OS X e Windows. Além disto, o SVN proporciona diversos *plugins*. Estes encontram-se disponíveis para IDEs como NetBeans, Eclipse e Visual Studio. A documentação encontrada sobre SVN também pode ser considerado como outro ponto positivo da ferramenta. A principal referência do aplicativo é um livro gratuito escrito por alguns dos desenvolvedores do SVN, disponibilizado na web.

O principal ponto fraco do SVN comparado a outros VCSs é o seu processo de implantação que não é tão simples e possui uma série de dependências para instalação.

2.4.3 Git

Segundo CHACON (2009), o Git é um DVCS *open source* que começou a ser desenvolvido em 2005 pela equipe responsável pelo desenvolvimento do kernel do Linux. Esse fato ocorreu pois a ferramenta utilizada pela equipe, o BitKeeper, deixou de ser gratuita. Ainda em 2005, no dia 21 de dezembro, a versão 1.0 do Git foi lançada. Até a data em que esse trabalho foi escrito, a versão mais atual do Git era a 1.8.4.3 , disponível em http://git-scm.com/downloads.

Apesar do Git ter sido desenvolvido para usuários do Linux ele também pode ser utilizado em outros sistemas operacionais como: Windows, Solaris, Darwin, BSD e outros sistemas baseados em Unix.

O Git, por ser um sistema de controle distribuído, só permite a configuração de *version merging*. Não faz sentido utilizar *file locking* sendo que todos os clientes possuem uma cópia local de cada arquivo.

Assim como o SVN o Git também possui uma boa documentação disponível e grande disponibilidade de *plugins* para diversas IDE's. A facilidade de instalação, comparado aos outros VCS's mencionados, é um ponto a favor do Git.

2.4.4 Considerações Finais

A análise realizada nos três softwares de controle de versão permitiu a geração da Tabela 1 que permite realizar as seguintes conclusões. Nos softwares gratuitos a diferença encontra-se no grau de facilidade de implantação e no tipo de CVS, distribuído ou centralizado. Nos demais critérios, como qualidade da documentação e grau de interoperabilidade ambos tem características semelhantes. Outra semelhança nos dois softwares *open source* é que existe a possibilidade de serem utilizados em praticamente qualquer sistema operacional. O SourceSafe, que além de ser pago, se restringe ao sistema operacional Windows, restrição inexistente nos demais. Além disso a documentação disponível sobre o mesmo não é tão boa comparada aos demais softwares. Este fato talvez se deva ao fato do mesmo ter sido descontinuado, o que desestimula a escrita de livros e artigos. Enquanto estava ativo o software foram feitas diversas integrações entre este e as IDE's que continuam funcionando até o momento.

Tabela 1 - Comparação entre softwares de controle de versão

	SourceSafe	Subversion	Git
Licença	Paga	Gratuita	Gratuita
Tipo	Centralizado	Centralizado	Distribuído
Compatibilidade	Windows	Diversos	Diversos
Interoperabilidade	Alta	Alta	Alta
Documentação	Boa	Excelente	Excelente
Implantação	Mediana	Mediana	Fácil

Fonte: Autoria própria.

3 SISTEMAS DE BANCOS DE DADOS

Como já foi explocado na introdução um banco de dados é um conjunto de dados que possuem algum tipo de relação e que ficam armazenados em algum local.

Independentemente do modelo de banco de dados, todos precisam de um Sistema Gerenciador de Banco de Dados (SGBD) que é o "software que incorpora as funções de definição, recuperação e alteração de dados em um banco de dados." (HEUSER, 2010). O SGBD deve oferecer, segundo BAPTISTA (2013), segurança, controle de redundância, compartilhamento de dados, independência de dados, backup e recuperação a falhas, aumento de produtividade e disponibilidade, flexibilidade e padronização.

Segundo Greenberg e Nathan (2001), para inclusão, remoção e alteração dos dados são utilizados os comandos de DML, *insert*, *update* e *delete.* Esses comandos podem ser utilizados tanto por usuários quanto por programas de aplicação. As estruturas onde os dados são inseridos, removidos e atualizados são as tabelas. As tabelas, assim como diversos outros objetos de bancos de dados, podem ser criadas por usuários que possuem permissão para executar comandos do tipo DDL. Comandos DDL como *create*, *alter* e *drop* são utilizados para criar, modificar, destruir os objetos do banco de dados e criar as relações entre eles. Para buscar os dados é utilizada a DQL. Apesar da DQL ter apenas um comando, o *select ,* este possui diversos subcomandos utilizados para especificar quais os dados devem ser recuperados e como estes serão mostrados.

Os dois últimos tipos de comandos são o DCL e o TCL. O primeiro é utilizado para controlar aspectos de visualização, manipulação e execução nos objetos. Os comandos *grant* e *revoke* são os que dão ou retiram as permissões. As permissões podem ser concedidas tanto a usuários quanto a objetos. O segundo define comandos que controlam as transações do banco de dados, o comando *commit* confirma que as informações devem ser salvas no BD e o *rollback* serve para desfazer as transações. Existem outros comandos TCL para definir propriedades e funcionamento das transações além da criação de *savepoints* mas a sintaxe destes varia muito entre os SGBDs.

3.1 PL/SQL

A PL/SQL, que significa *Procedural Language extensions to SQL*, foi lançada em 1991 na versão 1.0 juntamente com a Versão 6.0 do Oracle (HARDMAN; MCLAUGHLIN; URMAN, 2004) para ser utilizada com o SGBD e com outras ferramentas Oracle.

Atualmente na versão 11, a PL/SQL herda todas as funcionalidades existentes no SQL ANSI e acrescenta diversas funcionalidades existentes em outras linguagens de programação, como: variáveis, condições, laços (*loops*), exceções (*exceptions*), vetores, agendamento (*scheduling*), subprogramas, grandes objetos (*large objects*), SQL's dinâmicos, manipulação de arquivos, conceitos de orientação a objetos, paralelismo, proteção de código, etc.

3.1.1 Bloco PL/SQL

Segundo NATHAN e PATABALLA (2001a Oracle9i: Program with PL/SQL) A estrutura padrão da PL/SQL é um bloco. Todos os programas em PL/SQL são compostos por blocos, que podem estar uns dentro dos outros. Um bloco contém uma seção de declaração, uma de execução e uma de manipulação de exceções, sendo que apenas a seção de execução é obrigatória. A Figura 9 mostra como é a estrutura de um bloco e a Figura 10 é um exemplo de bloco PL/SQL com instruções SQL.

```
--Estrutura de um Bloco PL/SQL
DECLARE --(Opcional)
    /*Declara variáveis e objetos PL/SQL que
    irão ser utilizados dentro deste bloco*/
BEGIN --(Obrigatório)
    --Define as instruções que serão executadas
EXCEPTION --(Opcional)
    /*Define a ação a ser tomada caso
      ocorra algum erro ou exceção*/
END; --(Obrigatório)
```

Figura 9 - Estrutura de um Bloco em PL/SQL.
Fonte: Autoria Própria

```
DECLARE
  v_saida VARCHAR2(50);
  dt_dia  DATE;
  n_soma  NUMBER;

BEGIN
  --Comando DQL para buscar informação
  SELECT TRUNC(SYSDATE) INTO dt_dia FROM dual;

  --Comando PL/SQL para atribuição de valor após operação aritmética
  n_soma := 0 + 1;

  --Comando DML para inserir informação em uma tabela
  INSERT INTO nome_tabela (dia, valor) VALUES (dt_dia, n_soma);

  --Atribuição de valor
  v_saida := 'Comandos executados com sucesso.';

  --Comando TCL para gravar os dados
  COMMIT;

  --Comando PL/SQL para exibir informação
  dbms_output.put_line(v_saida);

EXCEPTION
  WHEN no_data_found THEN
    v_saida := 'Erro ao buscar dados.';
    dbms_output.put_line(v_saida);

  WHEN OTHERS THEN
    v_saida := 'Erro na execução.';
    dbms_output.put_line(v_saida);
END;
```

Figura 10 - Exemplo de um arquivo fonte do PL/SQL.
Fonte: Autoria própria.

3.1.2 Procedures e Functions

PL/SQL possui dois tipos de subprogramas: as *procedures* e as *functions*. Um subprograma é baseado na estrutura padrão do PL/SQL, porém com algumas modificações (Figura 11). Um subprograma deve ser nomeado, compilado e armazenado no banco de dados. Subprogramas proveem modularidade, extensibilidade, reusabilidade e manutenção (NATHAN e PATABALLA, 2001).

```
--Estrutura de um Bloco para Subprogramas PL/SQL
<header> --(Obrigatório) Especificação do Subprograma
is|as
    /*Declara variáveis e objetos PL/SQL que
    irão ser utilizados dentro deste bloco*/
BEGIN --(Obrigatório)
    --Define as instruções que serão executadas
EXCEPTION --(Opcional)
    /*Define a ação a ser tomada caso
      ocorra algum erro ou exceção*/
END; --(Obrigatório)
```

Figura 11 - Estrutura de um Bloco de um Subprograma PL/SQL.
Fonte: Autoria Própria

Tanto a *procedure* quanto a *function* podem ser facilmente definidas como um bloco PL/SQL que está armazenado no banco de dados, possui um nome e pode ser chamado por outros programas (NATHAN e PATABALLA, 2001). As sintaxes de utilização de uma *procedure* e de uma *function* podem ser visualizadas através das Figuras 12 e 13 respectivamente. A Figura 14 é um exemplo simplificado do que é mostrado na Figura 12, assim como a Figura 15 é um exemplo da Figura 13.

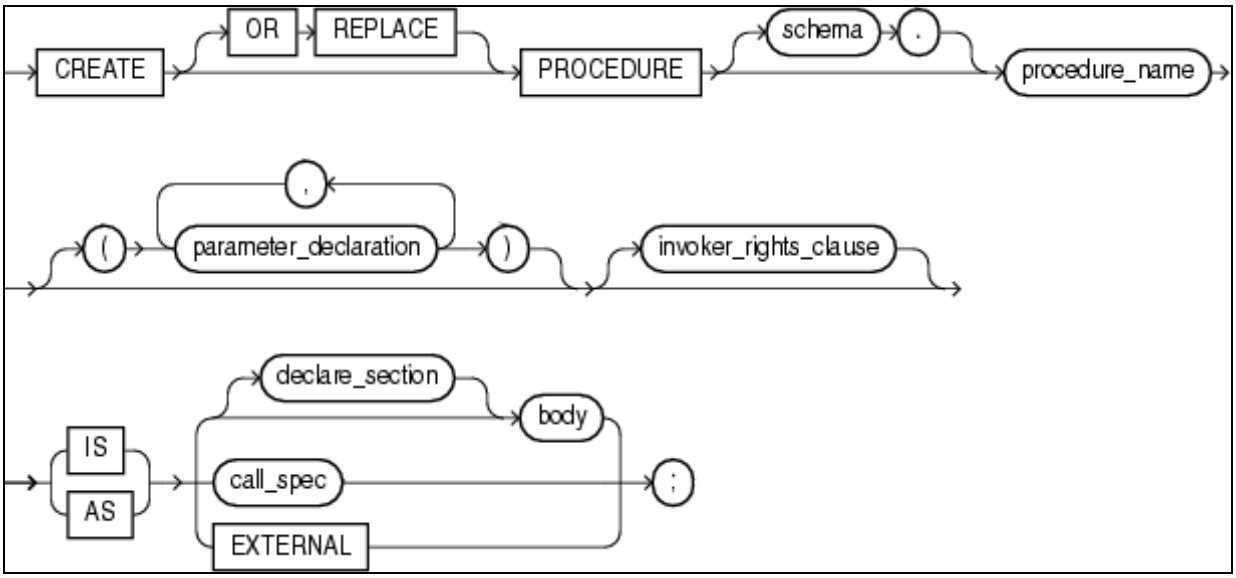

Figura 12 - Sintaxe de uma procedure PL/SQL.
Fonte: (MOORE, 2009).

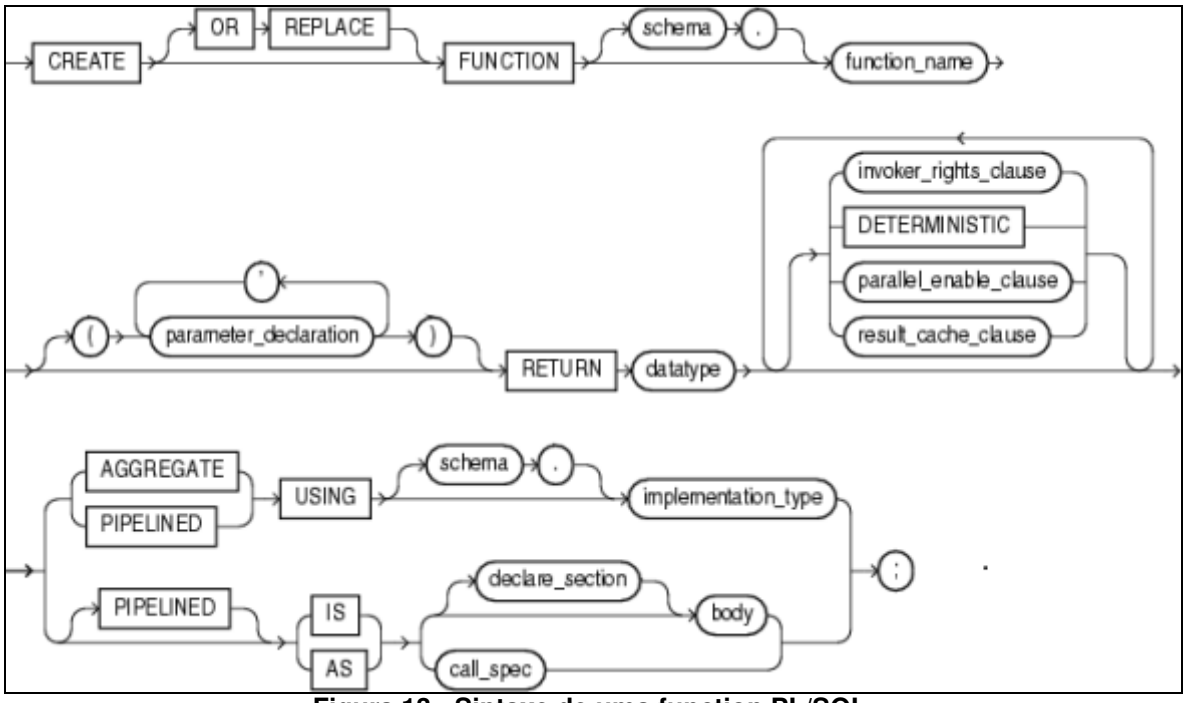

Figura 13 - Sintaxe de uma function PL/SQL.
Fonte: (MOORE, 2009).

```
CREATE OR REPLACE PROCEDURE p_exemplo(n_valor      IN NUMBER,
                                      v_msg_saida OUT VARCHAR2)

 IS
  dt_dia DATE;
BEGIN
  --Comando DQL para buscar informação
  SELECT TRUNC(SYSDATE) INTO dt_dia FROM dual;

  --Comando DML para inserir informação em uma tabela
  INSERT INTO nome_tabela (dia, valor) VALUES (dt_dia, n_valor);

  --Atribuição de valor
  v_msg_saida := 'Comandos executados com sucesso.';

  --Comando TCL para gravar os dados
  COMMIT;

EXCEPTION
  WHEN no_data_found THEN
    v_msg_saida := 'Erro ao buscar dados.';
    RETURN;

  WHEN OTHERS THEN
    v_msg_saida := 'Erro na execução.';
    RETURN;

END;
```

Figura 14 - Exemplo de uma procedure PL/SQL.
Fonte: Autoria própria.

```
CREATE OR REPLACE FUNCTION f_exemplo RETURN DATE IS
  dt_dia DATE;
BEGIN
  SELECT trunc(SYSDATE) INTO dt_dia FROM dual;
  RETURN(dt_dia);
END f_exemplo;
```

Figura 15 - Exemplo de uma function PL/SQL.
Fonte: Autoria própria.

A principal diferença entre a *function* e a *procedure* é que a primeira possui obrigatoriamente um parâmetro de retorno e a segunda pode ter nenhum, um ou mais parâmetros de retorno. Outra importante diferença entre as duas que pode ser citada é que a *function* pode ser utilizada juntamente com comandos DML e DQL.

3.1.3 Módulos

PATABALLA (2001) explica que, na área da tecnologia da informação, a modularização é um conceito no qual um sistema, software ou bloco é dividido em pequenas partes chamadas módulos. Após modularizado, um código pode ser reutilizado pelo mesmo programa ou ainda pode ser compartilhado com outros programas que necessitem da mesma funcionalidade. A divisão por módulos facilita a manutenção e a depuração do código. Ao ser utilizada, a modularização proporciona ainda uma economia de espaço e memória.

Subprogramas agilizam a manutenção do código, pois o código está localizado em um único local e quaisquer modificações que sejam necessárias serão realizadas uma única vez. Outra vantagem da utilização de subprogramas é a maior integridade e segurança dos dados. Os objetos de dados são acessados através do subprograma e um usuário pode invocar o subprograma somente se possuir um privilégio de acesso apropriado. A Figura 16 mostra como a utilização de subprogramas pode ser utilizada em um programa PL/SQL que possui uma mesma funcionalidade escrita diversas vezes. A Figura 17 mostra a procedure "p_exemplo", mostrada na Figura 14, sendo chamada em dois locais, ou seja, sem a necessidade de reescrever o código novamente.

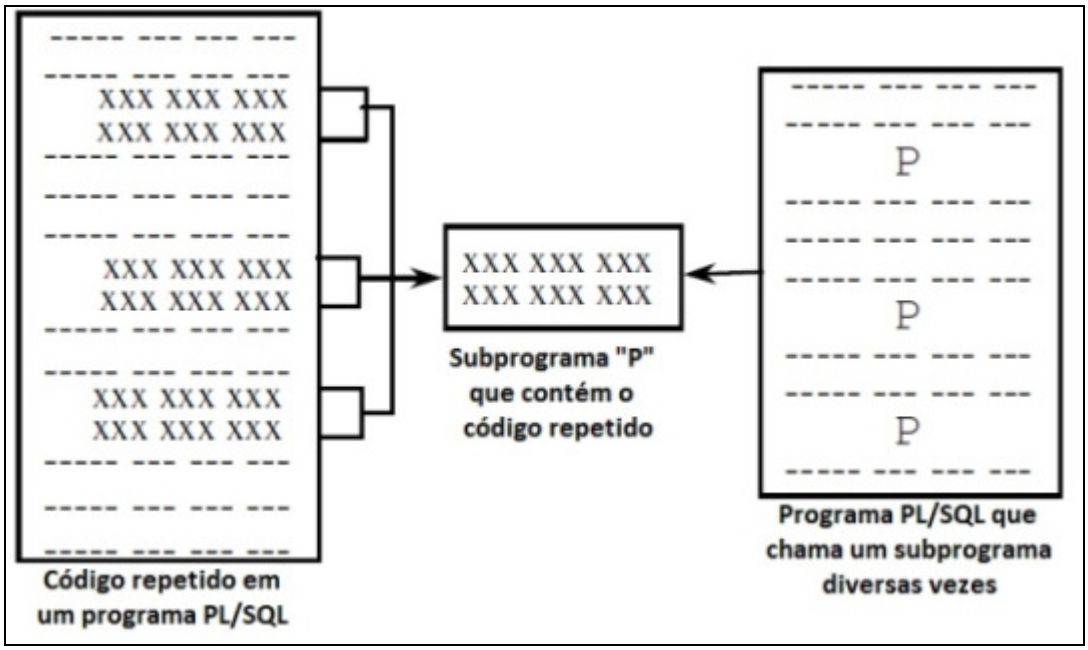

Figura 16 - Exemplo de utilização de Subprogramas no PL/SQL
Fonte: Adaptado de PATABALLA (2001).

```
DECLARE
  v_saida VARCHAR2(50);
  dt_dia  DATE;
  n_soma  NUMBER;

BEGIN

  SELECT MAX(valor) INTO n_soma FROM nome_tabela;

  IF n_soma IS NOT NULL THEN
    --Chamada da procedure
    p_exemplo(n_soma, v_saida);
  ELSE
    n_soma := 1;
    --Chamada da procedure
    p_exemplo(n_soma, v_saida);
  END IF;

  --Exibição da mensagem de retorno da procedure
  dbms_output.put_line(v_saida);

END;
```

Figura 17 - Exemplo de chamadas de subprogramas repetidas.
Fonte: Autoria própria.

3.1.4 Packages

Tanto as *procedures* quanto as *functions* podem ser armazenadas em bibliotecas chamadas *packages* na PL/SQL. A *package* pode ser definida como "uma coleção de objetos PL/SQL que estão agrupados cujos principais benefícios são ocultação de informações, design orientado a objetos, melhor desempenho e servir como objeto de persistência entre transações." (DAWES; FEUERSTEIN; PRIBYL, 2008). Além das *procedures* e *functions*, outros elementos podem ser colocados dentro de uma *package* como: constantes, variáveis, cursores, nomes de exceções (*exception names*), e tipos definidos.

Uma *package* é constituída por dois objetos, o objeto de especificação (*package specification*) e o de corpo (*package body*). A especificação é basicamente uma API, nela são colocados todos os objetos públicos, ou seja, que irão ser acessados do exterior da *package*. Além disso ela também fornece todas as informações que um desenvolvedor necessita para utilizar os objetos de dentro da *package*, como por exemplo os parâmetros de entrada e saída de uma *procedure*. A sintaxe da especificação de uma *package* pode ser vista na Figura 18 assim como um exemplo na Figura 19.

```
CREATE [OR REPLACE] PACKAGE package_name
[ AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
   [definitions of public TYPEs
   ,declarations of public variables, types, and objects
   ,declarations of exceptions
   ,pragmas
   ,declarations of cursors, procedures, and functions
   ,headers of procedures and functions]
END [package_name];
```

Figura 18 - Sintaxe da especificação de uma package
Fonte: (DAWES; FEUERSTEIN; PRIBYL, 2008).

```
CREATE OR REPLACE PACKAGE pac_exemplo IS

  FUNCTION f_exemplo RETURN DATE;
  PROCEDURE p_exemplo(n_valor IN NUMBER, v_msg_saida OUT VARCHAR2);

END pac_exemplo;
```

Figura 19 - Exemplo de uma especificação de package.
Fonte: Autoria própria.

O corpo de uma *package* pode conter objetos privados e todo o código necessário para implementar *procedures*, *functions* e cursores definidos na especificação da *package*. A Figura 20 ilustra a sintaxe do corpo de uma *package* e um exemplo de corpo de *package* pode ser visto na Figura 21.

```
CREATE [OR REPLACE] PACKAGE BODY package_name
   { IS | AS }
   [definitions of private TYPEs
   ,declarations of private variables, types, and objects
   ,full definitions of cursors
   ,full definitions of procedures and functions]
[BEGIN
   executable_statements
[EXCEPTION
   exception_handlers]]
END [package_name];
```

Figura 20 - Sintaxe do corpo de uma package.
Fonte: (DAWES; FEUERSTEIN; PRIBYL, 2008).

```
CREATE OR REPLACE PACKAGE BODY pac_exemplo IS

  FUNCTION f_exemplo RETURN DATE IS
    dt_dia DATE;
  BEGIN
    SELECT trunc(SYSDATE) INTO dt_dia FROM dual;
    RETURN(dt_dia);
  END f_exemplo;

  PROCEDURE p_exemplo(n_valor IN NUMBER, v_msg_saida OUT VARCHAR2) IS
  BEGIN

    INSERT INTO nome_tabela
      (dia, valor)
    VALUES
      (pac_exemplo.f_exemplo, n_valor);
    v_msg_saida := 'Comandos executados com sucesso.';
    COMMIT;

  EXCEPTION
    WHEN no_data_found THEN
      v_msg_saida := 'Erro ao buscar dados.';
      RETURN;

    WHEN OTHERS THEN
      v_msg_saida := 'Erro na execução.';
      RETURN;
  END;
END pac_exemplo;
```

Figura 21 - Exemplo de um corpo de package.
Fonte: Autoria própria.

3.1.5 Triggers

Um objeto de extrema importância em bancos de dados é a *trigger*, que funciona como um gatilho que é acionado em um evento predefinido. Apesar da maior parte das bibliografias relacionar esses eventos apenas com comandos DML, na PL/SQL elas também podem ser ativadas por comandos DDL, DCL e eventos do banco de dados (*database events*). Segundo (DAWES; FEUERSTEIN; PRIBYL, 2008) os eventos do banco que podem ativar uma *trigger* são: *analyze, associate statistics, audit, comment, db_role_change, disassociate statistics, noaudit, rename, servererror, logon, logoff, startup, shutdown e suspend*. Dos diversos benefícios que se tem ao utilizar *triggers*, BIANCHI (2013) cita: "impor uma integridade de dados

mais complexa do que uma restrição check, definir mensagens de erro personalizadas, manter dados desnormalizados e comparar a consistência dos dados". As *triggers* de eventos DML possuem uma sintaxe diferente das outras *triggers*. A Figura 22 mostra a sintaxe das *triggers* de DML e a Figura 23 é um exemplo desta. As *triggers* de outros tipos de eventos tem sua sintaxe ilustrada na Figura 24 e um exemplo na Figura 25.

```
CREATE [OR REPLACE] TRIGGER trigger_name
{ BEFORE | AFTER | INSTEAD OF | FOR } trigger_event
   ON {table_or_view_reference |
     NESTED TABLE nested_table_column OF view}
     [REFERENCING [OLD AS old] [NEW AS new]
       [PARENT AS parent]]
[FOR EACH ROW ]
[FOLLOWS other_trigger] [DISABLE]
[COMPOUND TRIGGER]
[WHEN trigger_condition]
trigger_body;
```

Figura 22 - Sintaxe de utilização de uma trigger de eventos DML.
Fonte: (DAWES; FEUERSTEIN; PRIBYL, 2008).

```
CREATE OR REPLACE TRIGGER t_exemplo
  BEFORE INSERT ON nome_tabela

DECLARE
  a VARCHAR2(50);
BEGIN
  a := 'Trigger executada';
  dbms_output.put_line(a);
END t_exemplo;
```

Figura 23 - Exemplo de trigger para evento DML.
Fonte: Autoria própria.

```
CREATE [OR REPLACE] TRIGGER trigger_name
{ BEFORE | AFTER } trigger_event
   ON [ DATABASE | schema ]
 [FOLLOWS other_trigger][DISABLE]
[WHEN trigger_condition]
trigger_body;
```

Figura 24 - Sintaxe de utilização de uma trigger de eventos não DML.
Fonte: (DAWES; FEUERSTEIN; PRIBYL, 2008).

```
CREATE OR REPLACE TRIGGER t_exemplo_2
   BEFORE CREATE ON database

DECLARE
   a VARCHAR2(50);
BEGIN
   a := 'Objeto criado';
   dbms_output.put_line(a);
END;
```

Figura 25 - Exemplo de trigger para eventos não DML.
Fonte: Autoria própria.

A PL/SQL possui diversos outros objetos como *views* (visualizações), *materialized views* (visualizações materializadas), *indexes* (índices), *directories* (diretórios), *synonyms* (sinônimos), *jobs* (tarefas), *database links* (conexões entre bancos de dados), etc. Porém esses objetos não serão utilizados neste trabalho.

3.2 Compilação na PL/SQL

Conforme FEUERSTEIN (2009), obter informações sobre o funcionamento interno da PL/SQL, como por exemplo as etapas realizadas pelo compilador, é uma tarefa extremamente difícil. O autor afirma que somente pessoas pertencentes ao quadro funcional da Oracle as contém. As informações contidas nesta subseção foram extraídas do livro do FEUERSTEIN (2009).

Inicialmente, para que seja possível uma compilação na PL/SQL é necessária uma conexão aberta. Assim que a conexão estiver aberta é possível enviar o código

a ser compilado para o servidor. Quando o código chega no servidor o compilador PL/SQL verifica se o mesmo possui instruções SQL a serem executadas ou se é puramente PL/SQL. Caso o código seja somente PL/SQL é feita uma checagem sintática e logo após a forma compilada do código, o *bytecode*, é colocada em uma área de memória compartilhada. Em caso de problemas em uma dessas fases o compilador irá retornar uma mensagem de erro.

Caso o código possua instruções SQL, antes de fazer a geração do *bytecode* o compilador PL/SQL envia todo o código SQL a um compilador SQL, que irá iniciar a fase de análise da execução SQL. Essa fase envolve resolver expressões, executar verificações de semântica e sintaxe, executar a resolução de nomes e determinar um plano de execução ideal. Durante a fase de análise é possível que o compilador SQL encontre instruções PL/SQL e se comunique com o compilador PL/SQL para resolução destas. Após a fase de análise é feita a substituição de variáveis e a execução dos comandos SQL.

Por último a máquina virtual PL/SQL (PVM) interpreta o bytecode e retorna sucesso ou falha. A PVM, também conhecida como PL/SQL *runtime engine*, é um componente do BD que transforma o *bytecode* em linguagem de máquina e faz chamadas para o servidor do BD. Por fim a PVM retorna o resultado para o ambiente que fez a requisição.

3.3 PL/SQL Developer

O PL/SQL Developer é um poderoso Ambiente de Desenvolvimento Integrado (IDE - *Integrated Development Enviroment*) criado pela Allround Automations[7] em 1997. Segundo a enciclopédia da revista PC MAGAZINE[8], um IDE pode ser definida como "um conjunto de programas executados a partir de uma única interface de usuário".

O PL/SQL Developer é um IDE que foi especialmente criado para o desenvolvimento de unidades de programação armazenadas no banco de dados da Oracle. Atualmente na versão 10.0, disponibilizada em fevereiro de 2013, o software

[7] http://www.allroundautomations.com/
[8] http://www.pcmag.com/encyclopedia/term/44707/ide

conta com funcionalidades como: janela de comandos, editor sql, janela de testes com *debugger* integrado, *code folding* (dobramento de código) , suporte unicode, *split editing*, sql *exporter*, *text and files importer*(importador de textos e arquivos), explain plan, gerador de relatórios, múltiplas conexões, identador de código configurável, *browser* de arquivos, conexões, objetos e templates, *query builder*, otimizador de performance, gerador de documentação, lista de tarefas, múltiplos *workspaces* (áreas de trabalho), assistente de código, acesso ao DBMS scheduler, comparador de arquivos e objetos, além de diversas extensões por disponibilizadas através de *plugins*.

Segundo a documentação disponibilizada pela Allround Automations, *plugins* são arquivos DLL (*Dynamic-link library*) que servem para adicionar novas funções ao *software*. Para a criação de *plugins* que funcionem no PL/SQL Developer se faz necessária uma linguagem de programação que possa compilar arquivos do tipo DLL. Além disso deve ter no mínimo três funções básicas para que o PL/SQL Developer identifique a DLL como um *plugin*. As três funções básicas são: IdentifyPlugIn, CreateMenuItem e OnMenuClick.

Inicialmente o software PL/SQL Developer não possui integração com nenhum software de controle de versão. Porém, através de *plugins*, é possível realizar a conexão com quatro softwares gerenciadores de versão, sendo eles: Microsoft Visual SourceSafe, Merant, MKS Source Integrity e o Subversion.

4 PROPOSTA DE SOLUÇÃO

Nos capítulos anteriores foram apresentados os comandos básicos de manipulação de Banco de dados do SGBD Oracle, bem como o seu processo de compilação de objetos. Este capítulo descreve esse cenário na empresa Rio Grande Energia (RGE) e seus problemas atuais. Na sequência é apresentada uma proposta de solução para este cenário.

4.1 Cenário Atual

Na RGE o SGBD do sistema de faturamento é o Oracle na versão 10g. Para desenvolvimento *de functions, procedures, packages, views* e *triggers* é utilizado o software PL/SQL Developer. Para o controle dos programas que serão passados para o ambiente de produção é utilizado o Microsoft Visual SourceSafe. O PL/SQL Developer e o SourceSafe se integram através de um *plugin* disponibilizado pela Allround Automations.

No ambiente de desenvolvimento todos os desenvolvedores acessam o SGBD com o mesmo usuário. Já no SourceSafe cada desenvolvedor precisa estar logado, via *plugin* do PL/SQL Developer, com seu próprio usuário. O SourceSafe está configurado para trabalhar como *file locking*. Neste caso, como objetos estão armazenados dentro do banco de dados, existe a possibilidade de que um desenvolvedor que não esteja logado no SourceSafe altere o código fonte de um objeto que está bloqueado pelo *check out* de outro desenvolvedor. Os dois fluxos de compilação de objeto no BD são ilustrados na Figura 26, a qual contém o fluxo com subprocessos. O Usuário 1 executa o fluxo completo e correto, que deveria ser executado por todos os desenvolvedores. O Usuário 2 executa um fluxo errado, porém possível e extremamente comum. Esse é o fluxo que este trabalho tem por objetivo impossibilitar.

As Figuras 27, 28, 29 e 30 representam os subprocessos relacionados a Figura 26.

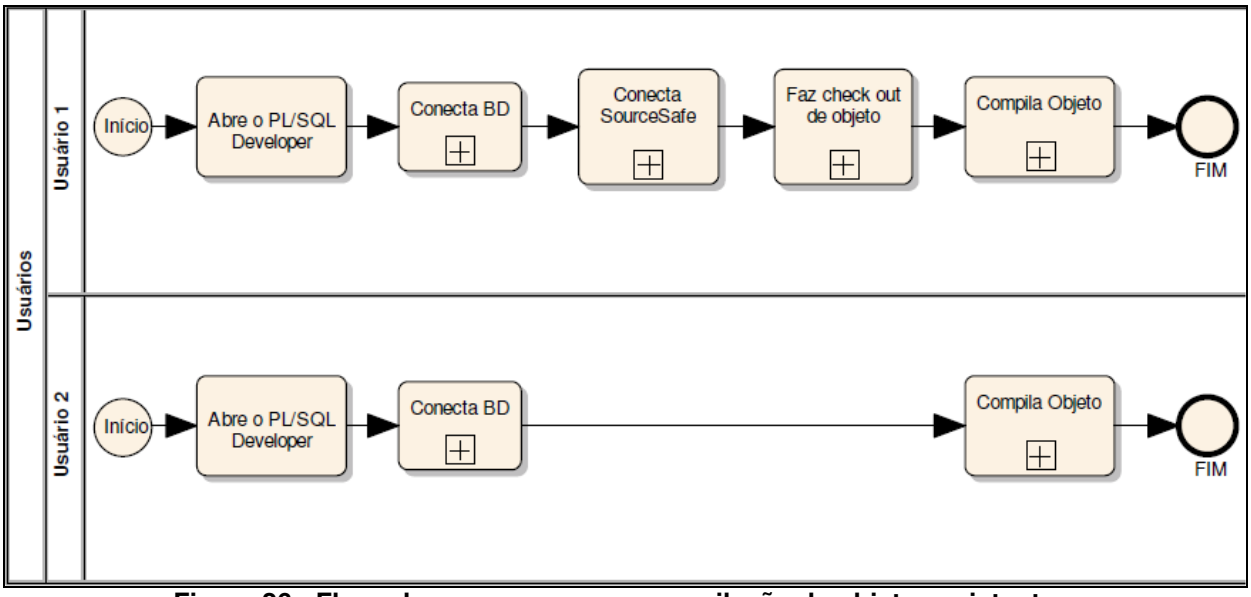

Figura 26 - Fluxo de processo para a compilação de objetos existentes.
Fonte: Autoria própria.

A figura 27 ilustra o subprocesso Conecta BD. Primeiramente o usuário, deverá selecionar o BD que deseja conectar, depois deverá entrar com seu usuário e senha. O PLSQL Developer irá receber estas informações e irá encaminhá-las para o SGBD. O SGDB por sua vez, irá tentar conectar no BD desejado, se conseguir irá enviar uma mensagem para o usuário informando a conexão, caso contrário o processo irá finalizar.

Se o usuário conseguiu se conectar ao BD pelo subprocesso da figura 27, o próximo passo é a realizar a conexão com o SourceSafe. Este subprocesso é ilustrado na figura 28. Este subprocesso inicia com a solicitação do usuário de abertura de um projeto do SourceSafe no PLSQL Developer. Ao abrir este projeto o PLSQL Developer irá solicitar a entrada de um local de arquivo, usuário e senha. Após o usuário colocar os dados solicitados, o PLSQL Developer irá receber estas informações e irá enviá-las para o SourceSafe. O SourceSafe receberá as informações e irá tentar realizar a conexão. Se a conexão for estabelecida o usuário receberá mensagem informando a conexão, caso contrário o processo irá finalizar.

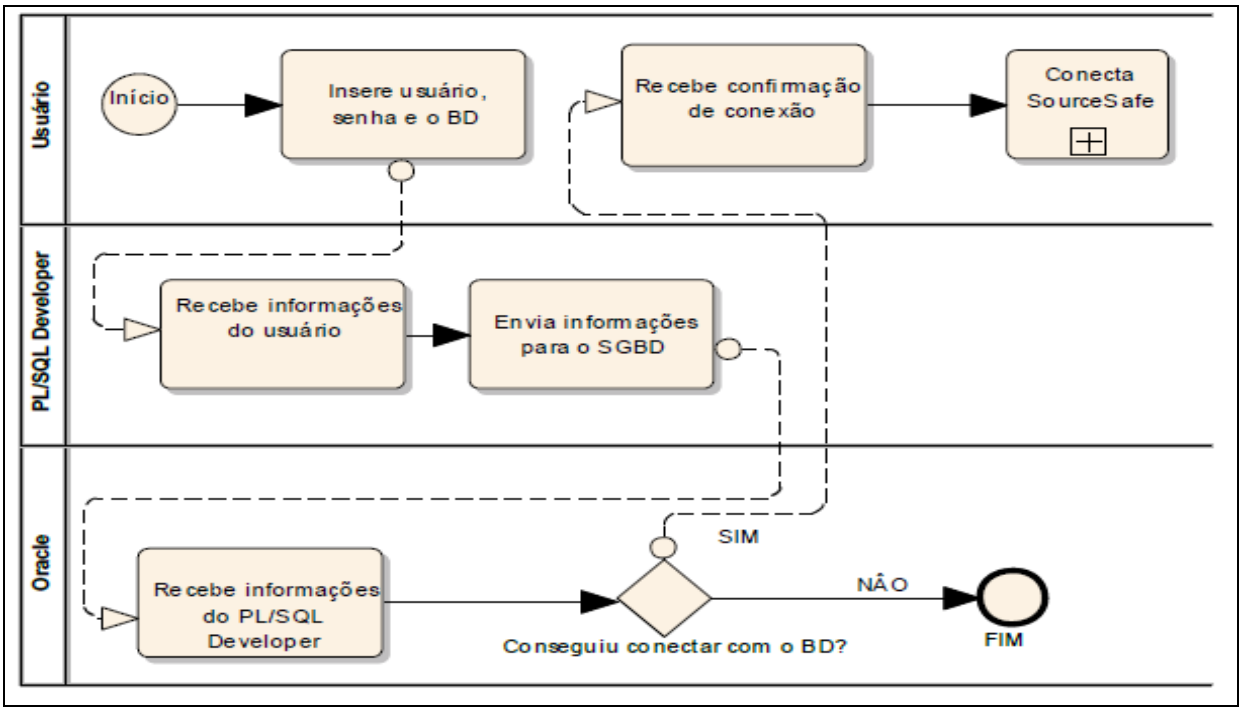

Figura 27 - Subprocesso Conecta BD.
Fonte: Autoria própria.

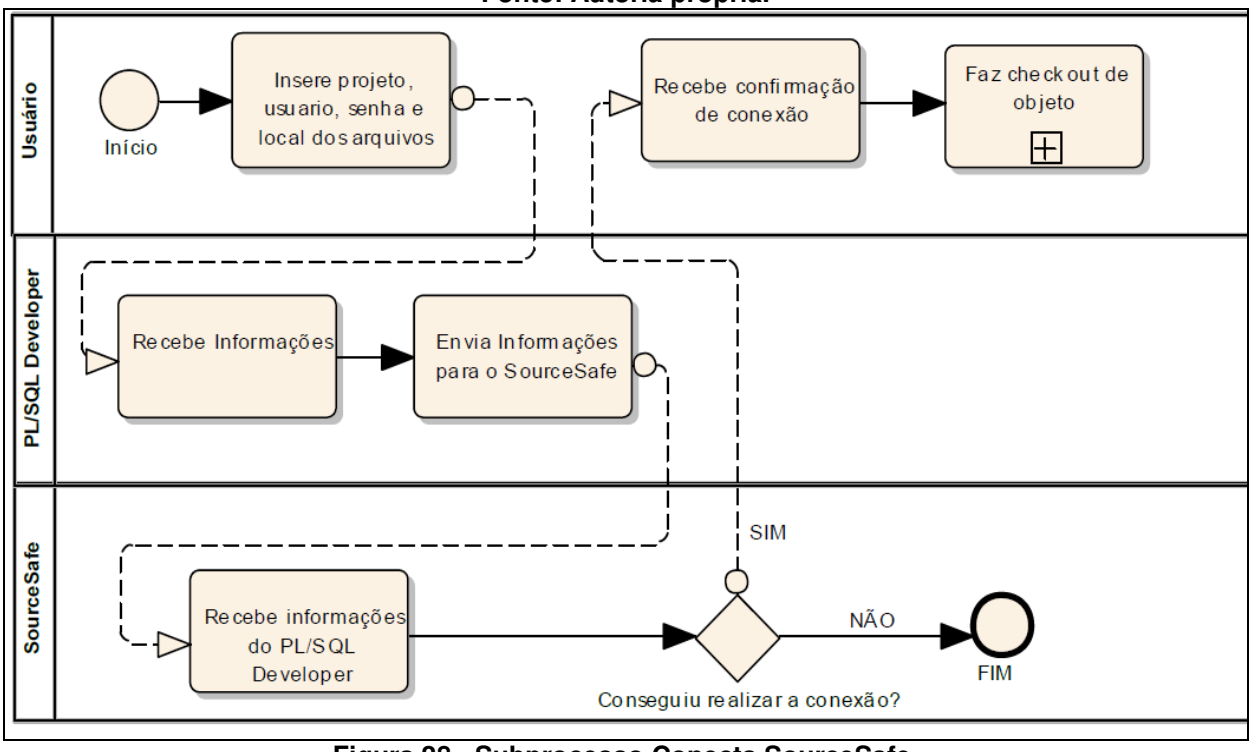

Figura 28 - Subprocesso Conecta SourceSafe.
Fonte: Autoria própria.

Após realizar a conexão com o SourceSafe o PLSQL Developer irá solicitar a escolha do objeto que se deseja trabalhar. Este subprocesso é o chamado de Check out de objeto, e é ilustrado pela figura 29. Após o usuário selecionar o objeto

desejado, o PLSQL Developer irá receber esta informação e irá encaminhá-la para o SourceSafe. O SourceSafe receberá a informação enviada pelo PLSQL Developer e irá verificar se o arquivo (objeto) está bloqueado, se estiver bloqueado o processo irá finalizar (pois outro usuário está utilizando este objeto), caso não esteja bloqueado o SourceSafe irá bloquear o arquivo e enviará mensagem para o usuário informando do bloqueio

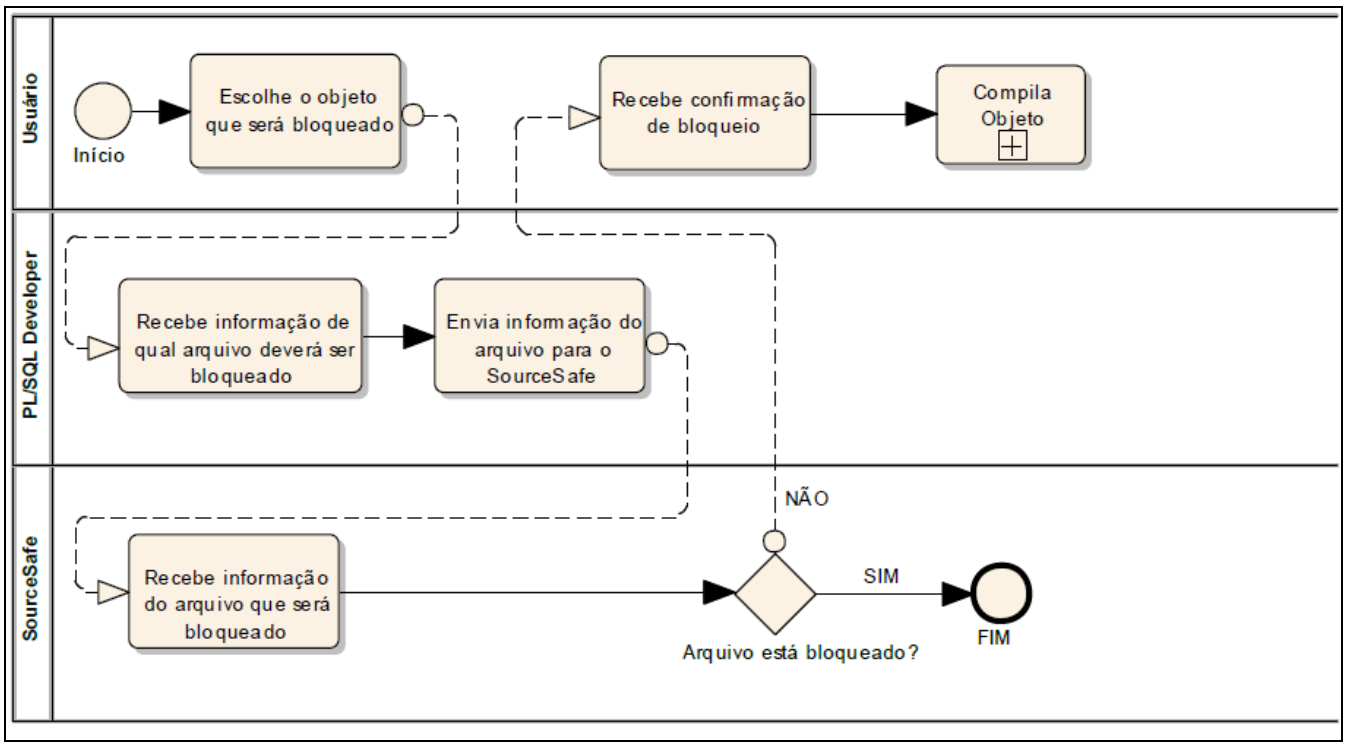

Figura 29 - Subprocesso Check Out de Objeto.
Fonte: Autoria própria.

Após ter o objeto desejado bloqueado para seu uso, o usuário irá executar os passos do subprocesso compila objeto. O usuário irá utilizar o editor e fará as alterações necessárias no arquivo (para atendimento de sua necessidade). Após fazer estas alterações o usuário irá requisitar a compilação do arquivo. O PLSQL Developer irá receber esta solicitação e irá repassá-la para o SGDB executar. Desta forma, o SGDB irá receber a solicitação e irá tentar compilar o arquivo, se conseguir irá enviar mensagem ao usuário informando a compilação, caso não consiga irá finalizar o processo. Desta forma o processo ilustrado na figura 26 chega ao fim

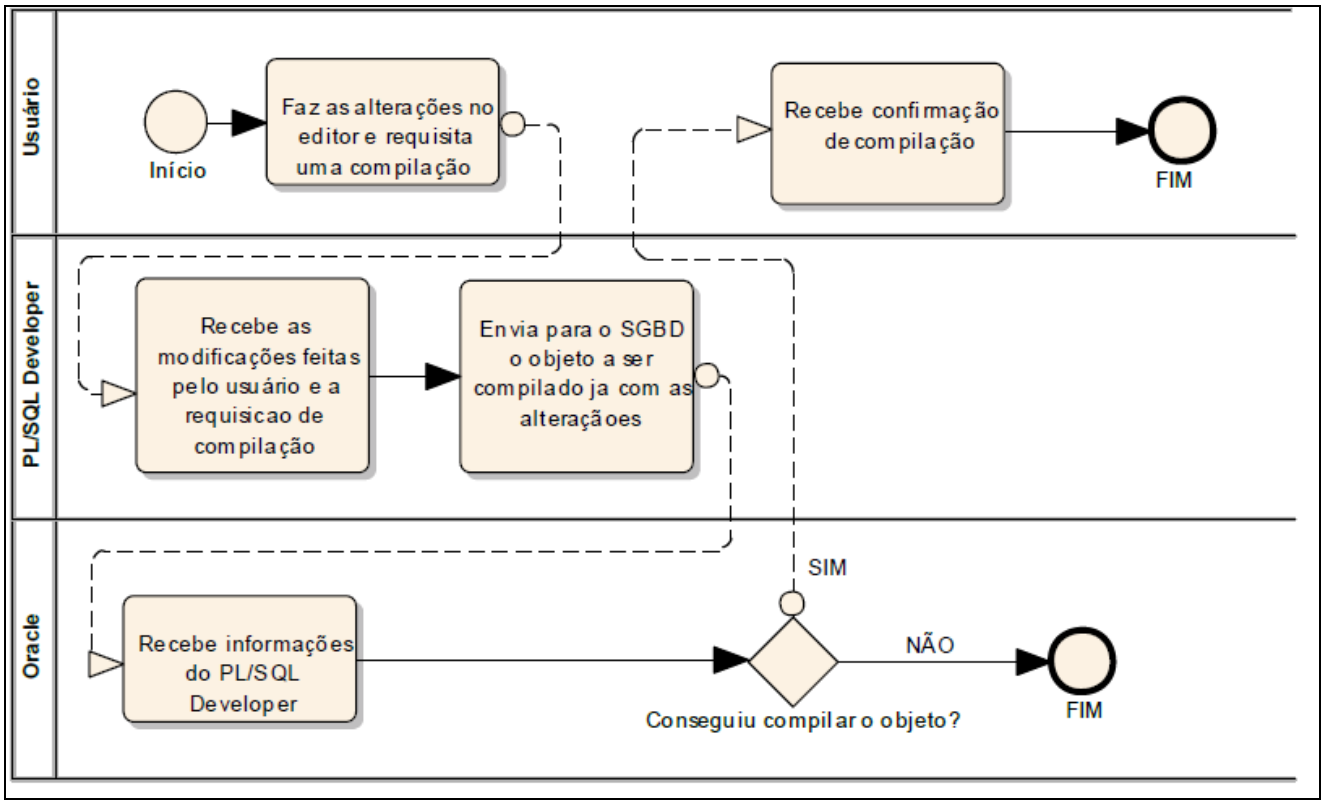

Figura 30 - Subprocesso Compila Objeto.
Fonte: Autoria própria.

4.2 Cenário Proposto

O cenário ideal seria um em que todos os desenvolvedores estivessem sempre logados no SourceSafe e que nenhum outro software, além do PL/SQL Developer, fosse utilizado para compilar objetos no SGBD. Como não é possível garantir esse cenário, o cenário a ser proposto possui duas etapas distintas. A primeira é, criar uma estrutura dentro do SGBD que garanta que apenas um desenvolvedor possa compilar um objeto dentro de um período de tempo. E a segunda parte é criar um novo *plugin* para o PL/SQL Developer que se integre ao SourceSafe e alimente essa nova estrutura. Estas etapas são explicadas a seguir.

4.2.1 Arquitetura de Software

A Figura 31 apresenta a visão de arquitetura de software. Esta demonstra que

o PL/SQL Developer e o SourceSafe irão fazer o controle dos programas que serão desenvolvidos fora do banco de dados. O PL/SQL Developer envia informações para o Oracle que terá a tarefa de verificar se as requisições recebidas são do usuário que tem o *check out* do objeto.

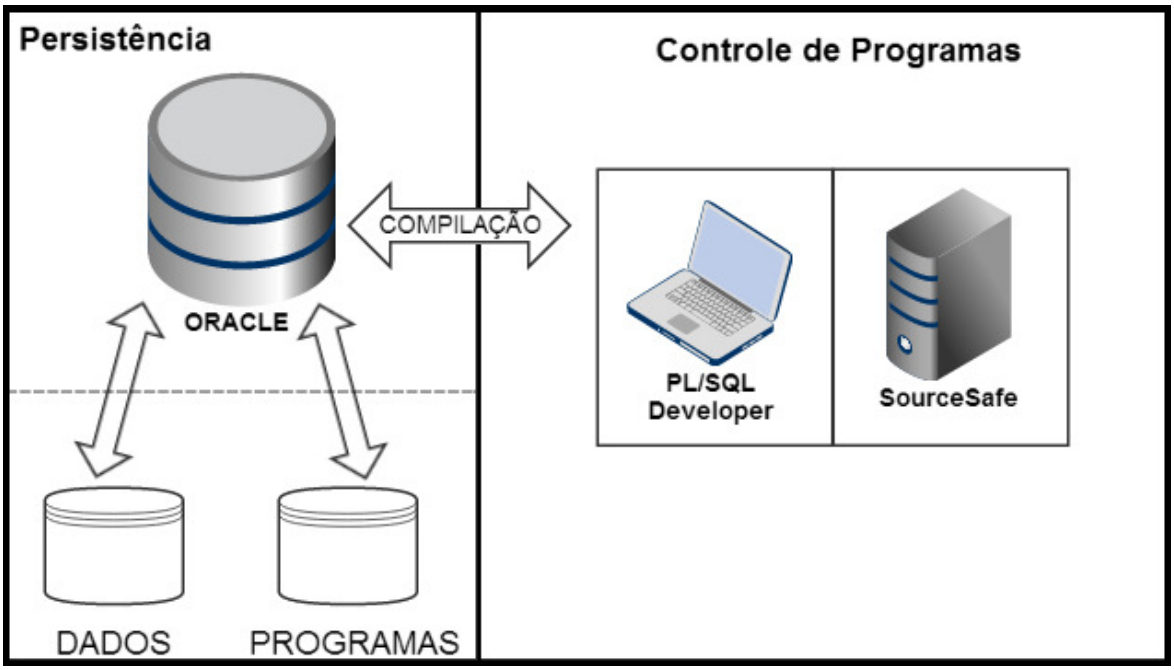

Figura 31 - Arquitetura de software.
Fonte: Autoria própria.

Para o SGBD fazer tal verificação será necessário definir como este estará estruturado a fim de armazenar tais informações a serem verificadas. A seguir é exposta a estrutura a ser criada no BD a fim de atender esta demanda

4.2.2 Estrutura BD

A estrutura do SGBD será composta por uma tabela, uma *trigger* e uma *package*. A tabela terá a informação dos objetos bloqueados e os usuários que o fizeram. A *trigger* irá verificar, a cada compilação, se o objeto que esta sendo compilado está na tabela e se o usuário que está requisitando a compilação é o mesmo que fez o bloqueio. A *package* será responsável por armazenar todos os

procedimentos e funções que se referem a alterações na estrutura.

A estrutura do BD é apresentada pela Figura 32, caso seja necessário serão criadas outras estruturas e novas *procedures* na *package*. Na *package* ficarão armazenadas todas as funcionalidades, tanto *procedures* quanto *functions*, utilizadas pela *trigger* e pelo *plugin*.

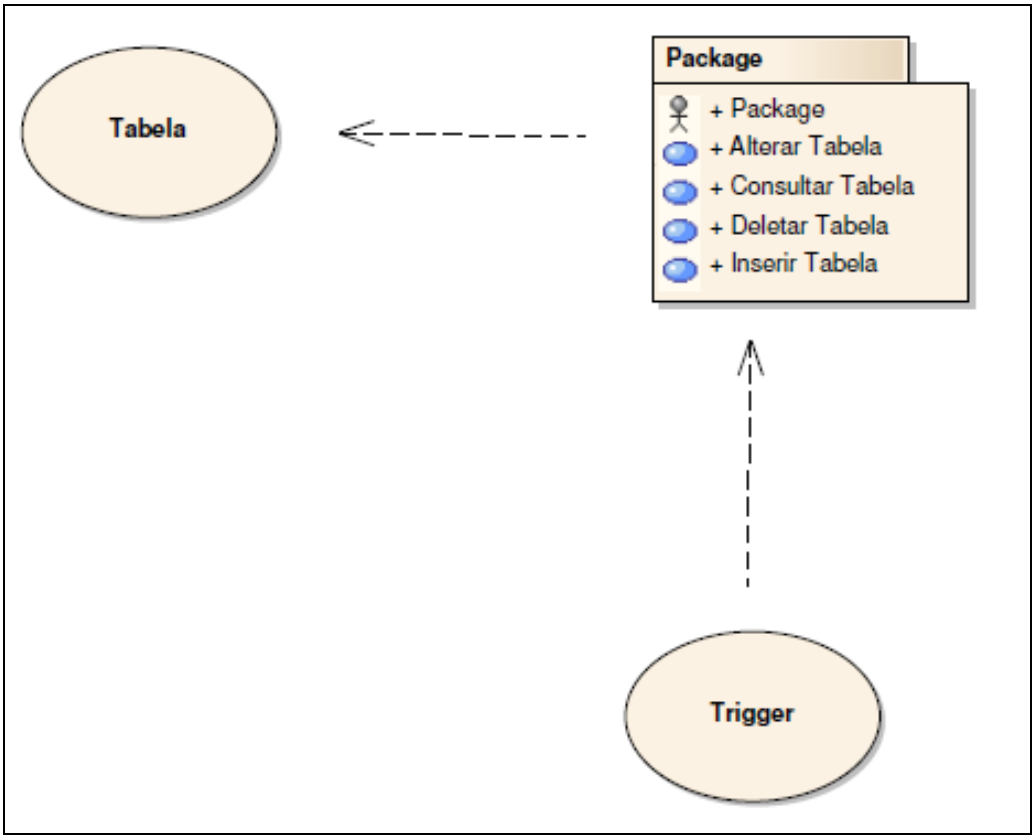

Figura 32 - Estrutura do BD.
Fonte: Autoria própria.

4.2.3 Plugin

O *plugin* para o PL/SQL Developer irá ser desenvolvido na linguagem LUA[9], este terá funções relacionadas a verificação de *check out* e *check in's.* Essas verificações irão utilizar os procedimentos e funções da *package* criada para alterar a estrutura.

[9] http://www.lua.org/

A linguagem LUA foi escolhida pois é uma linguagem de scripts projetada para estender aplicações (IERUSALIMSCHY, R., FIGUEIREDO, L., CELES, W, 2006).

O *plugin* do PL/SQL Developer terá diversas funcionalidades, além de alimentar a estrutura do BD ele terá que se conectar com o SourceSafe e prover uma interface para o usuário. As funcionalidades do *plugin* estão representadas na Figura 33. São elas:

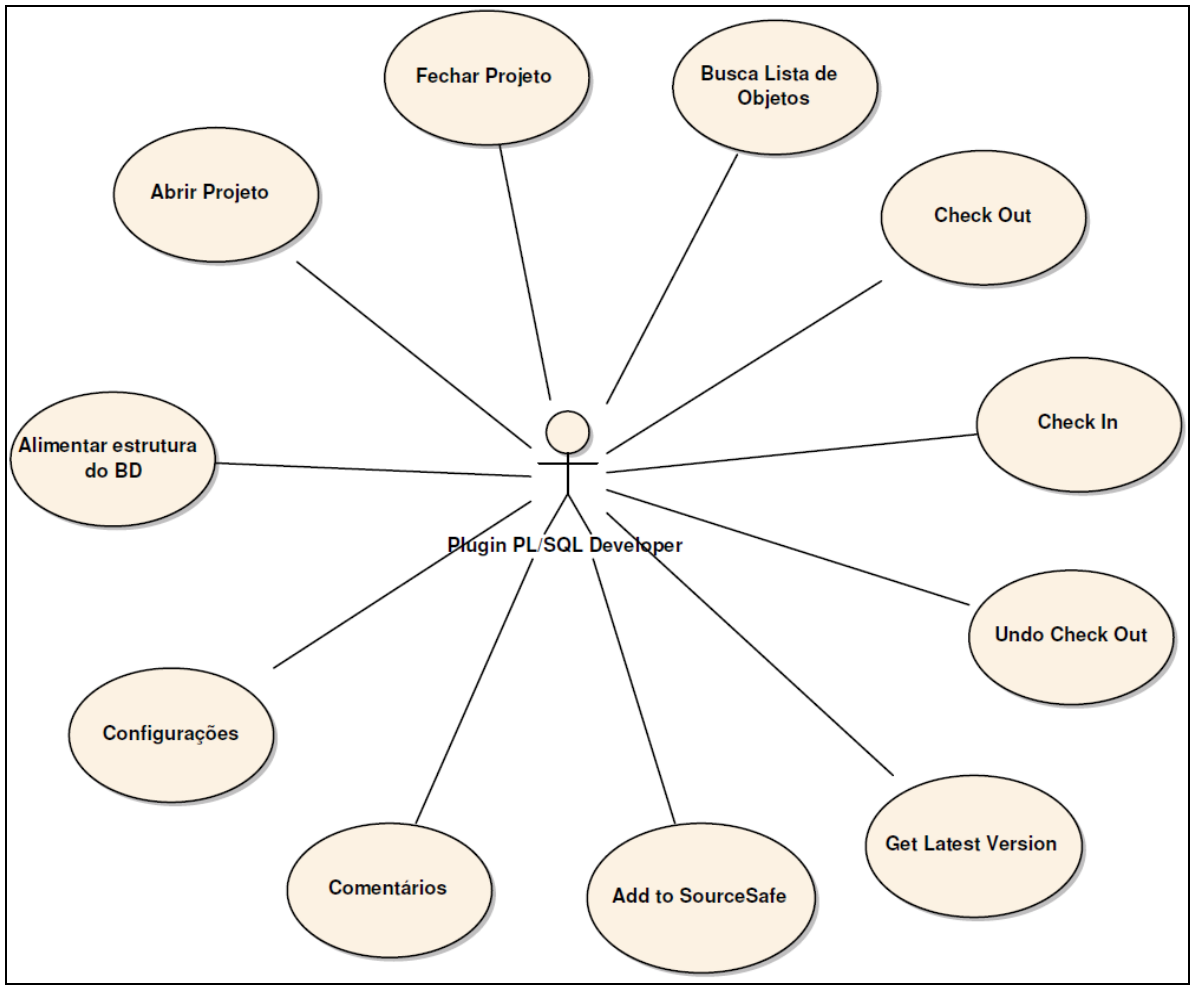

Figura 33 - Casos de uso do plugin.
Fonte: Autoria própria.

1. Abrir Projeto: Interface para colocar informações referentes ao projeto do SourceSafe com o qual o usuário deseja conectar. Um exemplo de tela pode ser visto na Figura 34.

2. Fechar Projeto: Desconecta usuário do projeto do SourceSafe.

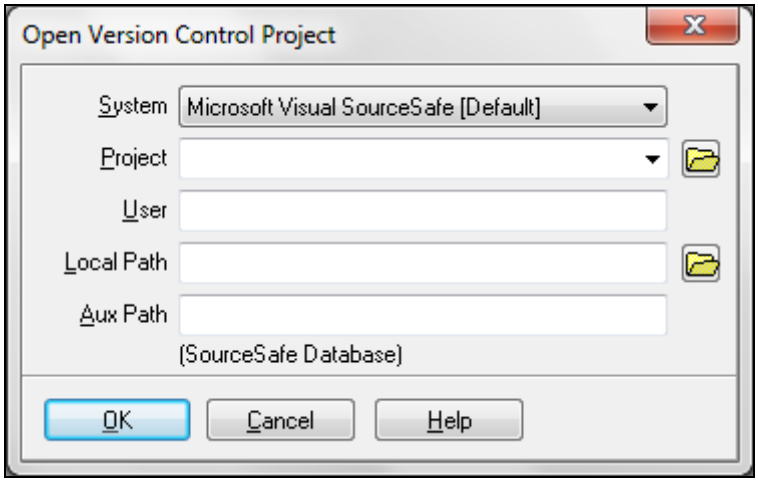

Figura 34 - Exemplo de tela para inserir informações do projeto.
Fonte: Autoria própria.

3. Busca Lista de Objetos: Deve buscar a relação de objetos do projeto no SourceSafe. Na Figura 35 um exemplo de lista de arquivos relativos ao projeto conectado e na Figura 36 a mesma lista vista diretamente no Visual SourceSafe Explorer.

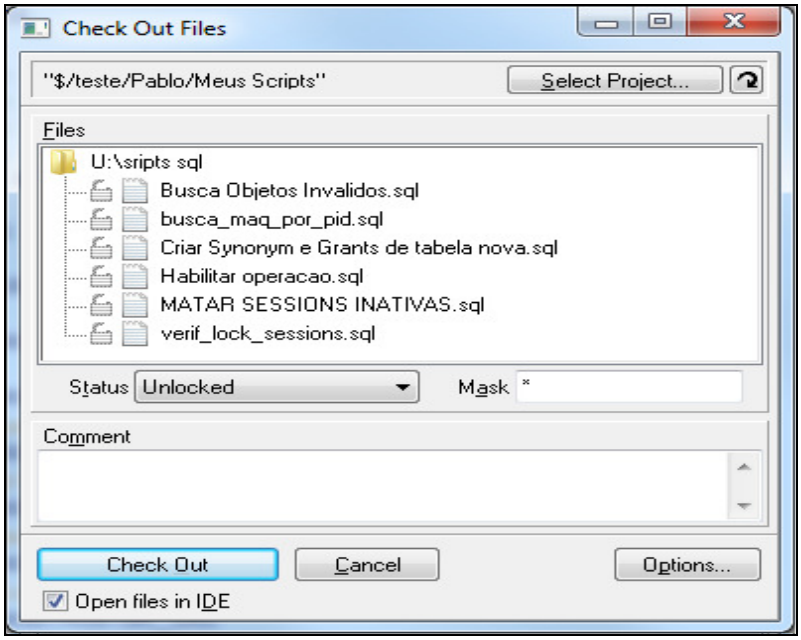

Figura 35 - Lista de objetos de um projeto.
Fonte: Autoria própria.

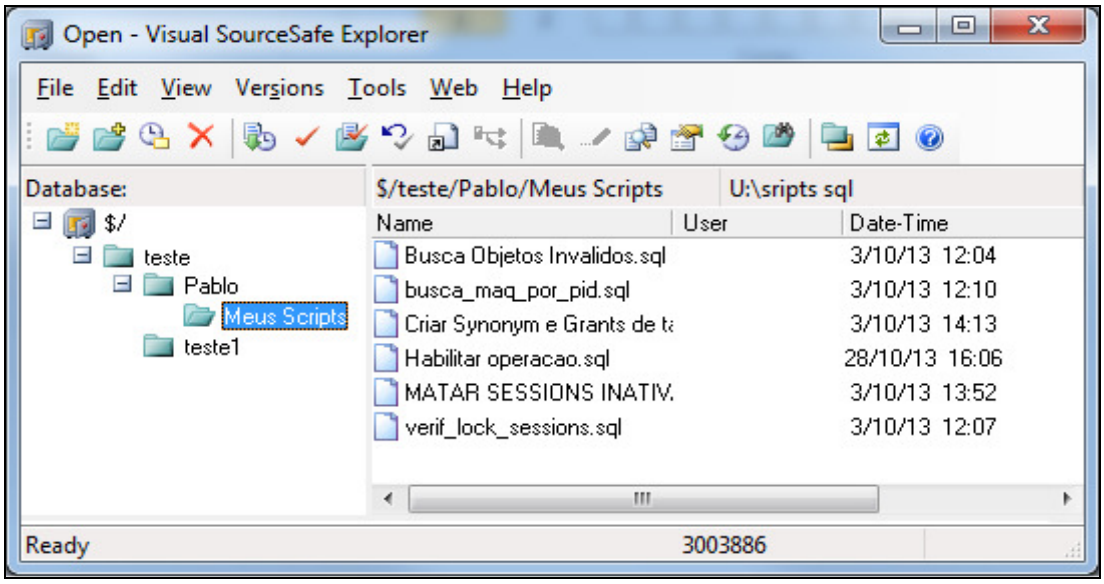

Figura 36 - Lista de arquivos vista no Visual SourceSafe Explorer
Fonte: Autoria própria.

4. Check Out: Ao selecionar um objeto listado na tela, exemplo Figura 35, deve permitir fazer o *check out* (bloqueio) do arquivo no SourceSafe.

5. Check In: Na tela de *check in* (envio de arquivo para o SourceSafe) deve trazer a lista apenas dos arquivos bloqueados pelo usuário. Para demonstração, Figura 37, foi feito o *check out* do arquivo "Busca Objetos Invalidos.sql".

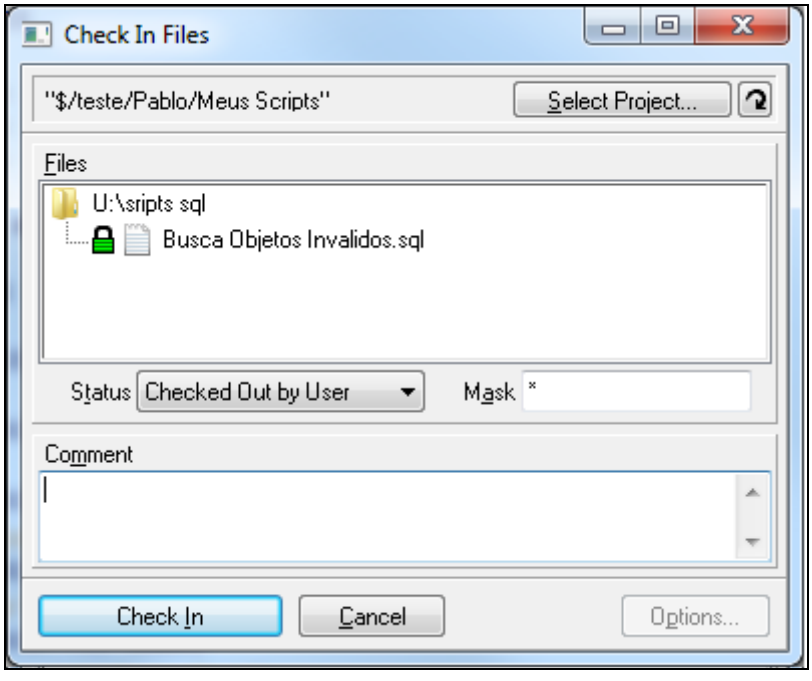

Figura 37 - Lista de objetos bloqueados para o usuário que está conectado no SourceSafe.
Fonte: Autoria própria.

6. Undo Check Out: Desfaz o *check out* (desbloqueio) de um arquivo.Exemplo de tela na Figura 38. Deve Trazer apenas os objetos bloqueados pelo usuário conectado no SourceSafe.

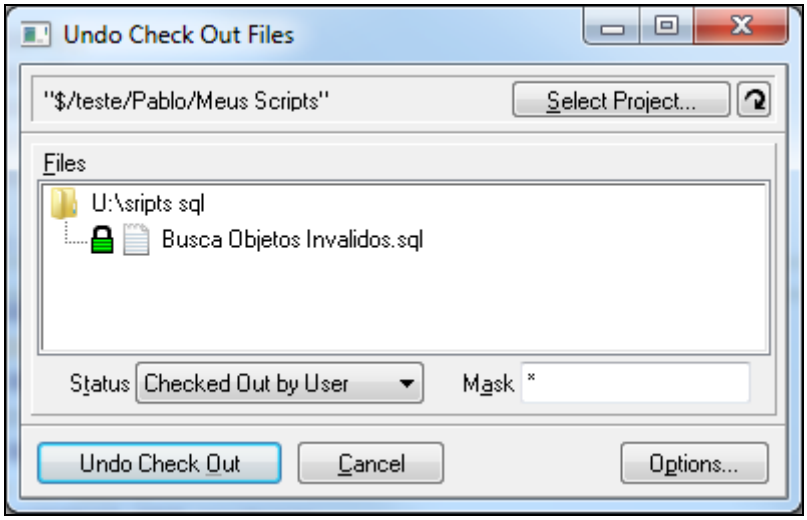

Figura 38 - Exemplo de tela para fazer o undo check out.
Fonte: Autoria própria.

7. Get Latest Version: Busca a última versão de um arquivo armazenada no SourceSafe, e joga no editor SQL do PL/SQL Developer.

8. Add to SourceSafe: Envia para o SourceSafe um arquivo salvo no diretório parametrizado na abertura do projeto (*Local Path*).

9. Comentários: Nas telas de *check out* (Figura 35), *check in* (Figura 37) deve ter um campo para comentários. Além dessas telas no momento de adicionar um novo arquivo ao SourceSafe (*Add to SourceSafe*) deve mostrar uma tela para colocar um comentário, como por exemplo, a Figura 39.

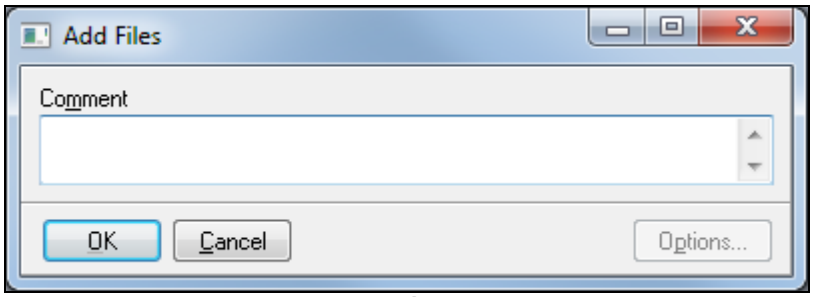

Figura 39 - Tela de comentários para novos arquivos.
Fonte: Autoria própria.

10. Configurações: A tela de configurações deverá ter toda a relação de informações do SourceSafe, como por exemplo, a obrigatoriedade de comentários nas operações. Além disso, deverá ter as configurações para utilização da estrutura do BD. Exemplos: *owner* (proprietário) da estrutura, exceções, usuários com acesso, necessidade de envio de *emails*, etc.

11. Alimentar estrutura do BD: Deve fazer as integrações entre os procedimentos definidos na *package* e os eventos do SourceSafe. Exemplo: Ao realizar um *check out* deve inserir informações na tabela e ao realizar *undo check out* ou *check in* remover as informações.

4.2.4 Protótipo

Um protótipo da estrutura já foi montado e está funcionando no SGBD de desenvolvimento do sistema de faturamento da RGE. Esse protótipo funciona por inserção manual em uma tabela, ou seja, sem integração com o SourceSafe. Nessa tabela é inserido o nome do objeto que está sendo alterado e a máquina que em que se está trabalhando (Figura 40 e Figura 41). A todo o momento objetos são compilados e existe uma *trigger* (Figura 42) que verifica se esse objeto existe na tabela. Caso exista, verifica se o nome da máquina que está solicitando a compilação é a mesma que está na tabela. Se não for a mesma, a compilação é bloqueada e o usuário recebe uma mensagem de erro.

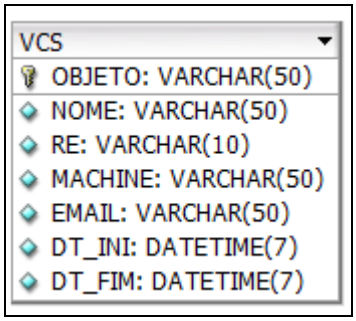

Figura 40 - Estrutura da tabela do protótipo.
Fonte: Autoria própria.

```
create table owner.VCS
(
  objeto  VARCHAR2(50) not null,
  nome    VARCHAR2(50) not null,
  re      VARCHAR2(10) not null,
  machine VARCHAR2(50) not null,
  email   VARCHAR2(50),
  dt_ini  DATE not null,
  dt_fim  DATE default '29991231' not null
);

comment on table owner.VCS
  is 'Tabela de VCS';

comment on column owner.VCS.objeto
  is 'Objeto que ira ficar bloqueado';
comment on column owner.VCS.nome
  is 'nome do usr';
comment on column owner.VCS.re
  is 'RE/Matricula do usr';
comment on column owner.VCS.machine
  is 'Maquina utilizada';
comment on column owner.VCS.email
  is 'email para receber notificações';
comment on column owner.VCS.dt_ini
  is 'Data inicio uso do objeto (check-out)';
comment on column owner.VCS.dt_fim
  is 'Data fim uso do objeto (check-in)';

alter table owner.VCS
  add constraint PK_VCS_OBJ primary key (OBJETO);

create public synonym VCS for owner.VCS;

grant select,insert,update,delete on owner.VCS to "users";
```

Figura 41 - Código da tabela do protótipo.
Fonte: Autoria própria.

```
create or replace trigger TCAD_VCS_DB
   before create or alter or drop on DATABASE

declare
   aux     number;
   v_mac   varchar2(30);
   v_user  varchar2(30);
   machine varchar2(30);

begin

   select count(*) into aux from vcs where vcs.objeto = ora_dict_obj_name;

   if aux > 0 then

      select sys_context('USERENV', 'TERMINAL'),
             sys_context('USERENV', 'OS_USER')
        into v_mac, v_user
        from dual;

      select machine into machine from vcs where vcs.objeto = ora_dict_obj_name;

      if trim(v_mac) = machine then
         raise_application_error(-20000, 'Objeto em check out!');
      end if;
   end if;
end;
```

Figura 42 - Trigger de eventos não DML, exemplo de protótipo.
Fonte: Autoria própria.

4.2.5 Cenário de Teste

Os cenários de teste tem como base o funcionamento da Figura 26.
Testes que devem funcionar:

- Fazer *check out* de um ou mais objetos através do PL/SQL Developer.
- Fazer o *check in* de um ou mais objetos através do PL/SQL Developer.
- Fazer o *undo check out* de um ou mais objetos através do PL/SQL Developer.
- Abrir a edição de um objeto com mais de um usuário.
- Fazer o *check out* e compilar um objeto (fluxo do Usuário 1 da Figura 26).

Testes que não devem funcionar:

- Fazer o *check out* de um objeto já bloqueado.
- Fazer o *check in* de um objeto em check out para outro usuário.
- Fazer o *undo check out* de um objeto em check out para outro usuário.
- Compilar um objeto sem ter feito *check out*.
- Compilar um objeto que esteja em *check out* para outro usuário (fluxo do Usuário 2 da Figura 26).

4.2.6 Estudo de Caso

Para execução dos cenários de teste, a equipe interna de desenvolvimento do sistema de faturamento da RGE irá utilizar a solução e reportará os problemas da estrutura e do *plugin* que serão desenvolvidos.

5 IMPLEMENTAÇÃO

Neste capítulo será explicado o que foi feito para atender a proposta de solução apresentada no Capítulo 4. A arquitetura do software será detalhada assim como o processo que o *plugin* executa para se integrar com o controlador de versão.

5.1 Arquitetura do Software

A Figura 31 ilustra uma arquitetura antiga e simplificada do software. Durante a implementação ela precisou ser modificada e agora pode ser vista com mais detalhes na Figura 43. O PL/SQL Developer continua como interface entre o Oracle e o SourceSafe. Para acesso ao software de controle de versão a IDE passa por três etapas, o LuaJIT, o *prompt* de comando do Windows e a versão oficial da Lua para Windows.

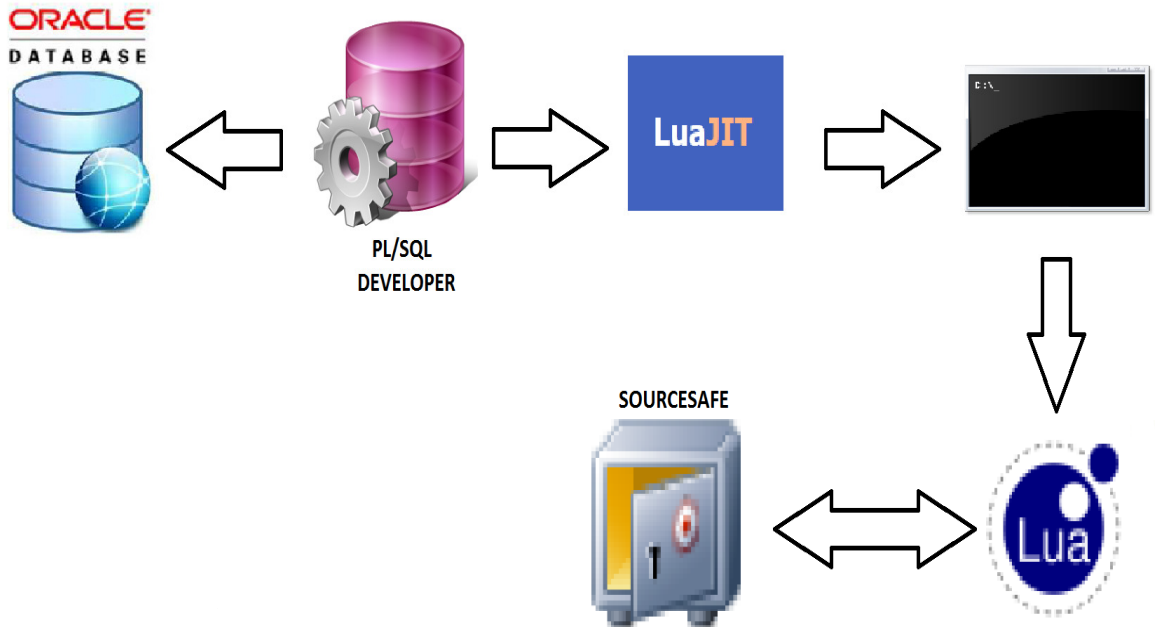

Figura 43 - Arquitetura de Software atualizada.
Fonte: Autoria própria.

5.1.1 Estrutura BD

A estrutura do banco de dados representada inicialmente pela Figura 32 foi atualizada e agora pode ser vista na Figura 44. Com relação ao que foi proposto pouco mudou foi criada mais uma tabela, a “VCS_VERSIONS”, para gravar o código dos objetos inválidos que forem compilados por outros usuários. A tabela “VCS” permanece com o mesmo propósito, o de armazenar os objetos que estão bloqueados. A estrutura das duas tabelas pode ser vista na Figura 45.

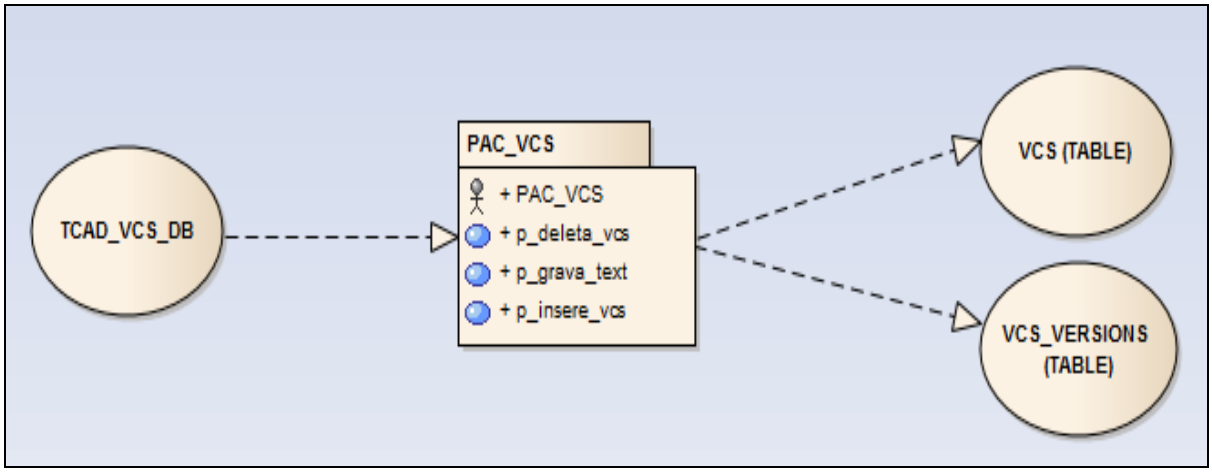

Figura 44 - Estrutura de objetos dentro do Oracle.
Fonte: Autoria própria.

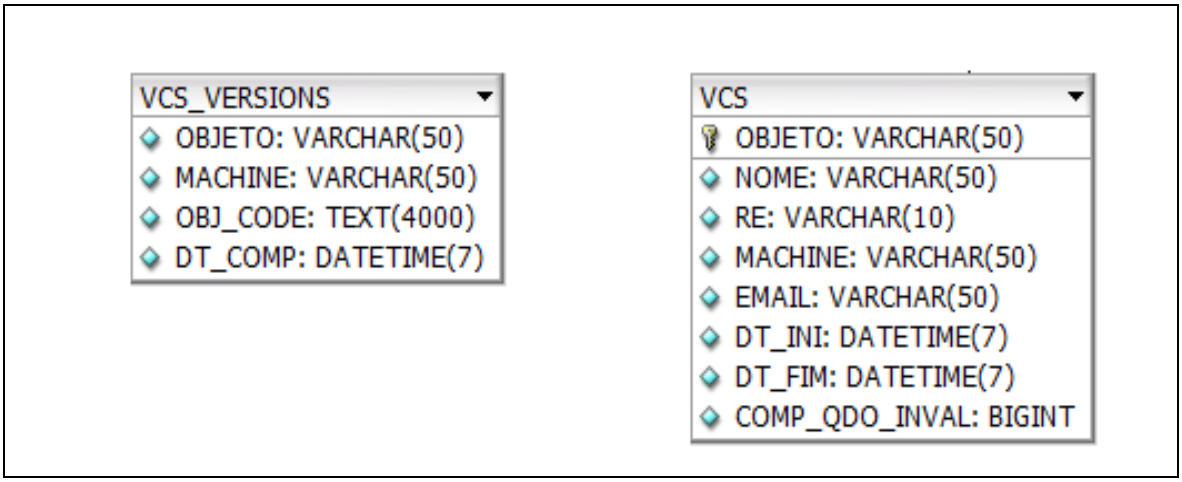

Figura 45 - Campos das tabelas utilizadas.
Fonte: Autoria própria.

5.1.2 Plugin LuaPlSql

O Plugin LuaPlSql oferece ao software PL/SQL Developer a possibilidade de programar novas funcionalidades utilizando a linguagem Lua. Para criar esse *plugin* o criador do projeto, utilizou o compilador LuaJIT. O LuaJIT é o responsável por executar os comandos recebidos do *plugin* LuaPlSql.

Como não foi possível fazer o LuaJIT chamar bibliotecas externas houve a necessidade de utilizar além do LuaJIT, a versão oficial da Lua.

Por padrão, o Plugin LuaPlSql reconhece apenas os arquivos de nome "main.lua" nas pastas abaixo de: <diretório instalação PL/SQL Developer>\PlugIns\lua* . A Figura 46 mostra a estrutura de diretórios utilizada.

Figura 46 - Diretórios lidos pelo Plugin LuaPlSql.
Fonte: Autoria própria.

Para cada diretório da Figura 46, o *plugin* irá ler os arquivos cujo nome é "main.lua". Para construir o menu de utilização do *plugin*, este irá verificar pelas chamadas de função "AddMenu". A Figura 48 possui uma chamada a essa função na linha 51. A Figura 47 mostra o menu criado pelo *plugin* no computador do autor.

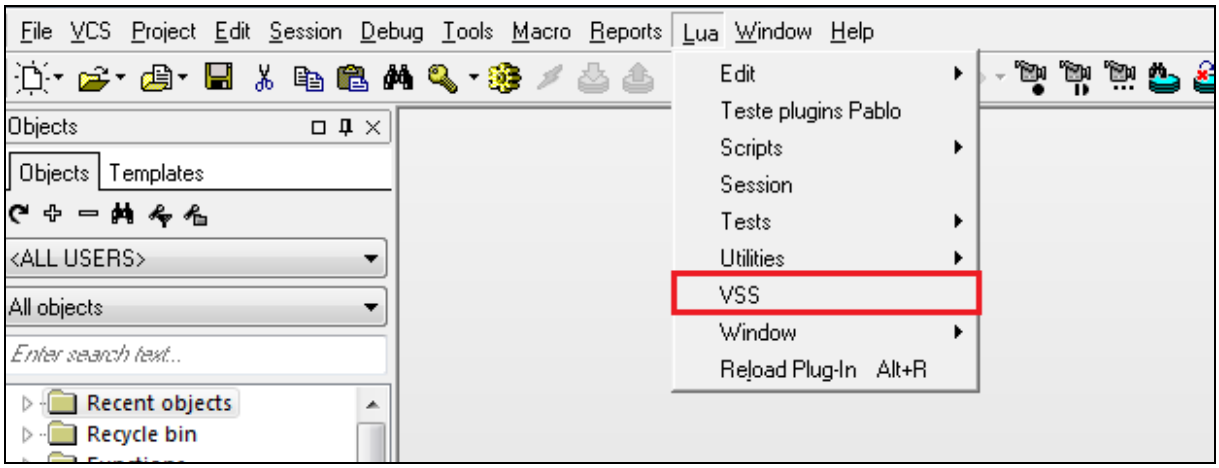

Figura 47 - Menu criado pelo Plugin LuaPlSql.
Fonte: Autoria própria.

Para iniciar a comunicação com o SourceSafe, o item “VSS” do menu “Lua” deve ser selecionado. A Figura 48 mostra o parte do código que é executado. Na linha 14 é feita uma chamada de execução do arquivo “vss_pl.lua” localizado na mesma pasta do arquivo “main.lua”, conforme mostra a Figura 49. É nesse momento que o LuaJIT faz uma chamada ao *prompt* de comandos do Windows, ilustrado na Figura 43.

```
1   local AddMenu = ...
2
3   local plsql = plsql
4   local SYS, IDE, SQL = plsql.sys, plsql.ide, plsql.sql
5
6   local ShowMessage = plsql.ShowMessage
7
8   local Gsub, Sub, Upper, Lower = string.gsub, string.sub, string.upper, string.lower
9
10  do
11      local function VSS()
12
13          --Chama o arquivo LUA a ser executado e joga o retorno em um arquivo
14          a = io.popen('"c://plsql developer/PlugIns/lua/VSS/vss_pl.lua"')
15          -- Lê o retorno do LUA no arquivo
16          b = (a:read("*a"))
17          --Fecha o arquivo
18          a:close()
19
20          reg_ss = 0
21          for linhas in string.gmatch(b, ".-\n") do --split \n
49      end
50
51      AddMenu(VSS, "&Lua / VSS")
52
53  end
```

Figura 48 - Trecho de código do arquivo main.lua da pasta VSS.
Fonte: Autoria própria.

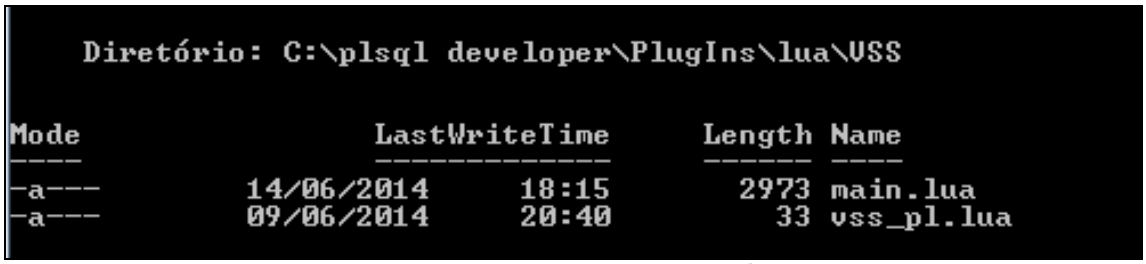

Figura 49 - Arquivos dentro do diretório VSS.
Fonte: Autoria própria.

O arquivo “vss_pl.lua” tem uma simples chamada da função “ini.ss()” da biblioteca “vss” conforme mostra a Figura 50. Esse comando chama a versão oficial da Lua. A biblioteca “vss.lua” esta localizada em <diretório instalação Lua>\lua como mostra a Figura 51.

```
vss = require "vss"
vss.ini_ss()
```

Figura 50 - Código do arquivo vss_pl.lua da pasta VSS.
Fonte: Autoria própria.

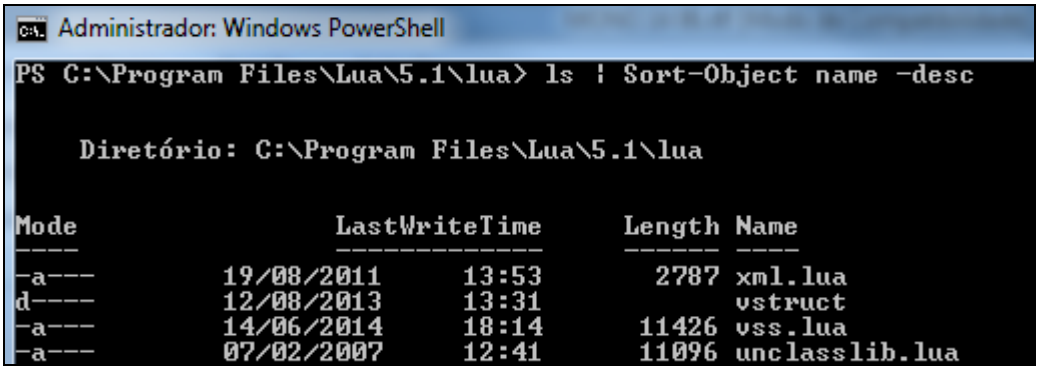

Figura 51 - Bibliotecas da Lua.
Fonte: Autoria própria.

A biblioteca “vss.lua” é responsável por criar a tela e realizar toda a comunicação com o SourceSafe. Seu código completo pode ser visto em anexo no final deste trabalho. A tela será melhor explicada na próxima sessão deste capítulo.

5.2 Tela

Na sessão 4.2.3 foi definido que, para cada caso de uso relacionado ao SourceSafe seria feito uma tela, porém no momento da implementação foi verificado que uma tela única com todas as funcionalidades divididas em botões iria agilizar o desenvolvimento e a utilização. Sendo assim, as Figuras 34 a 39 foram substituídas pela tela exibida na Figura 52. Esta foi desenvolvida em Lua utilizando a biblioteca IUP[10].

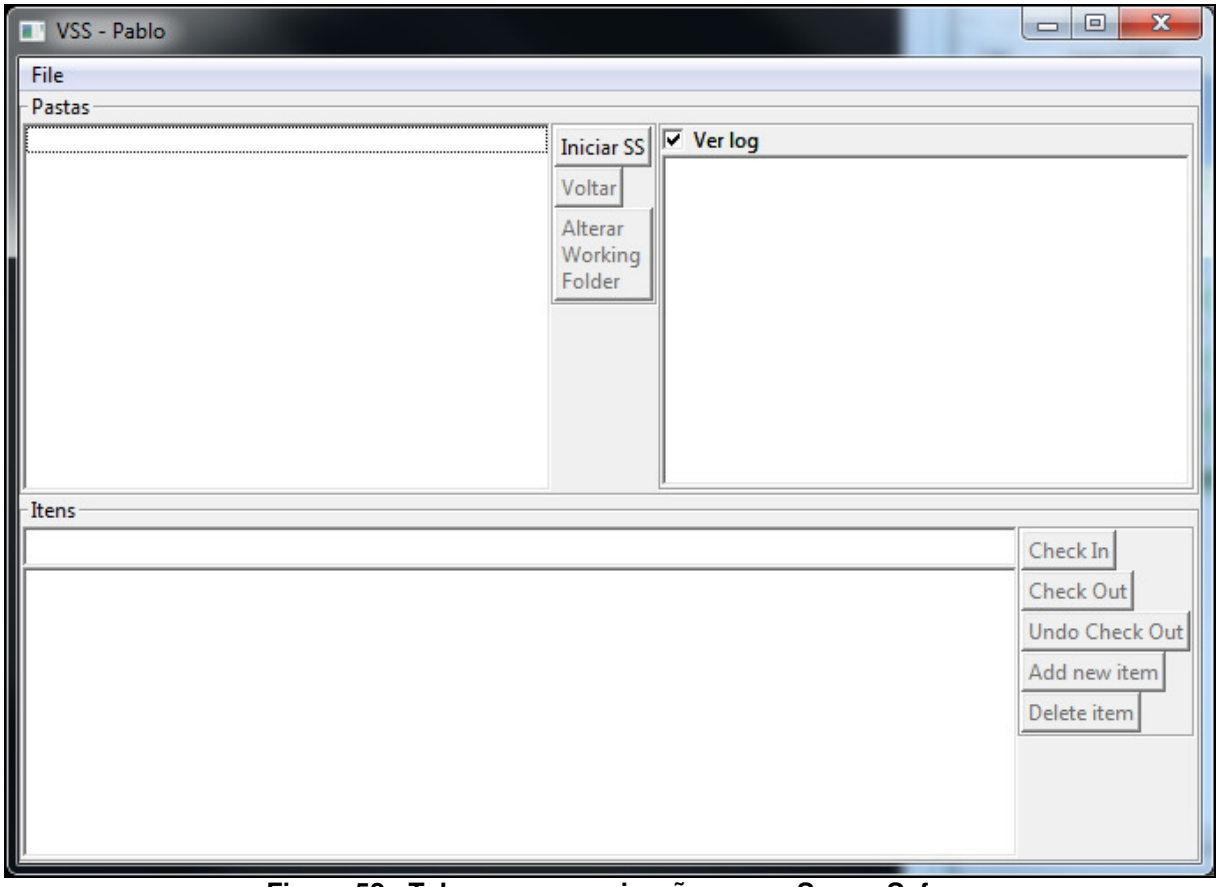

Figura 52 - Tela para comunicação com o SourceSafe.
Fonte: Autoria própria.

Para comunicação com o SourceSafe foi utilizada a ferramenta *Visual SourceSafe command line utility*, mencionada na sessão 2.4.1. A tela, Figura 52, foi dividida em três partes: a de navegação, a de Log e a de Itens.

A parte de navegação localizada no canto superior esquerdo, também sinalizada como “Pastas”, servirá para navegar dentro da arvore de diretórios do

SourceSafe além de definir onde o comando “Add new item” deverá inserir o novo item.

O registro de Log, localizado no canto superior direito, pode ser desabilitado ao clicar em “Ver log”. Esses serão os comandos que o *Visual SourceSafe command line utility* irá executar, além de serem os mesmos retornados para que o *plugin* do PL/SQL Developer envie para a estrutura localizada dentro do SGBD. Desativar a tela de Log não irá alterar o retorno para o *plugin*.

Os Itens vistos na parte inferior da tela são os que podem sofrer alterações, os seja, os arquivos. No momento em que um item for selecionado as ações que este pode sofrer são: *Check In*, *Check Out*, *Undo Check Out* e *Delete* Item. A Figura 53 mostra como uma Pasta e seus arquivos são exibidos pelo Visual SourceSafe Explorer e, a mesma pasta e itens sendo exibidos pela tela criada em Lua.

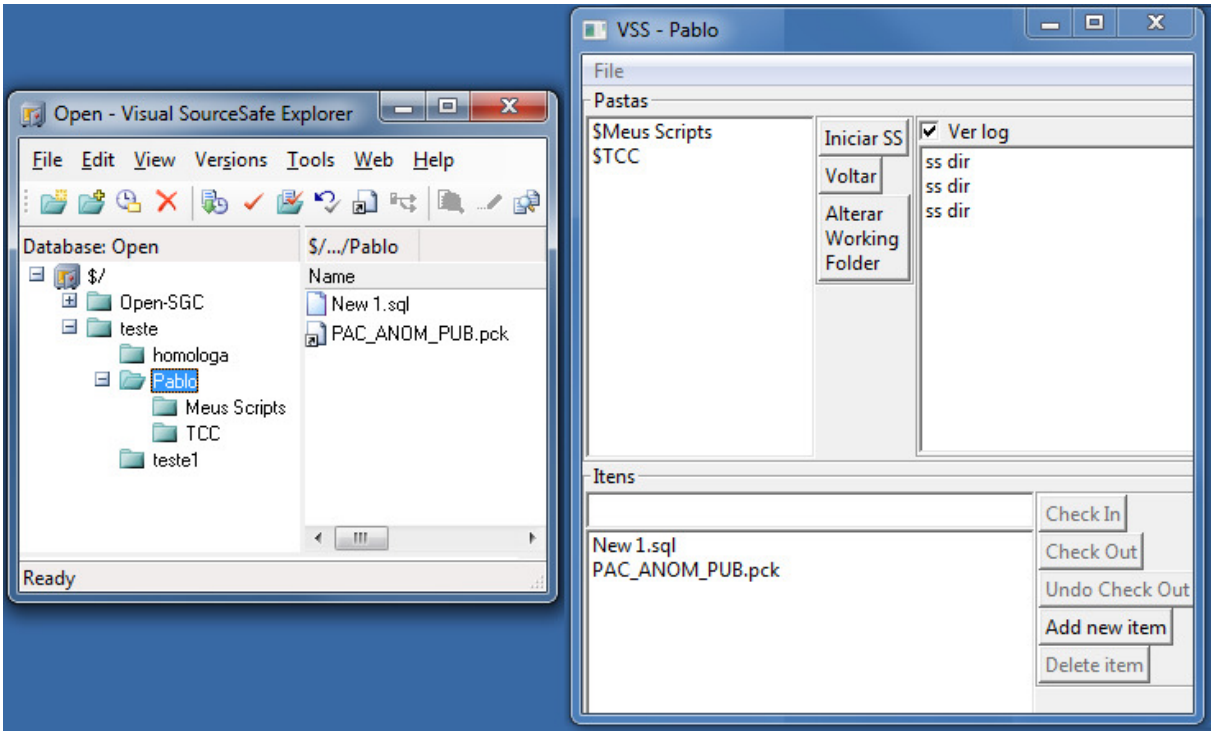

Figura 53 - Pasta sendo exibida pelo VSS Explorer e pela tela criada em Lua.
Fonte: Autoria própria.

5.3 Código fonte

Nesta sessão todos os arquivos que possuem código fonte criados para este trabalho serão apresentados:

• TCAD_VCS_DB: *Trigger* acionada nos eventos de criação, alteração e remoção do banco de dados, tem como objetivo ser o sistema de controle de versão dentro do Banco de Dados. Valida se os objetos podem ou não serem compilados. O código fonte da *trigger* pode ser visto no Apêndice A.

• VCS: Tabela que guarda as informações relativas aos objetos que serão bloqueados para compilação. O código de criação da tabela “VCS” pode ser visto no Apêndice B.

• VCS_VERSIONS: Tabela que guarda os códigos dos objetos que foram recompilados por outros usuários por estarem inválidos. O código de criação da tabela “VCS_VERSIONS” pode ser visto no Apêndice C.

• PAC_VCS: Package que contem as *procedures* de inserção e remoção das tabelas. O código fonte da package pode ser visto no Apêndice D.

• main.lua: Código em lua responsável por fazer a chamada ao *prompt* através do LuaJIT. Ao final ele recebe o retorno, faz as correções necessárias e envia as informações para a PAC_VCS. O código fonte do “main.lua” esta disponível no Apêndice E.

• vss_pl.lua: É acionado pelo *prompt* para fazer a chamada a biblioteca VSS. O código do “vss_pl.lua” pode ser visto no Apêndice F.

• vss.lua: Programa responsável por desenhar a tela, fazer a comunicação com o SourceSafe e retornar as informações para o *prompt*. O código do “vss.lua” pode ser visto no Apêndice G.

5.4 Problemas e Soluções

Dentre os diversos problemas encontrados no desenvolvimento desta solução, sem dúvida, o que mais demorou para ser resolvido foi o de acesso as bibliotecas externas pela versão de compilação da Lua do *plugin*, a LuaJIT. Apesar de relatos de que o LuaJIT consegue utilizar bibliotecas externas, não foi possível realizar as configurações necessárias para a sua utilização. Para solucionar este problema se fez necessária a instalação de outra versão da Lua. O desenvolvimento da tela na versão oficial da Lua resultou num módulo totalmente a parte, que não necessita do PL/SQL Developer para ser executado. Isso é bom pois essa solução

pode ser utilizada em diversos outros softwares, ou simplesmente como acesso ao SourceSafe sem necessitar abrir o VSS Explorer.

Em relação ao SGBD Oracle, o problema encontrado foi durante os testes com a estrutura criada. No momento que um objeto bloqueado para compilação ficar inválido e o usuário que bloqueou não estiver disponível, a estrutura mantinha o objeto inválido. Isto poderia atrapalhar o trabalho dos outros desenvolvedores, um objeto que seja muito referenciado e esteja inválido pode parar uma equipe inteira. Para solucionar esse problema foi feita uma alteração na *trigger "TCAD_VCS_DB".* Essa trabalha com um campo novo criado na tabela "VCS", o campo "COMP_QDO_INVAL". Toda vez que um objeto bloqueado estiver inválido e for feita uma requisição de compilação por outro desenvolvedor a *trigger* será ativada. Esta irá verificar se o valor do campo novo é "1", se for ela libera a compilação e salva o código atual do objeto na tabela "VCS_VERSIONS". Inicialmente foi feita uma tentativa de que a própria estrutura recompilasse o objeto inválido, porém o Oracle entende isso como uma recursividade e não permite que o código seja compilado. A recursividade é encontrada no momento que a *trigger* que verifica compilação solicita outra compilação. Por esse motivo esta tentativa de solução não foi mantida.

6 ESTUDO DE CASO

Todos os testes foram realizados pela equipe de desenvolvimento do sistema de faturamento da RGE, como havia sido mencionado na sessão 4.2.6.

Desde o início do desenvolvimento da estrutura de banco de dados até o momento da escrita deste, passaram-se aproximadamente seis meses. A estrutura foi ativada pouco mais de 1500 vezes entre bloqueios e recompilações.

A seguir, os cenários de teste propostos na sessão 4.2.5 serão apresentados

6.1 Cenários de Teste

Para execução dos testes foi utilizado o objeto “PAC_ANOM_PUB”, uma *package* do banco de dados da RGE. O usuário principal é o “3003886” e o secundário é o “2000110”.

Os testes que serão apresentados são para apenas um objeto. A tela não permite que mais de um objeto seja selecionado. Para que mais de um objeto seja afetado, as ações dentro da tela devem ser repetidas.

6.1.1 Fazer check out de um ou mais objetos através do PL/SQL Developer.

O teste de *check out* foi executado no objeto “PAC_ANOM_PUB”, conforme ilustra a Figura 54, este ainda não esta sendo controlado pela estrutura. A Figura 55 mostra que o objeto está liberado para *check out* no SourceSafe, pois a coluna “User” está em branco.

```
SQL> select count(*) from vcs where objeto like 'PAC_ANOM_PUB';
  COUNT(*)
----------
         0
```

Figura 54 - Objeto sem bloqueio na estrutura.
Fonte: Autoria própria.

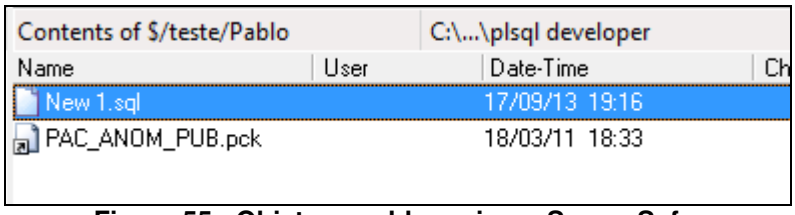

Figura 55 - Objeto sem bloqueio no SourceSafe.
Fonte: Autoria própria.

Após ser clicado no menu Lua do PL/SQL Developer, deve ser selecionada a opção “VSS”, como mostra a Figura 47. A tela da Figura 52 irá abrir e deve ser feita a navegação até o diretório listado na Figura 55, o resultado é mostrado na Figura 56.

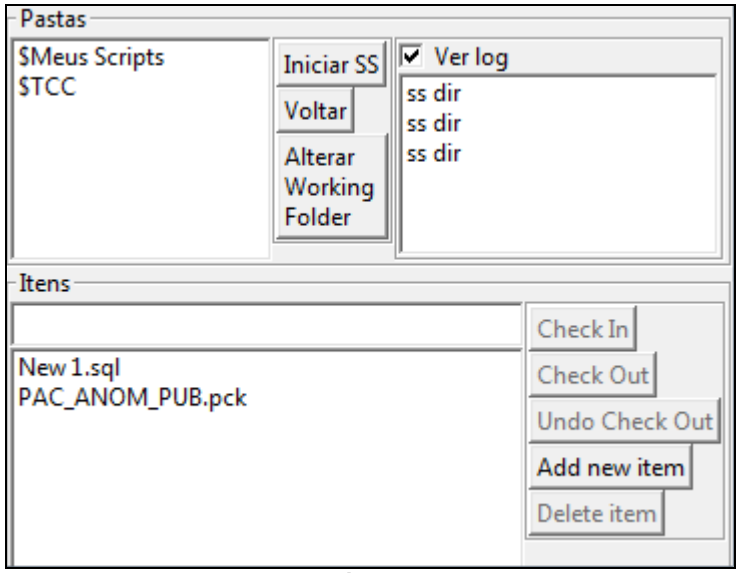

Figura 56 - Navegação nos diretórios do VSS Explorer feita pela tela.
Fonte: Autoria própria.

Ao selecionar o item “PAC_ANOM_PUB” o botão de “Check out” será desbloqueado e ao clicar neste, o Log irá registrar a alteração e uma mensagem de sucesso será exibida (Figura 57). Neste momento o SourceSafe Explorer já estará exibindo o *status* do arquivo como em *check out*, a Figura 58 mostra que o usuário “3003886” fez o bloqueio.

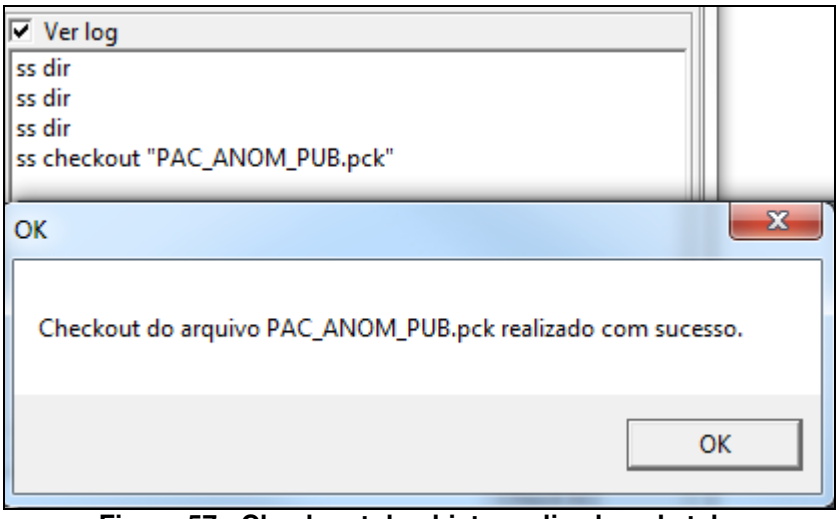

Figura 57 - Check out de objeto realizado pela tela.
Fonte: Autoria própria.

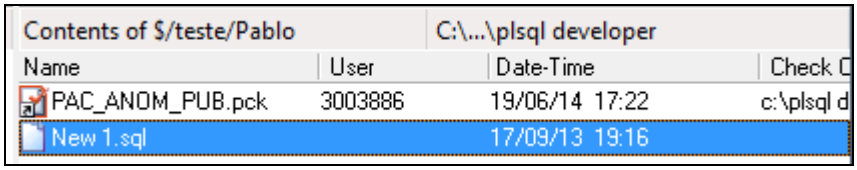

Figura 58 - Objeto bloqueado no SourceSafe pelo User 3003886.
Fonte: Autoria própria.

Apesar do VSS Explorer já estar marcando o bloqueio do objeto, o processo de inserção na tabela irá ser realizado apenas quando a tela for fechada e o "main.lua" receber o retorno da tela. A Figura 59 mostra que ao retornar para o PL/SQL Developer, o objeto já esta na Tabela de controle.

```
SQL> select count(*) from vcs where objeto like 'PAC_ANOM_PUB';
  COUNT(*)
----------
         1
```

Figura 59 - Objeto com bloqueio na estrutura.
Fonte: Autoria própria.

6.1.2 Fazer o check in de um ou mais objetos através do PL/SQL Developer.

O processo de *check in* segue a mesma sequência para abrir a tela que o processo 6.1.1. Após ser feita a navegação até o diretório do arquivo pela tela e o arquivo for selecionado o botão de "Check in" será desbloqueado. Ao clicar neste,

uma tela para inserir o comentário do que foi feito se abrirá (Figura 60).

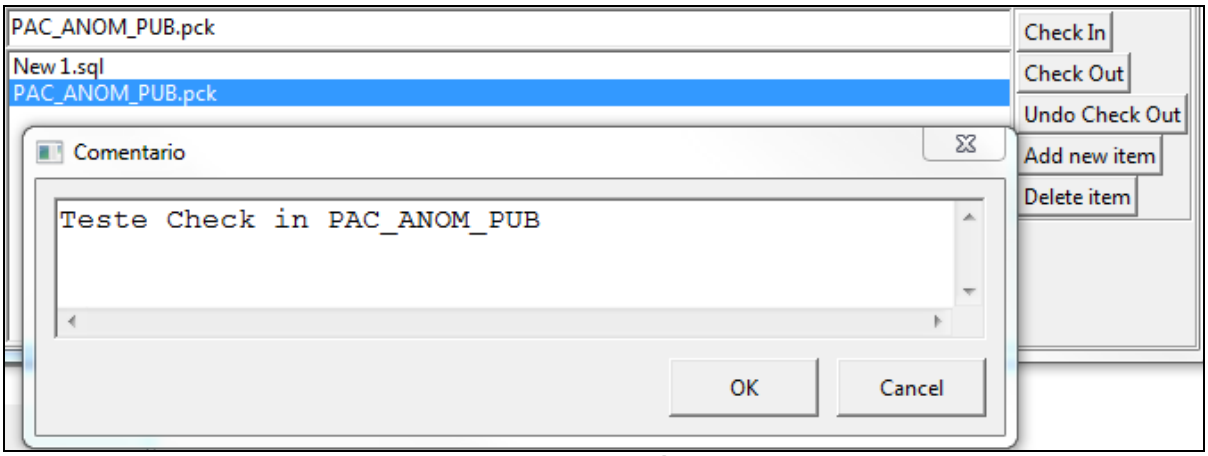

Figura 60 - Tela de comentário para check in.
Fonte: Autoria própria.

Após clicar no botão “OK” o Log irá registrar o *check in* e o VSS Explorer irá desbloquear o objeto conforme ilustra a Figura 61. Assim como acontece no processo de *check out* a tabela “VCS” só irá ter o objeto removido quando a tela for fechada.

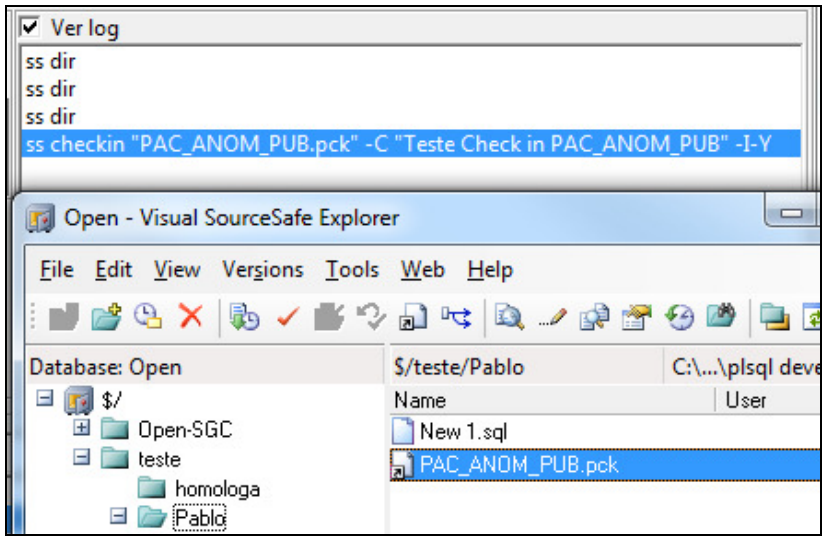

Figura 61 - Check in realizado pela tela com atualização no SourceSafe.
Fonte: Autoria própria.

6.1.3 Fazer o undo check out de um ou mais objetos através do PL/SQL Developer.

O processo de *undo check out* segue a mesma sequência do processo 6.1.1.

Utilizando a tela, o *check out* do arquivo “PAC_ANOM_PUB” foi refeito e em seguida foi feito o *undo check out*. Ao clicar no botão “Undo Check Out” o processo verifica se o arquivo já esta em *check out*, se estiver verifica se o usuário que bloqueou é o mesmo que está tentando realizar o *undo check out*. Caso esteja tudo certo, o *undo check out* é feito na tela, registrado em Log e apresenta uma mensagem de sucesso. A Figura 62 mostra os processos de check out e undo check out feitos em sequência, assim como a atualização no VSS Explorer e a mensagem de sucesso.

Assim como acontece nos outros processos a tabela “VCS” só irá ser atualizada quando a tela for fechada. Neste caso, como foi feito um *check out* e depois um *undo check out* a tabela terá um objeto inserido e logo em sequência o mesmo será removido.

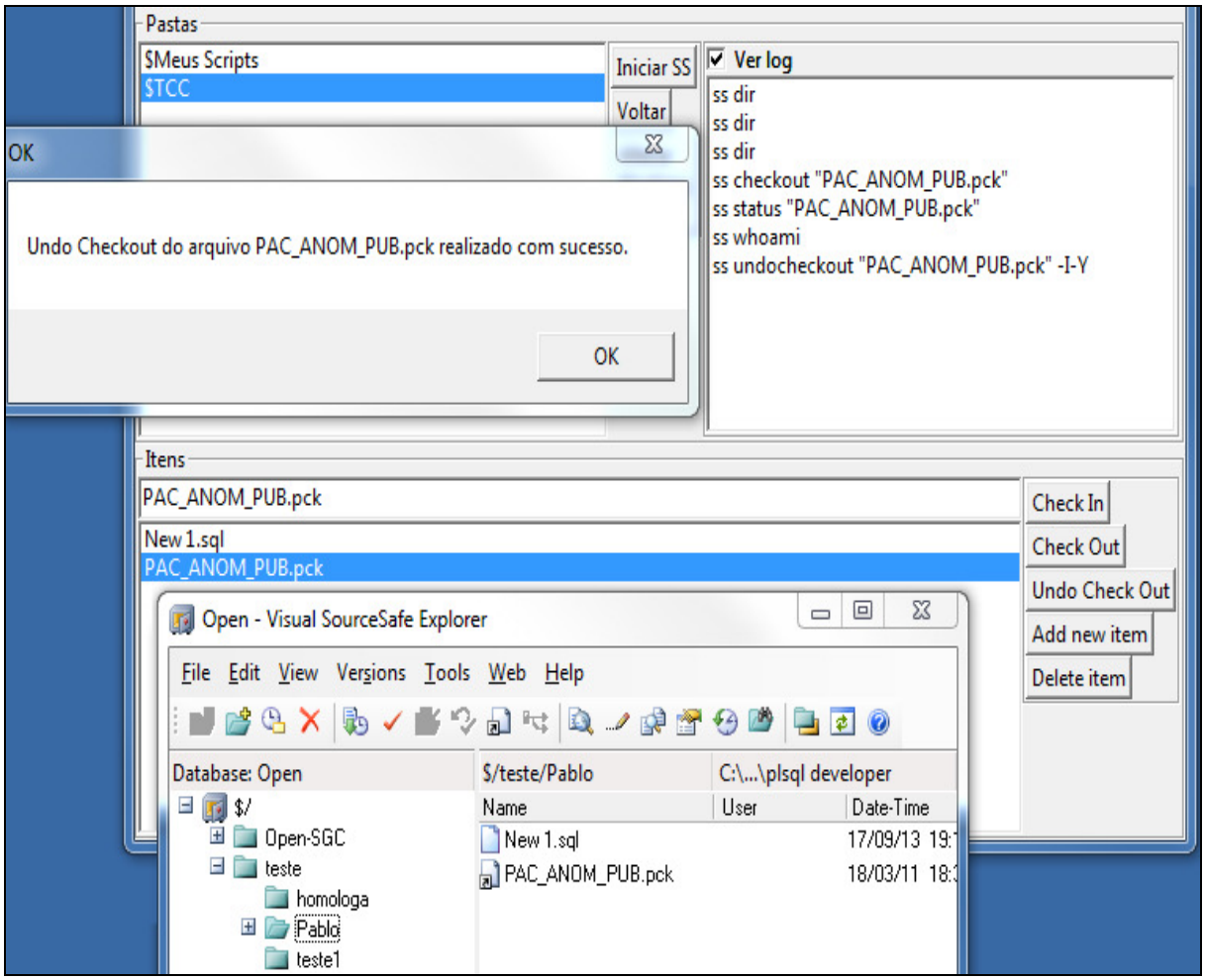

Figura 62 - Exemplo de check out e undo check out na tela.
Fonte: Autoria própria.

6.1.4 Abrir a edição de um objeto com mais de um usuário.

A Figura 63 mostra a “PAC_ANOM_PUB” sendo aberta em modo de edição por dois usuários, em máquinas diferentes, ao mesmo tempo.

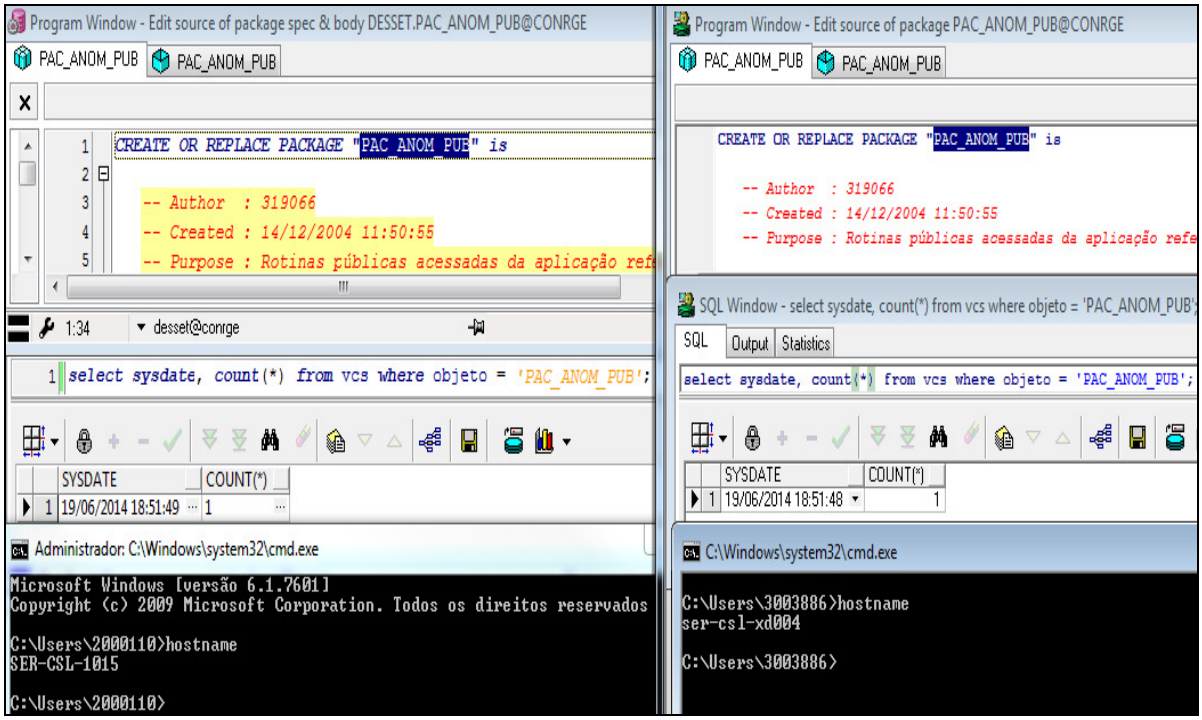

Figura 63 - Mesmo objeto em modo de edição em duas maquinas distintas.
Fonte: Autoria própria.

6.1.5 Fazer o check out e compilar um objeto

Para realizar esse teste, foi utilizada a package “PAC_ANOM_PUB”. A Figura 64 mostra que a package já está bloqueada e que a estrutura permitiu a compilação.

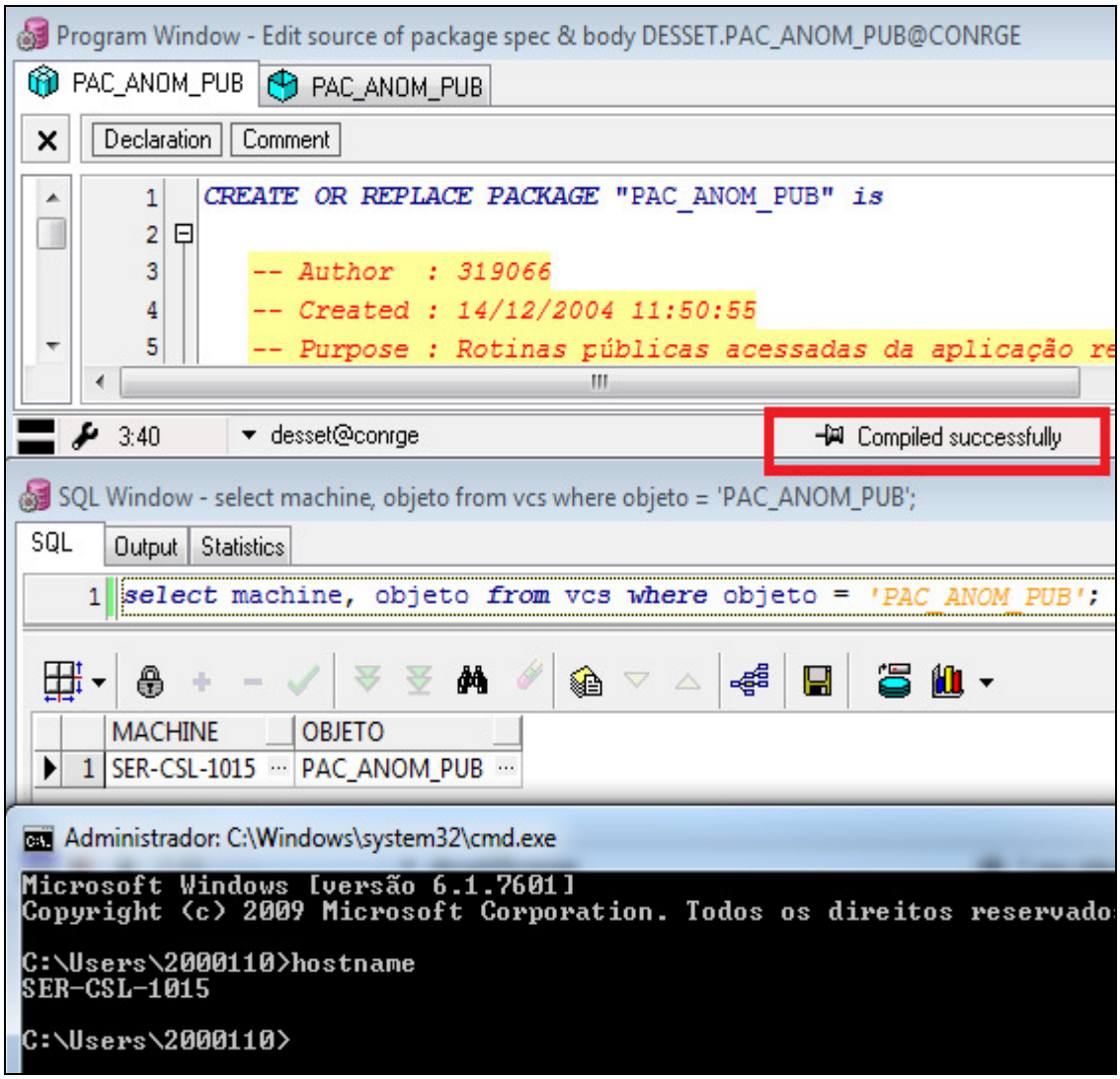

Figura 64 - Compilação realizada com sucesso.
Fonte: Autoria própria.

6.1.6 Fazer o check out de um objeto já bloqueado.

A Figura 65 mostra que ao tentar realizar o *check out* para um objeto que já esteja bloqueado para outro usuário, a tela exibe uma mensagem de erro.

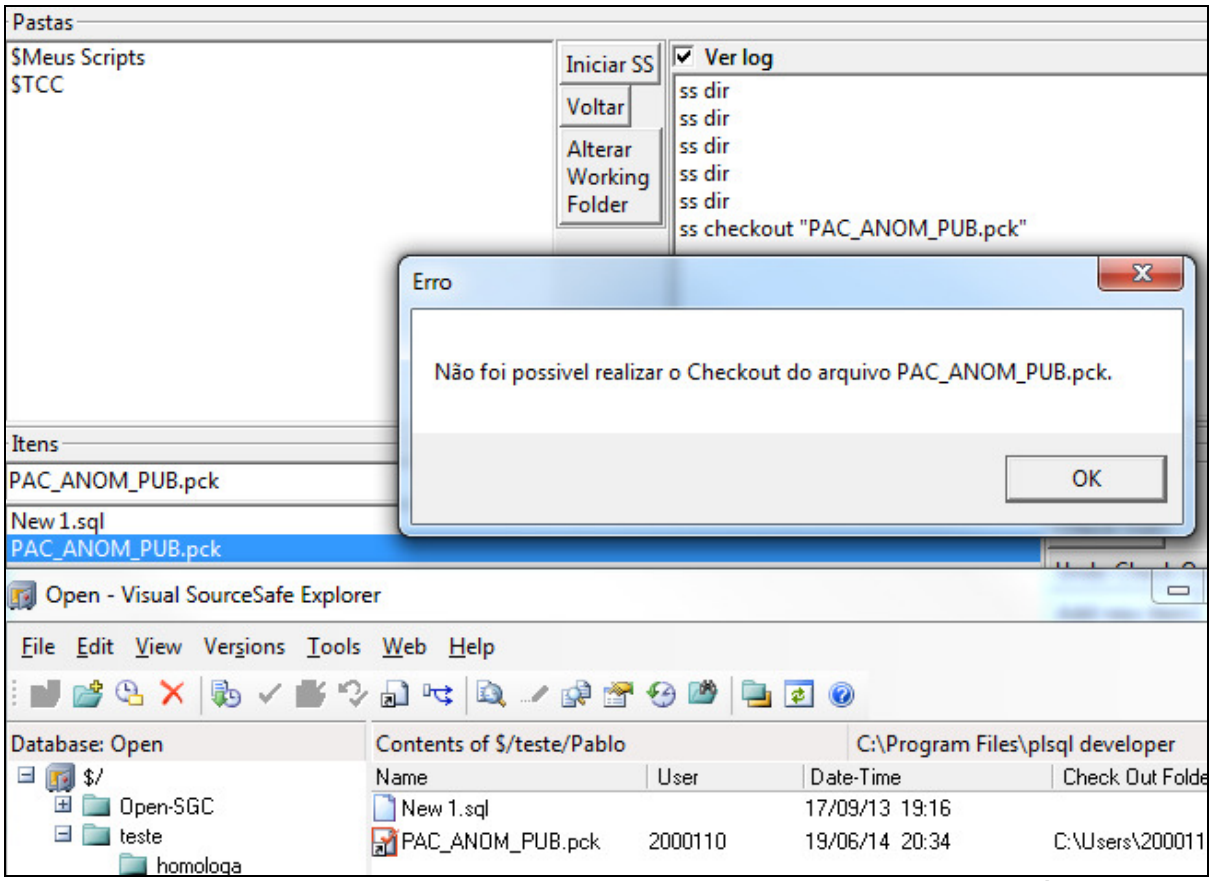

Figura 65 - Tentativa de check out para arquivo bloqueado por outro usuário.
Fonte: Autoria própria.

6.1.7 Fazer o check in de um objeto em check out para outro usuário.

Neste caso foi realizado o *check out* do arquivo “PAC_ANOM_PUB” por outro usuário. Ao tentar realizar o *check in* a tela faz a validação do usuário que realizou o *check out* e não permite a operação, como mostra a Figura 66.

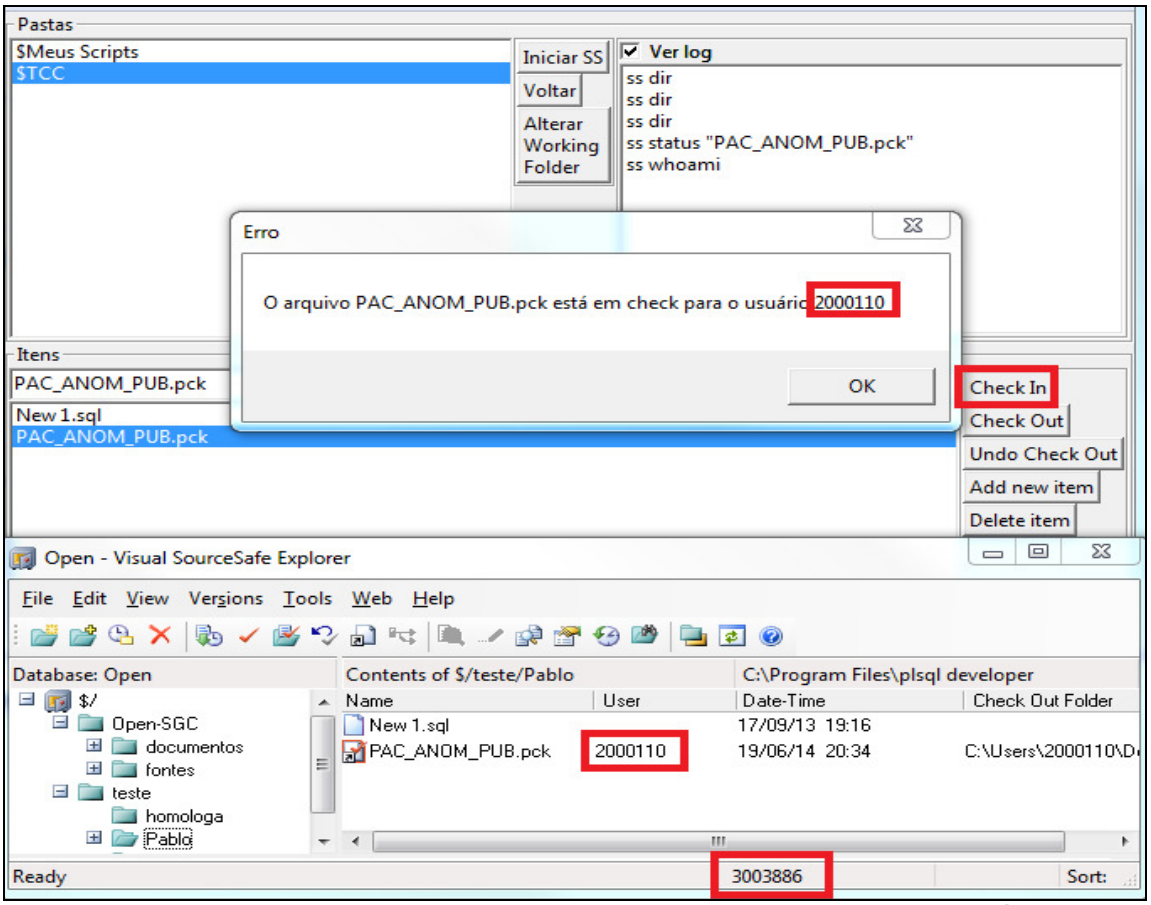

Figura 66 - Tentativa de check in para arquivo bloqueado por outro usuário.
Fonte: Autoria própria.

6.1.8 Fazer o undo check out de um objeto em check out para outro usuário

Ao tentar realizar *um undo check out* o sistema verifica se o usuário que bloqueou é o mesmo que está solicitando o desbloqueio. Caso o usuário seja outro a tela irá mostrar uma mensagem de erro conforme ilustra a Figura 67. A verificação realizada é a mesma mostrada no caso de *check in* para outro usuário.

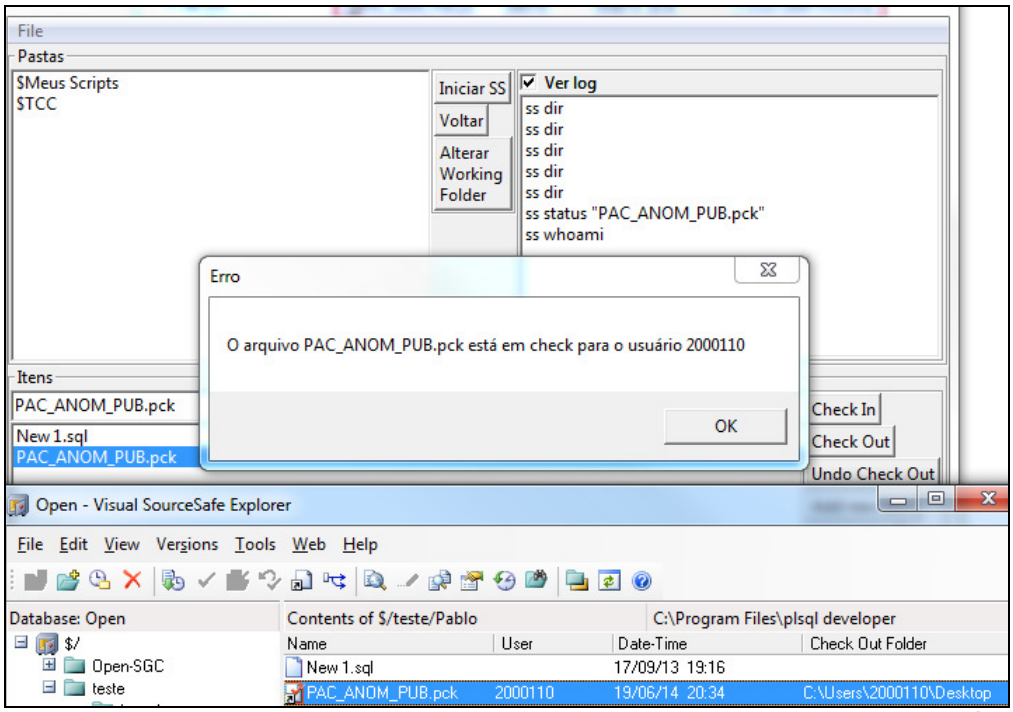

Figura 67 - Tentativa de undo check out de arquivo bloqueado para outro usuário.
Fonte: Autoria própria.

6.1.9 Compilar um objeto sem ter feito check out

Na sessão 4.2.5 foi escrito que deveria dar erro, porém durante a o desenvolvimento, optou-se por não bloquear a compilação de objetos desbloqueados.

6.1.10 Compilar um objeto que esteja em check out para outro usuário

A Figura 68 mostra que a package já está bloqueada, neste caso para outra máquina e a estrutura não permitiu a compilação.

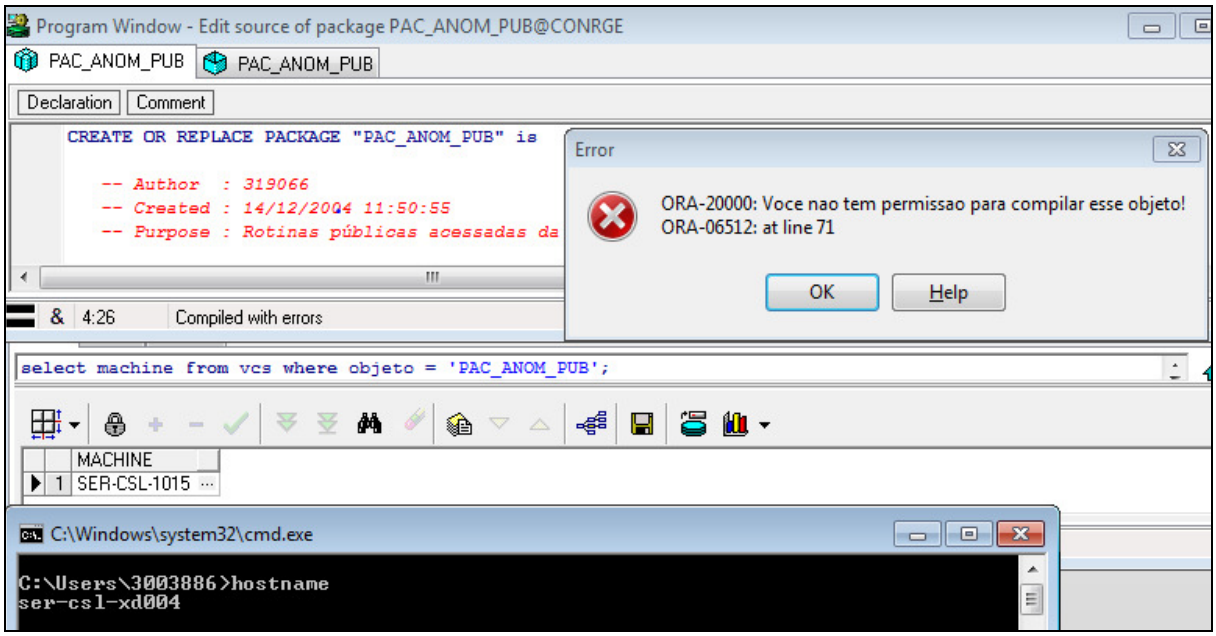

Figura 68 - Bloqueio de compilação realizado pela trigger.
Fonte: Autoria própria.

7 CONCLUSÃO

Neste trabalho foram estudados alguns objetos do SGBD da Oracle, como esses são compilados e onde ficam armazenados. Além disso, foi apresentada a importância de sistemas de controle de versão, como ênfase no antigo VCS da Microsoft o SourceSafe. Foi exposto o motivo pelo qual os VCSs não conseguem atuar nos objetos de BD, e os problemas que isso pode causar quando se tem uma equipe de desenvolvimento.

Visto que a integração de VCSs e objetos de BDs ainda é um problema sem solução, foi proposto e desenvolvido um método que garante a exclusividade de compilação para uma máquina. Para que esse método funcione é necessário, o cadastro prévio em uma estrutura. Esta estrutura deve ser alimentada por uma tela que possui integração com o VCS escolhido, nesse caso o SourceSafe.

O controle de compilação de objetos de BD em ambiente de desenvolvimento é de extrema importância, quando se fala em equipes de desenvolvimento. Este controle evita a perda, ou sobrescrita, de código.

A tela e a estrutura foram testadas e provaram estar funcionando, sendo assim, considera-se que os objetivos apresentados na sessão 1.1.1 foram amplamente atingidos. De qualquer forma ainda existem diversas melhorias que podem ser feitas. Algumas dessas melhorias serão apresentadas na sessão 7.1.

7.1 Trabalhos Futuros

Dentre as possíveis melhorias futuras, a que provavelmente traria maior ganho a este projeto seria a configuração correta do LuaJIT. Se o LuaJIT pudesse utilizar as bibliotecas externas da Lua, não haveria a necessidade do acesso ao *prompt* e da instalação da versão da Lua para Windows. Além disso, seria possível uma integração maior com a estrutura do BD, assim a tela não precisaria ser fechada para que o BD recebesse as alterações.

Outra alteração em Lua que poderia melhorar a utilização da tela seria, transformar a navegação de listas para árvore. Hoje a navegação por diretórios e arquivos se baseia em duas listas. Ainda, poderia ser feita a alteração de seleção simples na lista, ou árvore, para seleção múltipla, por exemplo, fazer *check out* de

diversos arquivos de uma pasta. A utilização de comandos em pastas é outra opção que não foi considerada neste trabalho.

Sem sair das alterações em Lua, uma funcionalidade que sincronizasse o BD com o VSS aumentaria a segurança. Para exemplificar, no caso da RGE, nem todos os desenvolvedores estão utilizando a estrutura nova. Isso causa uma enorme diferença entre o que esta no SourceSafe e que está no BD. No VSS Explorer é possível encontrar 57 objetos de BD versionados, já na tabela “VCS” apenas 12 estão cadastrados.

Na parte de melhorias para o SourceSafe, ficaram faltando as funcionalidades 1 e 2 propostas na sessão 4.2.3, Abrir e Fechar projeto. Atualmente a tela trabalha com o projeto aberto no VSS Explorer. Para alterar o projeto na tela é necessário fazer o mesmo no VSS Explorer.

Partindo para melhorias propostas na estrutura do BD, a primeira seria criar uma tabela nova permitindo que cada desenvolvedor fizesse um cadastro de máquinas utilizadas e email para receber alertas. Atualmente um objeto só pode estar relacionado a uma máquina.

Atualmente a estrutura dentro do BD deve ser criada manualmente, uma alteração na tela que permitisse a criação automática dessa estrutura, provavelmente iria facilitar a sua adesão.

Outra melhoria relacionada ao BD seria conseguir uma forma de recompilar objetos inválidos sem alteração do código. Isso iria inutilizar a tabela “VCS_VERSIONS”, que com o tempo pode passar a utilizar muito espaço.

REFERÊNCIAS

ABREU, M.; MACHADO, F. **Projeto de Banco de Dados: Uma Visão Prática**. 11. ed. São Paulo: Érica, 2004. 303p.

BEAULIEU, A.; MISHRA, S. **Mastering Oracle SQL**. Beijing: O'Reilly & Associates, 2002. 336p.

BOLINGER, D.; BRONSON, T. **Applying RCS and SCCS - From Source Control to Project Control**. 1. ed. : O'Reilly & Associates , 1995. 528p.

CHACON, S. **Pro Git**. 1. ed. : Apress, 2009. 288p.

COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.; PILATOCC, C. M. **Version Control with Subversion: For Subversion 1.7**. : O'Reilly & Associates, 2011. 468p. Disponível em: <http://svnbook.red-bean.com/en/1.7/svn-book.pdf> Acesso em: 12 nov. 2013

DATE, C. J. **An Introduction to Database Systems**. 8. ed. : Addison-Wesley, 2004. 1024p.

DAWES, C.; FEUERSTEIN, S.; PRIBYL, B. **Oracle PL/SQL Language Pocket Reference**. 4. ed. Beijing: O'Reilly & Associates, 2008. 180p.

FERREIRA, J. E.; ITALIANO, I. C.; TAKAI, O. K. **Introdução a Banco de Dados**. : , 2005. 124p.

FEUERSTEIN, S. **Oracle PL/SQL Programming**. 5. ed. Beijing: O'Reilly & Associates, 2009. 1299p.

FREITAS, R. C.; RAMOS, E. S. **Análise Comparativa de Sistemas de Controle de Versões Baseados em Código Aberto**. InfoBrasil, Fortaleza, n.1, p. , 2010.

GLASSER, A. L. **The evolution of a source code control system**. Proceedings of thesoftware quality assurance workshop on Functional and performance issues, New York, n.1, p.122-125, 1978.

GREENBERG, N.; NATHAN, P. **SQL1 - Student Guide**. : Oracle, 2001. 483p.

HARDMAN, R.; MCLAUGHLIN, M.; URMAN, S. **Oracle Database 10g PL/SQL Programming**. New York: Oracle Press, 2004. 895p.

HARPER, S. **ORACLE SQL Developer 2.1**. BIRMINGHAM - MUMBAI: Packt Publishing, 2009. 496p.

HEUSER, C. A. **Projeto de Banco de Dados**. 6. ed. Porto Alegre: Bookman, 2010. 282p.

IERUSALIMSCHY, R., FIGUEIREDO, L., CELES, W. ***Lua 5.1 Reference Manual*** (2006).

JONES, A. D.; PLEW, R.; STEPHENS, R. **Sams Teach Yourself SQL in 24 hours**. 5. ed. Indianapolis: Pearson, 2011. 497p.

JUNIOR, A. O. **APLICAÇÃO DE GERÊNCIA DE OBJETOS COMPARTILHÁVEIS DE UM SGBD ORACLE**. 206. 60f. Monografia (Bacharelado em CIÊNCIAS DA COMPUTAÇÃO) - UNIVERSIDADE REGIONAL DE BLUMENAU, BLUMENAU, 206.

KORTH, H. F.; SILBERSCHATZ, A.; SUDARSHAN, S. **Sistema de Banco de Dados**. 5. ed. : Campus, 2006. 804p.

LEITE, M. M. R. **UM ESTUDO COMPARATIVO DOS SISTEMAS DE BANCO DE DADOS ORACLE E POSTGRESQL COM ÊNFASE NAS EXTENSÕES ESPACIAIS**. 2011. 68f. Monografia (Especialização em GEOPROCESSAMENTO) - UNIVERSIDADE ESTADUAL DA PARAÍBA, CAMPINA GRANDE, 2011.

MOORE, S. **Oracle Database PL/SQL Language Reference 11g Release 1 (11.1)**. : Oracle, 2009. 712p.

NATHAN, P.; PATABALLA, N. **Oracle9i: Program with PL/SQL**. : Oracle, 2001. 336p.

NATHAN, P.; PATABALLA, N. **Introduction to Oracle 9i PL/SQL**. : ORACLE, 2001. 330p.

PATABALLA, N. **Oracle9i: Develop PL/SQL Program Units**. : Oracle, 2001. 496p.

POWELL, G. **Beginning Database Design**. : Wrox, 2006. 496p.

Banco de Dados Capítulo 1: Introdução. Disponível em: <http://www.dsc.ufcg.edu.br/~baptista/cursos/BDadosI/Capitulo1.pdf> Acesso em: 19 mai. 2013

COMPANY NEWS; MICROSOFT SAYS IT HAS ACQUIRED ONE TREE SOFTWARE. Disponível em: <http://www.nytimes.com/1994/11/16/business/company-news-microsoft-says-it-has-acquired-one-tree-software.html> Acesso em: 2 jul. 2013

Conceitos Básicos de Controle de Versão de Software - Centralizado e Distribuído.. André Felipe Dias Disponível em: <http://www.pronus.eng.br/artigos_tutoriais/gerencia_configuracao/conceitos_basicos_controle_versao_centralizado_e_distribuido.php?pagNum=0> Acesso em: 9 abr. 2013

Enciclopédia. Disponível em: <http://www.pcmag.com/encyclopedia/term/44707/ide> Acesso em: 3 jul. 2013

Introdução à TRIGGERS.. Wagner Bianchi Disponível em: <http://www.devmedia.com.br/introducao-a-triggers/1695> Acesso em: 16 jun. 2013

Sistemas de Controle de Versão.. Eduardo Cavalcante Lacerda Disponível em: <http://www.devmedia.com.br/sistemas-de-controle-de-versao/24574> Acesso em: 9 abr. 2013

Suporte Microsoft. Disponível em: <http://support.microsoft.com/lifecycle/search/default.aspx?sort=PN&alpha=sourcesafe&Filter=FilterNO> Acesso em: 2 jul. 2013

APÊNDICE A - CÓDIGO FONTE DO ARQUIVO TCAD_VCS_DB

```
create or replace trigger "TCAD_VCS_DB"
   before create or alter or drop on database

declare

   v_mac     varchar2(30);
   machine   varchar2(30);
   v_user    varchar2(30);
   aux       number;
   ln_count  number;
   ln_comp   vcs.comp_qdo_inval%type;

   sql_text dbms_standard.ora_name_list_t;
   n         pls_integer;
   v_stmt    clob;

begin
   v_stmt := ' ';
   select count(*) into aux from vcs where upper(vcs.objeto) =
upper(ora_dict_obj_name);

   if aux > 0 then

      select upper(sys_context('USERENV', 'TERMINAL')),
             upper(sys_context('USERENV', 'OS_USER'))
        into v_mac, v_user
        from dual;

      select upper(machine), comp_qdo_inval
        into machine, ln_comp
        from vcs
       where vcs.objeto = ora_dict_obj_name;

      if trim(v_mac) not like '%' || machine || '%' then

         select count(*)
           into ln_count
           from all_objects
          where status <> 'VALID'
            and object_name = upper(ora_dict_obj_name);

         if ln_count <> 0 and ln_comp = 1 then

            n := ora_sql_txt(sql_text);
            for i in 1 .. n loop
               v_stmt := v_stmt || sql_text(i);
            end loop;

            pac_vcs.p_grava_text(ora_dict_obj_name, v_stmt, v_mac);

         else
            raise_application_error(-20000, 'Voce nao tem permissao para
compilar esse objeto!');
         end if;
      end if;
   end if;
end;
```

APÊNDICE B - CÓDIGO FONTE DO ARQUIVO VCS

```
-- Create table
create table OWNER.VCS
(
  nome           VARCHAR2(50) not null,
  re             VARCHAR2(10) not null,
  machine        VARCHAR2(50) not null,
  email          VARCHAR2(50),
  objeto         VARCHAR2(50) not null,
  dt_ini         DATE not null,
  dt_fim         DATE,
  comp_qdo_inval NUMBER default 0 not null
);
-- Add comments to the table
comment on table OWNER.VCS
  is 'Tabela de VCS';
-- Add comments to the columns
comment on column OWNER.VCS.nome
  is 'Nome do usr';
comment on column OWNER.VCS.re
  is 'RE/ID/Matricula do usr';
comment on column OWNER.VCS.machine
  is 'Maquina utilizada';
comment on column OWNER.VCS.email
  is 'email para receber notificações';
comment on column OWNER.VCS.objeto
  is 'NOME DO OBJETO QUE ESTA SENDO VERSIONADO';
comment on column OWNER.VCS.dt_ini
  is 'Data inicio uso do objeto';
comment on column OWNER.VCS.dt_fim
  is 'Data fim uso do objeto';
comment on column OWNER.VCS.comp_qdo_inval
  is 'Permite compilacao alheia quando esta invalido ( 1 - S / 0 - N ) ';
```

APÊNDICE C - CÓDIGO FONTE DO ARQUIVO VCS_VERSIONS

```
-- Create table
create table OWNER.VCS_VERSIONS
(
  objeto   VARCHAR2(50) not null,
  machine  VARCHAR2(50) not null,
  obj_code CLOB not null,
  dt_comp  DATE not null
);
-- Add comments to the table
comment on table OWNER.VCS_VERSIONS
  is 'Tabela que guarda versoes dos objetos invalidos recompilados por
outros usrs';
-- Add comments to the columns
comment on column OWNER.VCS_VERSIONS.objeto
  is 'NOME DO OBJETO QUE ESTA SENDO RECOMPILADO';
comment on column OWNER.VCS_VERSIONS.machine
  is 'Maquina utilizada para compilar objeto invalido';
comment on column OWNER.VCS_VERSIONS.obj_code
  is 'CÓDIGO DO OBJETO QUE ESTA SENDO RECOMPILADO';
comment on column OWNER.VCS_VERSIONS.dt_comp
  is 'Data de Compilacao do objeto';
```

APÊNDICE D - CÓDIGO FONTE DO ARQUIVO PAC_VCS

```
create or replace package body pac_vcs is

   --------------------------------------------------------------------------
   -- p_insere_vcs
   --------------------------------------------------------------------------
   -- Author  : 3003886
   -- Created : 09/12/2013 11:23:24
   --------------------------------------------------------------------------

   procedure p_insere_vcs(piv_objeto  vcs.objeto%type,
                          piv_re      vcs.re%type,
                          piv_machine vcs.machine%type) is
   begin
      insert into vcs
         (objeto, nome, re, machine, dt_ini)
      values
         (piv_objeto, piv_re, piv_re, piv_machine, sysdate);
      commit;
   end p_insere_vcs;

   procedure p_deleta_vcs(piv_objeto vcs.objeto%type) is
   begin
      delete vcs where objeto = piv_objeto;
      commit;
   end p_deleta_vcs;

   procedure p_grava_text(piv_objeto vcs.objeto%type, pclob clob,
piv_machine vcs.machine%type) is
   begin
      insert into vcs_versions values (piv_objeto, piv_machine, pclob,
sysdate);
   end p_grava_text;
end pac_vcs;
```

APÊNDICE E - CÓDIGO FONTE DO ARQUIVO MAIN.LUA

```
local AddMenu = ...

local plsql = plsql
local SYS, IDE, SQL = plsql.sys, plsql.ide, plsql.sql

local ShowMessage = plsql.ShowMessage

local Gsub, Sub, Upper, Lower = string.gsub, string.sub, string.upper,
string.lower

do
      local function VSS()

            --Chama o arquivo LUA a ser executado e joga o retorno em um
arquivo
            a = io.popen('"c://plsql
developer/PlugIns/lua/VSS/vss_pl.lua"')
            -- Lê o retorno do LUA no arquivo
            b = (a:read("*a"))
            --Fecha o arquivo
            a:close()

            reg_ss = 0
            for linhas in string.gmatch(b, ".-\n") do --split \n
                  --Pega somente registros de alteracao SS
                  if reg_ss == 1 then
                        --pega da terceira letra ate o fim/tira o "ss "
                        for linha in string.gmatch(linhas:sub(3), ".-\n")
do
                              --pega a primeira palavra
                              aux = 0
                              lin = ' '
                              for palavras in string.gmatch(linha, "%w+")
do
                                    if palavras:gsub("^%s*(.-)%s*$", "%1")
== 'add' or
                                       palavras:gsub("^%s*(.-)%s*$", "%1")
== 'checkout' or
                                       palavras:gsub("^%s*(.-)%s*$", "%1")
== 'checkin' or
                                       palavras:gsub("^%s*(.-)%s*$", "%1")
== 'undocheckout' then

                                          command = palavras:gsub("^%s*(.-
)%s*$", "%1")
                                          objeto = linha:sub(command:len()
+2)
                                          objeto =
objeto:sub(1,objeto:find('%.')-1)
                                          insere_tabela(command, objeto)
                                    end
                              end
                        end
                  end
                  if linhas:gsub("^%s*(.-)%s*$", "%1") == 'INICIO' then --
Testa depois para ignorar Linha de INICIO
                        reg_ss =1
                  end
```

```
            end
      end

      AddMenu(VSS, "&Lua / VSS")

end

      function insere_tabela (c, o)
            re = prompt('echo %username%')
            re = re:sub(1,re:len()-1)
            maq = prompt('hostname')
            maq = maq:sub(1,maq:len()-1)
            obj = o:sub(3,o:len())
            p = string.char(39)

            if c == 'checkout' then
                  ins = 'begin \n pac_vcs.p_insere_vcs( '.. p.. obj.. p..
', ' ..

                   p .. re .. p .. ', ' ..

                   p .. maq .. p ..

                   ');\n end;'
                  ins = SQL.Execute(ins)

                  elseif c == 'undocheckout' or c == 'checkin' then
                        command = 'begin \n pac_vcs.p_deleta_vcs( '.. p ..
obj .. p.. ');\n end;'
                  ins = SQL.Execute(command)
            end

      end--insere_tabela

      function prompt(c)--executa um comando no prompt e retorna o valor
pra "b"
            a = io.popen('cmd.exe /c call ' .. c)
            b = (a:read("*a"))
            a:close()
            return b
      end --FIM prompt

return {
      OnActivate,
      OnDeactivate,
      CanClose,
      AfterStart,
      AfterReload,
      OnBrowserChange,
      OnWindowChange,
      OnWindowCreate,
      OnWindowCreated,
      OnWindowClose,
      BeforeExecuteWindow,
      AfterExecuteWindow,
      OnConnectionChange,
      OnPopup,
      OnMainMenu,
      OnTemplate,
      OnFileLoaded,
```

```
    OnFileSaved,
    About,
    CommandLine,
    RegisterExport,
    ExportInit,
    ExportFinished,
    ExportPrepare,
    ExportData
                                    }
```

APÊNDICE F - CÓDIGO FONTE DO ARQUIVO VSS_PL.LUA

```
vss = require "vss"
vss.ini_ss()
```

APÊNDICE G - CÓDIGO FONTE DO ARQUIVO VSS.LUA

```
local vss = {}
local msg_no_check = "No checked out files found."
local problemas   ='1. Todas as perguntas são respondidas com SIM(Y)'..
' para evitar travar.\n'..
'2. Algumas vezes os arquivos "travam" para checkout, necessário '..
'realizar pelo VSS Explorer.\n'..
'3. Funcionalidade de comentario somente para o CheckIn.\n'..
'4. Checkin nao valida se esta em checkout. \n'..
'5. Para alterar o ambiente de trabalho a pasta selecionada deve '..
'conter algum arquivo\n'..
'-Fica com a tela do prompt aberta atraz da aplicação.'

require "iuplua"

function vss.ini_ss()
      retorno = "INICIO"
      ------------------------------------------------------------------------
--------------------
      --Cria a interface
      ------------------------------------------------------------------------
--------------------

      --Cria os botoes
      btn_on_off = iup.button{ title = "Iniciar SS" }
      btn_voltar = iup.button{ title = "Voltar" }
      btn_checkin = iup.button{ title = "Check In" }
      btn_checkout = iup.button{ title = "Check Out" }
      btn_undo_checkout = iup.button{ title = "Undo Check Out" }
      btn_add_new = iup.button{ title = "Add new item" }
      btn_del_item = iup.button{ title = "Delete item"}
      btn_alter_workfold = iup.button{ title = "Alterar \nWorking
\nFolder"}

      --Cria checkbox
      tg_log = iup.toggle{title ="Ver log", value = "YES"}

      --Cria os itens dos Menus
       item_probls = iup.item{title = "Problemas conhecidos"}

      --Cria os Menus
      menu_file = iup.menu {item_probls}
      submenu_file = iup.submenu {menu_file; title = "File"}
      menu = iup.menu {submenu_file}

      --Cria as listas
      list_folder = iup.list { multiple="NO", size="200x100",
                                          expand = 'YES'}
      list_itens = iup.list { multiple="NO", size="200x100",
                                        editbox ='YES', expand = 'YES'}
      list_log = iup.list { multiple="NO", size="200x100",
                                      expand = 'YES'}

      --Cria os frames/box
      frm_bts1 = iup.frame {iup.vbox {btn_on_off,btn_voltar,
                                                    btn_alter_workfold}}
      frm_bts2 = iup.frame {iup.vbox {btn_checkin,btn_checkout,

      btn_undo_checkout,btn_add_new,
```

```
                                                              btn_del_item}}
      frm_log = iup.frame {iup.vbox {tg_log,list_log}}

      frm_top = iup.frame {iup.hbox {list_folder,frm_bts1,frm_log};
                                                          title = "Pastas"}
    frm_bot = iup.frame {iup.hbox {list_itens,frm_bts2};
                                                              title = "Itens"}

      --Cria a tela
      dlg = iup.dialog {iup.vbox{frm_top,frm_bot};
                                title="VSS - Pablo", menu=menu}

      --Bloqueia os botoes
      btn_checkin.active = "NO"
      btn_checkout.active = "NO"
      btn_undo_checkout.active = "NO"
      btn_add_new.active = "NO"
      btn_alter_workfold.active = "NO"
      btn_del_item.active = "NO"
      btn_voltar.active = "NO"
      --Inicia a tela principal centralizada
      dlg:showxy(iup.CENTER, iup.CENTER)


      ------------------------------------------------------------------------
--------------------
      --Custom functions
      ------------------------------------------------------------------------
--------------------
      function open_dir(dir)
            --Abre o diretorio
            os.execute('cmd.exe /c call ss cd '.. dir)
            --Lista arquivos e pastas
            return prompt('ss dir')
      end --FIM open_dir

      function refresh_list (new_list)
            --Zera as variaveis auxiliares
            l1 = 0
            fim = 0

            --Limpa as duas listas
            list_folder.removeitem = ALL
            list_itens.removeitem = ALL

            --Bloqueia os botoes
            btn_checkin.active = "NO"
            btn_checkout.active = "NO"
            btn_del_item.active = "NO"

            --Separa a string recebida por  linha "\n"
            for c in string.gmatch(new_list, ".-\n") do

                  --Se for a primeira linha ignora
                  if l1 == 1 then
                        --Substitui espacos por branco (trim)
                        d =  trim(c)
                        --Se encontrar uma linha em branco
                        if d == '' then
                              --Seta como fim da lista de arquivos/pastas
```

```
                                    --Apos a linha em branco traz o total de
resultados
                                    fim = 1
                              end
                              --Se ainda tem itens a inserir
                              if fim ~= 1 then
                                    --Verifica se o primeiro caracter é cifrao
("$"). Se for entao é pasta
                                    p = string.sub(c,0,1)
                                    --Se for pasta insere na list_folder
                                    if p == '$' then
                                          list_folder.appenditem = ""..c
                                    else
                                          --Se for realmente um item
                                          if
string.sub(trim(rem_endline(c)),1,22) ~= 'No items found under $' then
                                                --senao insere na list_itens
                                                list_itens.appenditem = ""..c
                                          else
                                                fim = 1
                                          end
                                    end
                              end
                        end
                        --Seta primeira linha OK
                        l1 = 1
                  end
            end --FIM refresh_list

            function trim(b)
                  b = b:gsub("^%s*(.-)%s*$", "%1")
                  return b
            end --FIM trim

            function rem_endline(b)
                  b=b:gsub((string.char(10)), "")
                  return b
            end--FIM rem_endline

            function prompt(c)--executa um comando no prompt e retorna o valor
pra "b"
                  a = io.popen('cmd.exe /c call ' .. c)
                  b = (a:read("*a"))
                  a:close()
                  list_log.appenditem = c
                  retorno = retorno ..'\n'.. c
                  return b
            end --FIM prompt

            function get_fisrt_word(d)
                  aux=0
                  for word in string.gmatch(d, "%w+") do --pega a primeira
sequenca de caracteres ate espaco
                        if aux == 0 then
                              d = word
                              aux = 1
                        end
                  end
                  return d
            end--FIM get_fisrt_word
```

```
	function busca_ss_usr_obj(b)
		d=string.sub(b,20,40)--pega apartir do fim do nome do arquivo
		return get_fisrt_word(d)
	end --FIM busca_ss_usr_obj

	function busca_ss_usr()
		return prompt('ss whoami')
	end --FIM busca_ss_usr

	function busca_os_usr()
		return prompt('echo %username%')
	end --FIM busca_os_usr

	function ss_undo_checkout(c)
		c = 'ss undocheckout '..'"'..c..'"'..' -I-Y'
		return prompt(c)
	end --FIM ss_undo_checkout

	function ss_add(f)
		return prompt('ss add "'.. trim(rem_endline(f)) .. '" -C- -I-
Y')
	end --FIM ss_add

	-----------------------------------------------------------------------
--------------------
	--Menus functions
	-----------------------------------------------------------------------
--------------------
	function item_probls:action()
		iup.Message("Problemas Conhecidos", problemas)
	end

	-----------------------------------------------------------------------
--------------------
	--Buttons functions
	-----------------------------------------------------------------------
--------------------
	function btn_on_off:action()
		b = open_dir('$/')
		refresh_list(b)
		btn_voltar.active = "YES"
		btn_alter_workfold.active = "YES"
		if string.sub(trim(rem_endline(b)),1,1) ~= '$' then
			iup.Message('ERRO','Não foi possível abrir o projeto do
SourceSafe verifique as variáveis de ambiente e tente novamente.\n'..
							'Configurar o path com o
diretorio do SS e SSDIR com o diretorio do arquivo srcsafe.ini.\n'..
							'Pode ser necessário
reiniciar o sitema após setar as variáveis.')
			iup.Close()
		end
	end--FIM btn_on_off:action

	function btn_voltar:action()
		b = open_dir('..')
		refresh_list(b)
	end--FIM btn_voltar:action

	function btn_checkin:action()
		b = prompt('ss status '..'"'.. trim(rem_endline(item)) ..'"')
		b = trim(b)
```

```
            if b ~= msg_no_check and b ~= '' then
                  obj_owner = busca_ss_usr_obj(b)
                  ss_usr = busca_ss_usr()
                  if trim(ss_usr) == trim(obj_owner) then
                        a =   iup.GetText("Comentario","")
                        a = trim(rem_endline(a))
                        if a ~= '' then
                              a = ' -C ' .. '"' .. a .. '"'
                              else
                              a = ''
                        end
                        a = 'ss checkin '..'"'.. trim(rem_endline(item)) ..
'"' .. a .. ' -I-Y'
                        b = prompt(a)
                        b = trim(rem_endline(b))
                  else
                        iup.Message("Erro", 'O arquivo '..
trim(rem_endline(item)).. ' está em check para o usuário '.. obj_owner)
                  end
            else
                  iup.Message("Erro", 'O arquivo
'..trim(rem_endline(item)).. ' não está em checkout.')
            end
      end--FIM btn_checkin:action

      function btn_alter_workfold:action()
            f, err = iup.GetFile("*")
            g, count = f:gsub(string.char(92),string.char(92))
            if err == 0 then

                  f = f:sub(1, (f:len() - string.find(f:reverse(),
string.char(92))))
                  if count == 1 then
                        f = f .. string.char(92)
                  end
                  g = prompt(' ss workfold "' .. f.. '"' )
                  os.execute('taskkill /f /t /im ssexp.exe')
            elseif err == -1 then
                  iup.Message("Cancelada", "Operação cancelada.")
            elseif err == -2 then
                  iup.Message("Erro", "Allocation error")
            elseif err == -3 then
                  iup.Message("Erro", "Invalid parameter")
            end

      end--FIM btn_alter_workfold:action

      function btn_checkout:action()
            b = 'ss checkout '.. string.char(34)..trim(rem_endline(item))
.. string.char(34)
            b = prompt(b)
            if trim(b) == '' then
                  iup.Message("Erro", 'Não foi possivel realizar o Checkout
do arquivo '.. trim(rem_endline(item)) ..'.')
            else
                  if trim(rem_endline(b)) == trim(rem_endline(item)) then
                        iup.Message("OK", "Checkout do arquivo "..
trim(rem_endline(item)) .. " realizado com sucesso.")
                  end
            end
      end--FIM btn_checkout:action
```

```
	function btn_undo_checkout:action()
		b = prompt('ss status '..'"'.. trim(rem_endline(item)) ..'"')
		b = trim(b)
		if b ~= msg_no_check and b ~= '' then
			obj_owner = busca_ss_usr_obj(b)
			ss_usr = busca_ss_usr()
			if trim(ss_usr) == trim(obj_owner) then
				b= ss_undo_checkout(trim(rem_endline(item)))

				if trim(rem_endline(b)) == trim(rem_endline(item))
then
					iup.Message("OK", "Undo Checkout do arquivo
"..trim(rem_endline(item))..." realizado com sucesso.")
				else
					iup.Message("Erro", trim(rem_endline(b)) ..
'\n' .. trim(rem_endline(item)))
				end
			else
				iup.Message("Erro", 'O arquivo '..
trim(rem_endline(item))..' está em check para o usuário '.. obj_owner)
			end
		else
			iup.Message("Erro", 'O arquivo
'..trim(rem_endline(item))..' não está em checkout.')
		end
	end--FIM btn_undo_checkout:action

	function btn_add_new:action()
		f, err = iup.GetFile("*")
		if err == 0 then
			--File already exists
			ss_add(f)
			refresh_list(prompt('ss dir'))--refresh lista atual
		elseif err == -1 then
		iup.Message("Cancelada", "Operação cancelada.")
		elseif err == -2 then
			iup.Message("Erro", "Allocation error")
		elseif err == -3 then
			iup.Message("Erro", "Invalid parameter")
		end
	end--FIM  btn_add_new:action

	function btn_del_item:action()
		prompt('ss delete '.. trim(rem_endline(item)))
		refresh_list(prompt('ss dir'))
	end --FIM btn_del_item:action

	------------------------------------------------------------------------
--------------------
	--Lists functions
	------------------------------------------------------------------------
--------------------
	function list_folder:action(t,i,v)
		if v == 0 then
			btn_add_new.active = "NO"
		else
			btn_add_new.active = "YES"
		end
	end--FIM list_folder:action
```

```
      function list_itens:action(t, i, v)
            if v == 0 then
                  btn_checkin.active = "NO"
                  btn_checkout.active = "NO"
                  btn_undo_checkout.active = "NO"
                  btn_del_item.active = "NO"
            else
                  btn_checkin.active = "YES"
                  btn_checkout.active = "YES"
                  btn_undo_checkout.active = "YES"
                  btn_del_item.active = "YES"
                  item = t
            end
      end--FIM list_itens:action

      function list_folder:dblclick_cb(i, t)
            b = open_dir('"'..trim(rem_endline(t))..'"')
            refresh_list(b)
            return iup.DEFAULT
      end--FIM list_folder:dblclick_cb

      ------------------------------------------------------------------------
--------------------
      --Checks functions (toggle)
      ------------------------------------------------------------------------
--------------------
      function tg_log:action()
            list_log.visible = "NO"
            frm_log.size = "10x100"
      end

      ------------------------------------------------------------------------
--------------------
      --MainLoop
      ------------------------------------------------------------------------
--------------------
      if (iup.MainLoopLevel()==0) then
        iup.MainLoop()
        print(retorno)
      end
end
return vss
```