UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

APLICAÇÃO DE GERÊNCIA DE OBJETOS COMPARTILHÁVEIS DE UM SGBD ORACLE

ANTONIO DE OLIVEIRA JUNIOR

BLUMENAU
2006

2006/1-02

ANTONIO DE OLIVEIRA JUNIOR

APLICAÇÃO DE GERÊNCIA DE OBJETOS COMPARTILHÁVEIS DE UM SGBD ORACLE

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

Prof. Alexander Roberto Valdameri, MEP - Orientador

BLUMENAU
2006

2006/1-02

APLICAÇÃO DE GERÊNCIA DE OBJETOS COMPARTILHÁVEIS DE UM SGBD ORACLE

Por

ANTONIO DE OLIVEIRA JUNIOR

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

Presidente:	Alexander Roberto Valdameri, MEP – Orientador, FURB
Membro:	Prof. Nome do professor, Titulação – FURB
Membro:	Prof. Nome do professor, Titulação – FURB

Blumenau, 07 de Junho de 2006

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram na realização deste.

AGRADECIMENTOS

A minha noiva Adriana Thom Zimmermann pela sua compreensão e paciência durante a realização deste trabalho.

Aos meus amigos Edney Imme, Fernando Lunelli, Fabiano Rosa e Marcos Gorll que me auxiliaram a concretizar a conclusão do curso.

Ao meu orientador e amigo, Alexander Roberto Valdameri, por ter acreditado na conclusão deste trabalho.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

RESUMO

Neste trabalho é apresentada a especificação e a implementação de uma aplicação para gerenciar objetos compartilháveis de um SGBD Oracle, também contendo a funcionalidade de um ambiente de compartilhamento de código dos objetos no banco de dados. Os objetos compartilháveis do banco de dados Oracle são utilizados para a automatização de processos visando garantir a consistência dos dados.

Palavras-chave: Objetos compartilháveis. Gerenciamento. Oracle.

ABSTRACT

This work shows the specification and implementation of a shared object manager to the Oracle database, also are available an environment to share the source code of the objects at the database. The Oracle shared objects are used to improve the consistence of the data stored on database.

Key Words: Shared Objects. Manage. Oracle.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AD – Administrador de dados

DBA – *Database administrator*

DML – *Data manipulation language*

JDE – *Java Development Environment*

MER – Modelo de Entidade Relacionamento

MVC – *Model-View-Controler*

SGBD – Sistema Gerenciador de Banco de Dados

SQL – *Structure Query Language*

UML – Linguagem de Modelagem Unificada

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Em um mundo corporativo os softwares e os procedimentos para criação dos mesmos estão sendo a cada dia aprimorados, objetivando assim, acelerar o processo de desenvolvimento de novas aplicações e de auxiliar na manutenção de aplicações já existentes.

Aplicações criadas internamente em um banco de dados fazem com que rotinas validem os dados diretamente no banco, sem necessitar de rotinas em nível de aplicação, como é o caso de rotinas criadas com a linguagem *Procedural Language Structure Query Language* (PL/SQL) da Oracle. Estas rotinas transformam-se em objetos que por sua vez podem ser compartilhados e utilizados em qualquer aplicação. Dentre os objetos disponíveis para a criação de aplicações estão *packages*, *procedures*, *triggers*, *functions* e *views*, os quais são objetos abordados neste trabalho.

O Sistema Gerenciador de Banco de Dados (SGBD) Oracle é largamente utilizado em empresas de grande porte. Tais empresas em geral utilizam-se das funcionalidades do banco, criando assim objetos que automatizam processos e possibilitam que os dados armazenados no banco passem por validações que certificariam a consistência da informação (Bumerang, 2005).

Após o desenvolvimento de tais objetos, faz-se necessário realizar o processo de entrega do software, isto é, a liberação das rotinas de tratamento de dados que irão interagir com as aplicações. Em caso de utilização de objetos de um banco de dados faz-se necessário um *Database Administrator* (DBA). Este coletará todas as informações pertencentes aos programas internos do banco antes da liberação e, caso haja problemas com os novos

programas, o DBA deverá desfazer todo o fluxo de entrega dos objetos referentes ao processo, realizando assim o retorno dos objetos antigos e suas informações sobre direitos de acesso.

Para atender a demanda e auxiliar o desenvolvimento, foi desenvolvido um software para o gerenciamento de objetos compartilhados do banco de dados Oracle, fornecendo assim um controle de alocação de código, um local para armazenamento de fontes e registrando e consistindo as informações pertencentes aos programas PL/SQL da Oracle.

O trabalho será especificado utilizando o padrão de projeto chamado *Model-View-Controler* (MVC), que determina uma programação em camadas. Cada camada é responsável por uma tarefa, estas tarefas são distribuídas entre a camada modelo que realiza a comunicação com a base de dados, camada de controle que contém as regras de negócio e a camada de visualização que exibe as informações na tela.

1.2 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é criar um software para disponibilizar um ambiente de manutenção, realizando liberação e validação de objetos do banco de dados Oracle, e garantindo que o processo seja executado corretamente em horários previamente agendados.

Os objetivos específicos do trabalho são:

a) permitir uma automatização de liberação de objetos;

b) adicionar a funcionalidade de agendamento de liberações;

c) garantir que a liberação efetuada não gere impactos em outros objetos, indisponibilizando algumas funcionalidades dos objetos do banco de dados;

d) tornar desnecessária a alocação do administrador dos dados da empresa, de estar disponibilizando os objetos no ambiente;

e) possibilitar a restauração do ambiente em caso de insucesso na liberação de

objetos;

f) permitir atualização de múltiplos ambientes.

1.3 MOTIVAÇÃO

Devido à utilização de recursos computacionais para o auxílio nas tomadas de decisões, a necessidade de liberar novas aplicações para um ambiente de produção torna-se altamente crítica. Visto que um erro na liberação de objeto do banco de dados Oracle poderá comprometer todas as aplicações que de alguma forma utilizam os objetos afetados.

Outro fator relevante é que com esta ferramenta, torna-se fácil para um profissional disponibilizar novos objetos no ambiente corporativo. O usuário desta ferramenta não necessitará conhecer informações referentes aos objetos e nem como conseguir tais informações, com o processo automatizado dar-se-á agilidade e garantia de que o ambiente ficará valido após a entrega, tornando a empresa mais sensível às mudanças do mercado.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está divido em capítulos conforme descritos a seguir.

O primeiro capítulo descreve a justificativa no que diz respeito à origem do trabalho, também é exposto uma síntese do que será tratado o trabalho, e também os objetivos a serem alcançados.

O segundo capítulo aborda os mecanismos do SGBD e também engloba os conceitos de objetos compartilháveis com ênfase aos objetos do banco de dados Oracle.

O terceiro capítulo apresenta as etapas do desenvolvimento da ferramenta, bem como

utilização do software desenvolvido.

O quarto capítulo não somente expõem as conclusões, considerações mas também é sugerido algumas sugestões para a continuação do trabalho.

2 REVISÃO BIBLIOGRAFICA

Neste capítulo serão abordados os assuntos referentes a sistemas gerenciadores de banco de dados, padrão de projeto e também os objetos compartilháveis, dando ênfase a objetos compartilháveis do banco de dados Oracle.

2.1 SISTEMAS GERENCIADORES DE BANCO DE DADOS

Conforme (DATE, 1991, p.5) o banco de dados é um simples sistema de manutenção de registros. Trata-se de um sistema com o objetivo de manter as informações ou torná-las disponíveis quando solicitadas, entretanto, tratando-se do banco de dados Oracle utilizado neste trabalho, além destas funcionalidades também pode-se dizer que o Oracle é composto por um Sistema Gerenciador de Banco de Dados (SGBD).

Conforme (DATE, 1991, p.40), o SGBD é o software que gerencia os acessos, privilégios e também analisa as solicitações destinadas ao banco de dados, define também uma interface de comunicação com o usuário além de fazer vários controles para a garantia da integridade das informações.

Segundo Luchtenberg (2002), o banco de dados Oracle pode ser considerado um SGBD por suas características e algumas características que o SGBD Oracle proporciona é conter objetos que são escritos na linguagem PL/SQL.

Estes objetos que estão no banco de dados são objetos compartilháveis entre os usuários. As aplicações que acessam o banco de dados disparam as execuções dos objetos e quando isto acontece o código que compõe o corpo dos objetos é executado.

2.2 OBJETOS COMPARTILHAVEIS

Conforme Ault (1997, p. 109), as *triggers* são códigos PL/SQL contidos no banco de dados que são executados quando um determinado evento especificado no corpo da *trigger* ocorre. Segundo (DAMIANO, 1995, p. 91) estes eventos tratam-se de simples comandos de *Data Manipulation Language* (DML), que são comandos utilizados para a manipulação dos dados dentro de um banco de dados.

Para entender melhor esta situação, pode-se citar um sistema de folhas de pagamento, e que ao inserir as horas de um funcionário é necessário que se realize alguma validação. Um exemplo seria enviar uma requisição ao setor de pagamentos da empresa para que seja efetuado o pagamento. Não é necessário que uma aplicação externa faça esta requisição, simplesmente adicionando uma *trigger* no banco esta situação seria resolvida.

A lógica da aplicação pode ser muito complexa para uma simples *trigger*. Neste caso, pode-se reutilizar a lógica em outra *trigger* para outra aplicação. Assim, o uso de uma *procedure* ou *function* é requerido, fazendo com que o código fique armazenado em um objeto separado da *trigger*.

Em Ault (1997, p. 132) sugere-se que sejam colocados todas as *procedures* e *functions* para uma determinada aplicação dentro de uma *package* concentrando assim as atividades da aplicação em um único objeto.

Outro objeto utilizado no banco de dados é a *view* (Figura 6). Este objeto é uma tabela lógica composta por uma ou mais tabelas ou *views*, de fato a *view* não contém dados por si só, realizando uma busca sempre nas tabela que é composta. A *view* possibilita que você crie uma forma customizada para apresentar os dados, e também pode disponibilizar o recurso de inserir dados através de uma referência a este objeto (DAMIANO, 1995, p. 56).

Todos estes objetos por serem utilizados por diversos usuários conectados no banco de

dados formam um aglomerado de objetos compartilháveis que são acessados constantemente durante a utilização dos dados.

Considerando o exemplo da folha de pagamento anteriormente abordado, no momento em que o programa do usuário estiver inserindo dados no banco, pode-se considerar um instante crítico para realizar manutenções no banco de dados, especialmente na *trigger*, que dispara o processo de envio de requisições para o setor de pagamento. Caso esta *trigger* não seja executada não será enviado nenhum pedido e a lógica do programa estaria comprometida.

Neste caso se houver uma liberação de versão da *trigger* de banco, este momento seria crítico e impraticável de realizar a liberação, deveria ser agendado a entrega da nova versão para um outro horário fora do período em que se esta alterando a base, ou que seja feita uma parada momentânea para efetivar a liberação da *trigger*.

2.3 PADRÕES DE PROJETO *MODEL-VIEW-CONTROLER*

Segundo Gama (2000), o padrão *Model-View-Controler (*MVC) separa os elementos de um projeto em três objetos distintos: *Model* (Modelo), *View* (Visão) e *Controler* (Controlador). A camada de modelo é definida por conter os dados e ser responsável pelos objetos da aplicação, a camada de visão é a camada que o usuário estará acessando e a camada de controle é a responsável por reagir de acordo com a entrada dos dados do usuário. Este modelo distribui responsabilidades para cada camada fazendo com que ganhe mais flexibilidade e que seja mais reutilizável.

O *Singleton* é um padrão de projeto, este padrão determina que uma classe contém somente uma instância durante toda a execução do programa (GAMA, 2000).

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são apresentados os requisitos, a especificação e o desenvolvimento do trabalho.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Abaixo estão os requisitos levantados para que a ferramenta possa exercer o papel de aplicação de gerência de objetos compartilháveis de um Sistema Gerenciador de Banco de Dados (SGBD) Oracle.

Os requisitos funcionais são:

a) disponibilizar um ambiente com controle de compartilhamento de códigos;
b) fornecer a opção de validação de objetos em um banco de dados de testes;
c) prover a funcionalidade de entregar os objetos de banco com garantia de compilação e assim mantendo o banco de dados funcional;
d) efetuar um retorno da situação anterior a liberação, caso ocorram erros no processo de entrega dos objetos;
e) agendar liberação de objetos;
f) armazenar as informações referentes aos objetos de banco que foram liberados;
g) disponibilizar informações sobre o processo de liberação de objetos realizada;
h) viabilizar a liberação dos seguintes objetos de banco: *packages*, *procedures*, *triggers*, *functions* e *views* escritos com a linguagem PL/SQL da Oracle.

Os requisitos não funcionais são:

i) obter todas as informações dos objetos que estão no banco de dados e que podem ser afetados antes de qualquer liberação;

j) tratar erros que são provenientes da base de dados Oracle para disponibilizar ao usuário detalhes dos erros de compilação do mesmo;

k) ser desenvolvido em Java.

3.2 ESPECIFICAÇÃO

Para realizar a especificação do software foi utilizada a ferramenta *Enterprise Architect* através da linguagem UML. Também foi utilizada a ferramenta *Sybase Power Designer* para criar o modelo de entidade relacionamento.

3.2.1 DIAGRAMAS DE CASOS DE USO

A seguir (figura 1) é apresentado o diagrama de caso de uso referente aos pacotes de cada modulo do software.

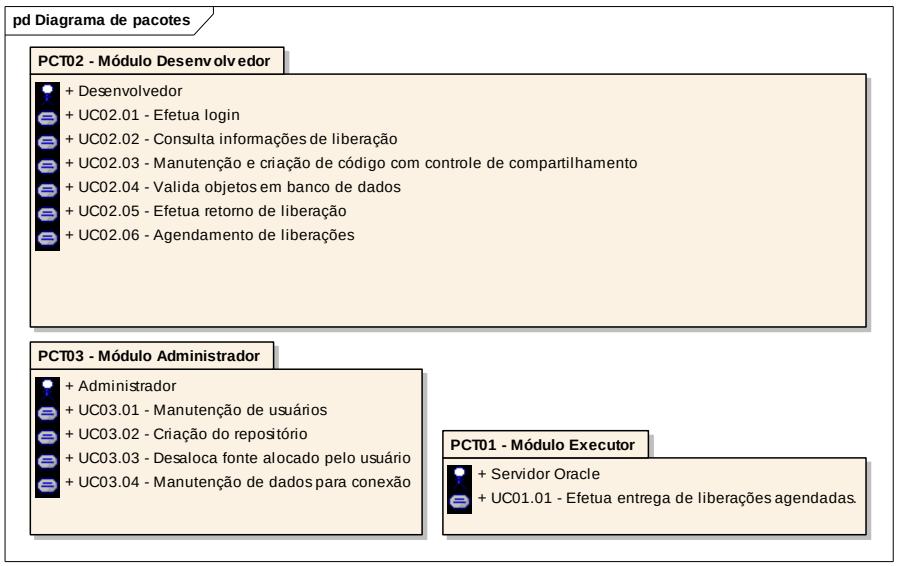


Figura 1 - Diagrama de caso de uso “pacotes do programa”

Os três módulos representam o programa por completo, cada módulo esta sendo

apresentado a seguir, como é o caso do módulo executor (Figura 2).

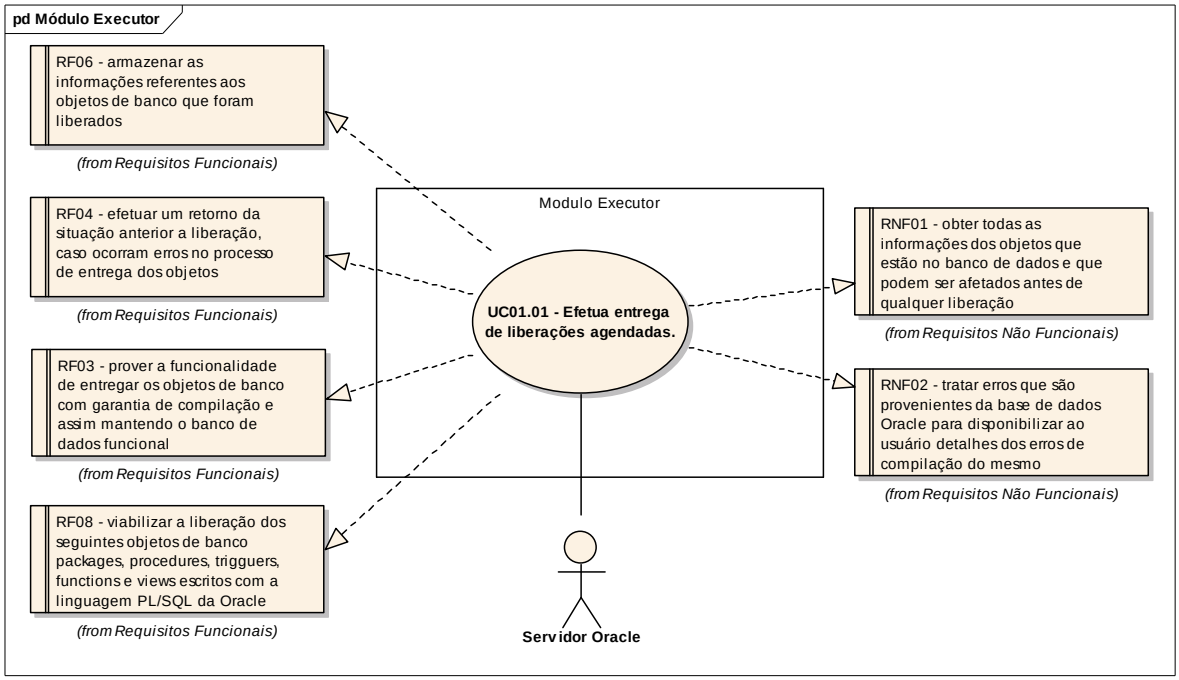


Figura 2 - Diagrama de caso de uso "módulo executor"

Este módulo contempla a entrega das liberações agendadas e garante os requisitos funcionais descritos. A seguir tem-se o diagrama de atividades referente a liberação dos objetos.

A figura 3 contempla o módulo desenvolvedor que fará a maior parte dos controles do software.

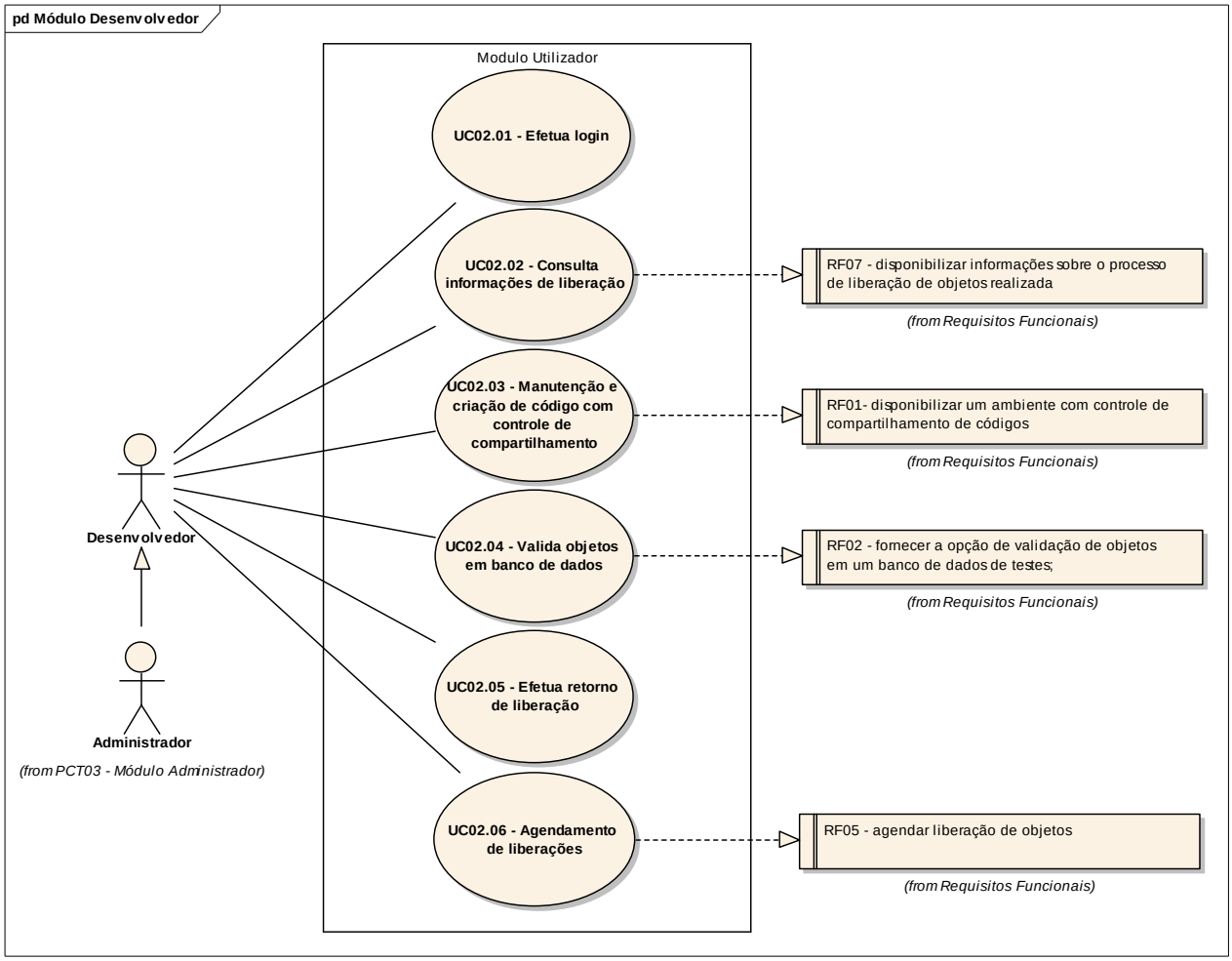


Figura 3 - Diagrama de caso de uso “módulo desenvolvedor”

Por último pode-se verificar (figura 4) o módulo do administrador. Além de conter as funcionalidades do desenvolvedor herdando-as (ver figura 3), o administrador contém a manutenção dos usuários, cadastro de banco de dados, controle de alocação do código e a criação do repositório.

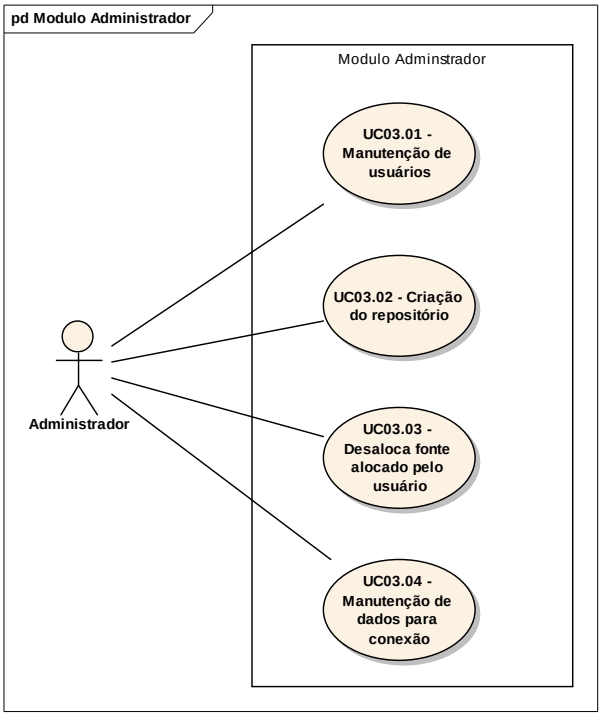


Figura 4 - Diagrama de caso de uso “módulo administrador”

Em seguida estão detalhados os casos de uso que foram citados acima.

3.2.2 DESCRIÇÃO DOS CASOS DE USO

Nos quadros abaixo apresentam-se informações detalhadas dos casos de uso. Cada caso de uso está identificado pelo número de referência nas figuras dos módulos anteriores:

UC02.01 – Efetua login
O programa apresentará uma tela contendo dois campos a serem preenchidos, o login e a senha, o usuário digita o login e a senha e o programa valida estas informações na base, se o usuário e senha estiverem corretos será apresentada a tela principal do programa, caso contrário uma mensagem apresentará as informações que os dados estão incorretos.

Quadro 1 – Descrição do caso de uso “efetua login”

UC02.02 – Consulta informações de liberação
O desenvolvedor acessa a tela de consulta de informações, então o programa apresenta uma listagem que referência cada liberação que foi ou que será realizada. O desenvolvedor seleciona uma liberação e as informações referentes são apresentadas, o usuário pode remover a referencia selecionando a opção de remover disponível na tela.

Quadro 2 – Descrição do caso de uso “Consulta informações de liberação”

UC02.03 – Manutenção e criação de código com controle de compartilhamento
Na tela principal do programa o desenvolvedor acessa o menu arquivo, e identifica a opção que deseja realizar: a) caso o desenvolvedor selecione a opção de um novo fonte, uma área de texto estará disponível na tela principal e poderá ser editada, as informações ali inseridas são os fontes dos objetos compartilháveis do banco de dados; b) caso o desenvolvedor selecione a opção de abrir o fonte uma tela contendo o nome dos códigos fontes já criados será exibida, o desenvolvedor terá a opção de escolher e abrir o código na tela principal ou remover o código fonte da base de dados, entretanto se o fontes estiver sendo utilizado por outro usuário não será possível efetuar nenhum ação com o código fonte escolhido; c) caso o desenvolvedor selecione a opção de salvar fonte, se houver código fonte a ser salvo e ou o código fonte não conter um nome para referência, será requisitado para que o desenvolvedor digite exatamente o nome do código fonte que deseja salvar, o fonte deverá ser o nome do objeto compartilhável do banco que o usuário esta criando; d) o desenvolvedor também poderá selecionar a opção fechar fonte, que irá retirar a área de texto disponível para edição de códigos fontes da tela principal.

Quadro 3 – Descrição do caso de uso “Manutenção de código e compartilhamento”

UC02.04 – Valida objetos em banco de dados
O desenvolvedor acessa a tela de compilação, é então apresentado os bancos de dados cadastrados no sistema através de uma listagem e também dois campos onde serão digitados o usuário e a senha , o desenvolvedor seleciona o banco de dados e digita o usuário e senha para testar a compilação do programa que esta sendo desenvolvido, o sistema testa a compilação tentando efetuar a criação do objeto no banco. Após a criação do objeto no banco é identificado se houve algum erro com a criação do objeto, caso exista uma mensagem referente ao processo de criação é informada no campo de detalhes de compilação na tela.

Quadro 4 – Descrição do caso de uso “Valida objetos em banco de dados”

UC02.05 - Efetua retorno de liberação
O desenvolvedor acessa a tela de consulta de informações, então o programa apresenta uma listagem que referência cada liberação que foi ou que será realizada. O desenvolvedor seleciona uma liberação e as informações referentes são apresentadas, o usuário pode retornar liberação selecionando a opção de retorno disponível na tela.

Quadro 5 – Descrição do caso de uso “Efetua retorno de liberação”

UC02.06 – Agendamento de liberações
O desenvolvedor acessa a tela de agendamento, então uma listagem de códigos criados é apresentada, o desenvolvedor cria uma listagem de códigos que serão liberados, também é apresentada uma listagem de bancos de dados Oracle já cadastrada que deverá ser selecionado para a liberação. Também é requerido ao desenvolvedor digitar o usuário e a senha do banco de dados. Tais informações são úteis para a liberação assim como o dia, mês, ano, hora e minuto que o objeto deverá ser liberado. Por fim, informar quanto tempo que o processo de liberação aguardará se o objeto de destino estiver sendo usado por um outro usuário, também é necessário informar uma referência para identificar a liberação.

Quadro 6 – Descrição do caso de uso “Agendamento de liberações”

UC03.01 – Manutenção de usuários
O administrador ao acessar a tela de manutenção de usuários, é apresentada uma listagem de contas para acesso ao programa, como também campos para informar um novo usuário, o administrador identifica a ação que deseja realizar: a) para o administrador inserir um novo usuário basta preencher os campos de login, senha, usuário e função e selecionar a opção de salvar; b) caso o administrador selecione um usuário da listagem, e selecionar a opção de alterar os dados do usuário preencherão as informações de login, senha, usuário e função que ao entrar na pagina estiveram vazias, o administrador pode então efetuar a alteração no cadastro e salvar a alteração; c) caso o administrador selecione um usuário da listagem, e selecione a opção de deletar o usuário será então removido da base de dados;

Quadro 7 – Descrição do caso de uso “Agendamento de liberações”

UC03.02 – Criação do repositório
O sistema é configurado para ser executado em um novo ambiente, então o administrador requisita fazer o seu primeiro login no sistema, a tela de login com o usuário e senha é apresentada. O administrador digita o usuário e senha padrão e seleciona a opção para entrar no sistema, o sistema identifica que não há as informações das tabelas na base de dados e informa ao usuário que serão criadas tabelas novas criado assim um repositório de informações, o administrador confirma a criação e o repositório é criado.

Quadro 8 – Descrição do caso de uso “Criação do repositório”

UC03.03 – Desaloca código fonte alocado pelo usuário
O administrador acessa a tela de desalocar fontes e uma listagem de fontes é informada, o administrador então seleciona o código fonte que deseja desalocar, o sistema apresenta informações do usuário que esta alocando e questiona se deseja confirmar a ação que deseja realizar: a) caso o administrador selecione desalocar o usuário o fonte será sinalizado como desalocado e poderá ser editado por outro usuário; b) se o administrador selecionar não desalocar o fonte nenhuma ação será realizada.

Quadro 9 – Descrição do caso de uso “Desaloca código fonte alocado pelo usuário”

UC03.04 – Manutenção de dados para conexão
O administrador ao acessar a tela de manutenção de fonte de dados, é apresentada uma listagem de banco de dados, como também um campo para informar uma configuração de acesso a banco de dados, o administrador identifica a ação que deseja realizar: a) para o administrador inserir um novo banco de dados basta preencher o campo de banco de dado e selecionar a opção de salvar; b) caso o administrador selecione um banco de dados da listagem, e selecionar a opção de alterar os dados será preenchida a informação do banco de dados que ao entrar na pagina estava vazia, o administrador pode então efetuar a alteração no cadastro e salvar a alteração; c) caso o administrador selecione um banco de dados da listagem, e selecione a opção de deletar a referência do banco de dados será então removido da base de dados;

Quadro 10 – Descrição do caso de uso “Manutenção de dados para conexão”

A seguir é apresentado o diagrama de atividades referente ao caso de uso que não foi abordado neste capítulo por se tratar da parte mais complexa do trabalho.

3.2.3 DIAGRAMA DE ATIVIDADES

A figura 5 apresenta o diagrama de atividades referente ao processo de entrega do objeto. Esta parte da entrega dos objetos foi descrita pelo diagrama de atividades por se tratar da parte principal do trabalho.

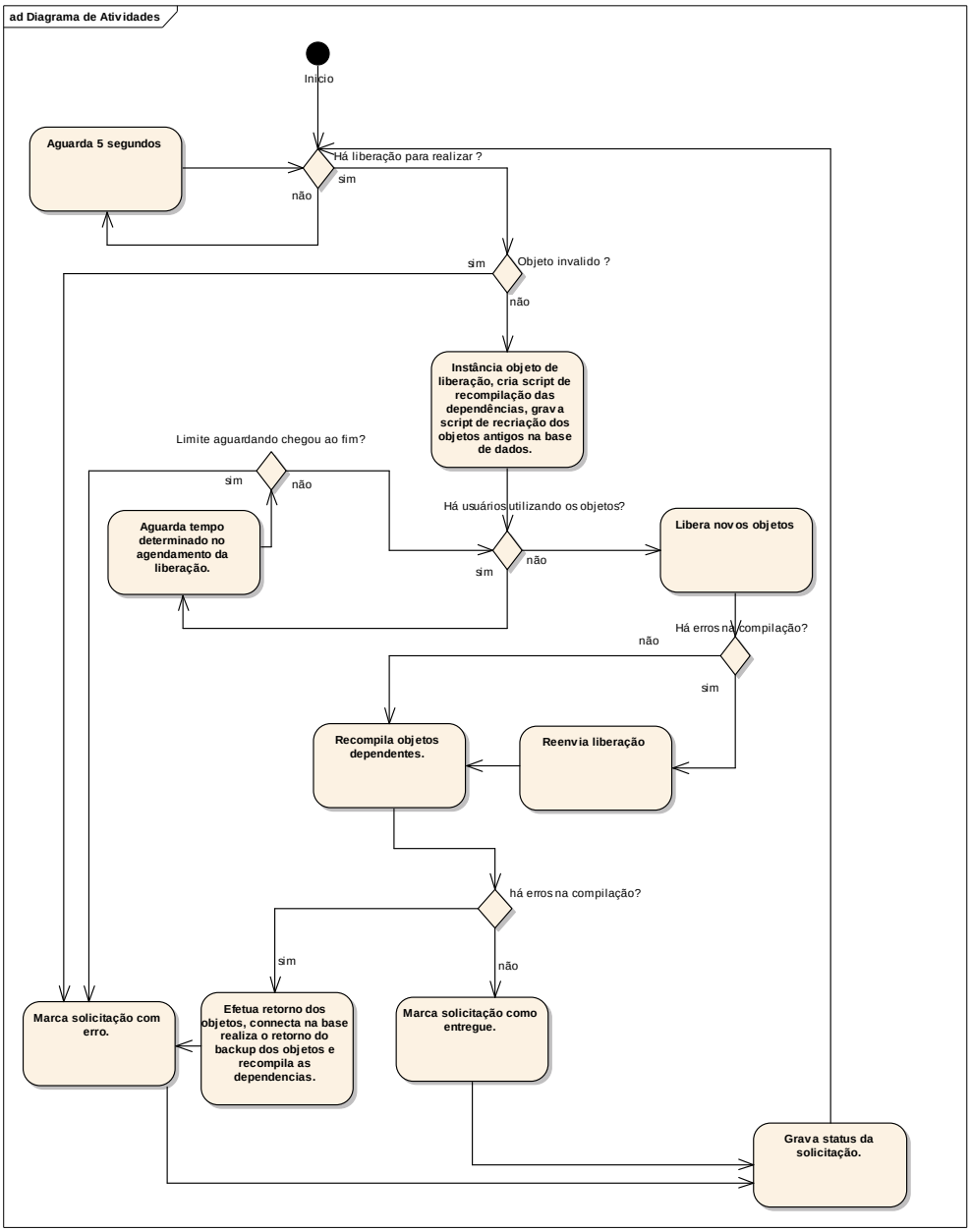


Figura 5 - Diagrama de atividades do caso de uso que efetua entrega das liberações

3.2.4 DIAGRAMA DE CLASSES

A especificação foi realizada baseada em camadas, das quais pode-se citar a camada de modelo. Esta camada refere-se aos objetos de conexão com o banco. Tais objetos realizam o controle dos dados e também são responsáveis pela representação dos objetos durante a execução do programa.

A figura 6 descreve o diagrama de classes referentes ao modelo.

Figura 6 - Diagrama de classes “camada de modelo”

A seguir serão descritas as classes da camada de modelo:

a) FonteDados: esta classe contém as informações para conexão nos bancos de dados, também é responsável por manter estas informações;

b) Liberacao: é uma classe responsável por realizar a liberação, esta classe chama as classes FonteDados e Fonte, para efetuar a liberação do fonte;

c) Fontes: contém os códigos fontes, informação se está sendo alocada por um usuário ou não e mantém os dados referentes aos fontes;

d) Agendamento: o agendamento não só contém como também mantém as informações a respeito do agendamento e instancia o objeto de Liberacao no momento da liberação;

e) Usuário: o objeto de usuário é uma classe abstrata que existe para forçar a implementação dos métodos de administrador e desenvolvedor;

f) Administrador: classe utilizada para manter os dados do administrador, porem também contem a funcionalidade de instanciar um objeto de repositório, esta classe herda as informações da classe abstrata Usuario, isto faz com que implemente os métodos da classe pai Usuário;

g) Desenvolvedor: classe que mantém os dados referentes ao desenvolvedor, esta classe também herda as informações da classe abstrata Usuario, isto faz com que implemente os métodos da classe pai Usuario.

Outra camada é a de controle (figura 7), que contém todas as regras de negócios referentes ao programa. A mesma faz a comunicação entre as telas da camada de visualização e a de modelo.

Pode-se também reparar que nesta camada de controle, existe um objeto que faz o controle da conexão dos dados. Isto se faz necessário em virtude da preocupação em evitar que não haja várias conexões durante a execução do programa.

Estas classes são instanciadas somente uma fez durante a execução do programa, fazendo com que elas sejam instâncias únicas durante a execução do programa.

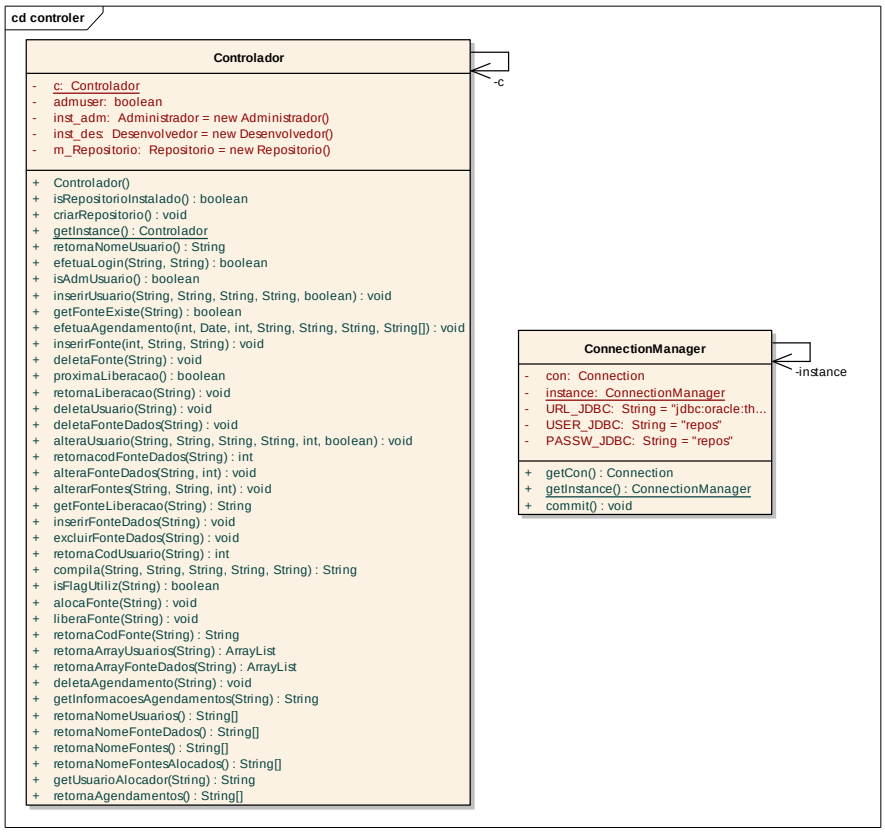


Figura 7 - Diagrama de classes “camada de controle”

A seguir serão descritas as classes da camada de controle:

a) Controlador: o controlador contém todos os métodos para fazer a conversação entre a camada de visualização, e também com a camada de modelo que contém os dados de todo o sistema, esta camada é responsável por garantir as funcionalidades do programa, é a camada conhecida como a de regra de negócio, esta classe é considerada Singleton, pois somente há uma instância desta classe durante toda a execução do programa;

b) ConnectionManager: classe responsável por controlar a conexão durante toda a execução do programa, esta classe também é considerada Singleton, pois somente há uma instância desta classe durante toda a execução do programa.

A camada de visualização (figura 8), estas classes são as telas propriamente ditas, as telas instanciam a classe de controle para poder executar as regras de negócios.

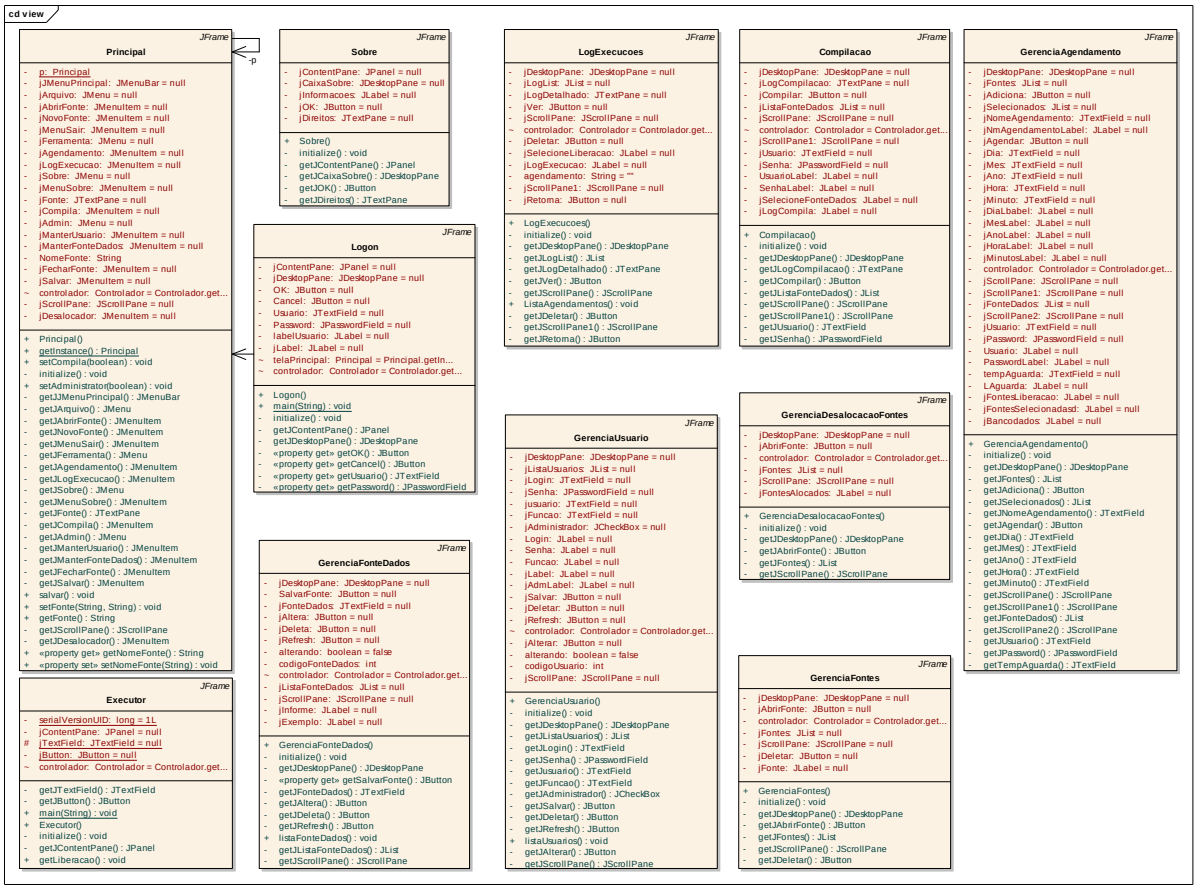


Figura 8 - Diagrama de classes “camada de visualização”

A seguir serão descritas as classes da camada de visualização:

a) Principal: é a classe da tela principal, como esta classe é uma tela principal e pode ser referenciada por outra classe também desta camada, a classe Principal foi também considerada um Singleton, por motivos de referencia;

b) Logon: é a classe que chama a classe Principal e instancia o controlador para receber informações se chamará ou não a classe Principal;

c) Sobre: classe que apresenta uma tela com informações do produto;

d) GerenciaFonteDados: classe responsável por visualizar e requisitar manter a fonte de dados ou cadastro de banco de dados;

e) GerenciaUsuario: classe que visualiza e requisita alterações para o objeto Controlador as informações são referentes a manutenção do usuário;

f) Executor: O executor faz parte do programa tccexecutor.jar, o programa executor

chama o controlador para efetuar a chamada dos métodos de liberação, esta tela somente mostrará informações na liberação;

g) LogExecucoes: esta classe chama o controlador para obter os dados referentes as liberações e também envia requisições para retorno ou remoção do agendamento;

h) Compilacao: esta classe chama uma instância da tela principal para obter o fonte do programa que está sendo editado, também chama o controlador passando o código a ser compilado, o controlador faz a execução da criação do código e retorna o sucesso ou os erros que ocorreram com a compilação;

i) GerenciaDesalocacaoFontes: esta classe instancia o controlador para obter informações a respeito de códigos fontes alocados, e também requisita ao controlador que eles sejam desalocados conforme requisição do usuário;

j) GerenciaFontes: contém as chamadas para o objeto controlador e apresenta uma listagem de fontes que estão disponíveis no banco de dados, além desta funcionalidade pode-se deletar o fonte ou abrir, para isso é requisitado ao controlador as informações como o código do fonte e o nome do mesmo, estas informações são repassadas ao objeto Principal, obtida a referência através de uma chamada para este Singleton;

k) GerenciaAgendamento: recupera uma instância do objeto controlador, chamando os métodos para cadastrar um agendamento que deverá ser executado pelo módulo executor.

Pode-se perceber que na camada de visualização existe a tela principal. Esta interface é instanciada por algumas telas que necessitam executar métodos remotamente.

3.2.5 MODELO DE ENTIDADE RELACIONAMENTO

A persistência dos dados referentes aos códigos fontes, informações de conexão, dados da liberação, controle de acesso e todos estes dados referentes à utilização do programa são armazenados em um banco de dados. Para tal foi criado um modelo de entidade relacionamento no banco de dados Oracle. Na figura 9 é apresentando o modelo mapeado na estrutura.

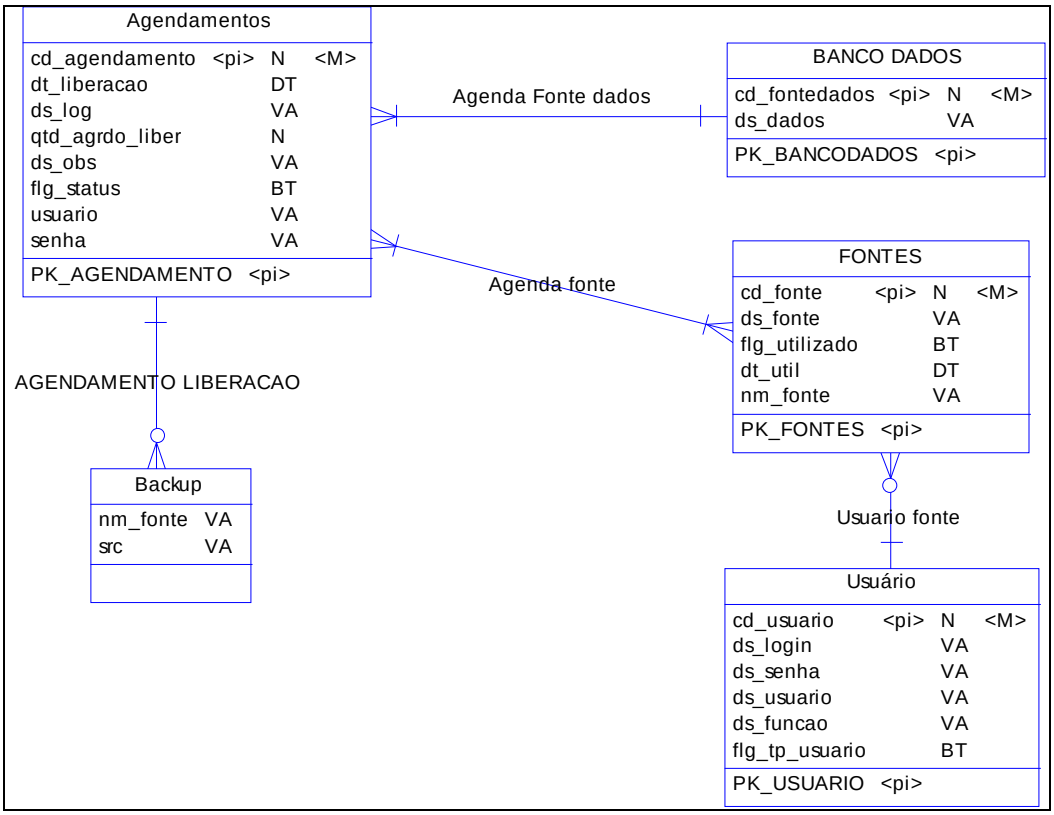


Figura 9 - Modelo de entidade e relacionamento

O modelo apresentado é o modelo conceitual referente ao repositório que compõe este trabalho. Nota-se que tanto a entidade de usuário quanto a entidade agendamento contém um campo de usuário e senha. Porém estas informações são armazenadas por diferentes motivos. Na entidade de usuário estas informações são armazenadas para validar a entrada do usuário no sistema, já na entidade do agendamento esta informação é usada para efetuar o *logon* no banco de dados quando for realizada a liberação. A seguir é demonstrado passos sobre a

implementação do trabalho.

3.3 IMPLEMENTAÇÃO

Nesta etapa estão definidas as técnicas e ferramentas utilizadas para o desenvolvimento do trabalho.

3.3.1 Técnicas e ferramentas utilizadas

O desenvolvimento do software começou com a criação dos casos de uso e dos diagramas de classe no *Enterprise Architect*. As classes foram geradas e então importadas na *Java Developer Environment* (JDE) Eclipse para o desenvolvimento.

Com as classes uma vez importadas foram criadas as camadas de controle, modelo e visualização. Os programas exportados do *Enterprise Architect* tornaram-se a camada de modelo, sendo esta responsável por obter os dados que serão utilizados no decorrer do programa, disponibilizados em forma de objetos. A descrição completa do método para efetuar a liberação encontra-se no Apêndice A. No Apêndice B encontra-se o método de retorno da liberação dos objetos, referenciados no decorrer do programa.

No Eclipse foi desenvolvida a camada de controle. Esta camada é definida por conter as regras de negócios. As regras de negocio são as funcionalidades e lógicas necessárias para o programa funcionar, é na camada de controle que se localizam os métodos referentes a regra de negócio que serão chamados entre as camadas de modelo e visualização.

Uma vez tendo a camada de controle e de modelo implementada, faz-se necessário interagir com o usuário, isto é feito através da camada de visualização.

A camada de visualização foi escrita com um pacote de ferramentas chamada Editor

Visual. O Editor Visual foi instalado na JDE Eclipse para facilitar o desenho das telas em Java. As telas contemplam a camada de visualização. Nesta camada o único objeto que é referenciado é o objeto que contém as regras de negócio, encontrado na camada de controle.

Somente uma instância de cada objeto da camada de controle pode existir. A camada de controle é composta por dois objetos. Um objeto é o gerenciador das conexões. Este objeto é responsável por manter a conexão durante a execução do programa fazendo com o que não seja necessário ficar efetuando conexões no decorrer do uso do programa.

O outro objeto é o controlador que é o programa que contém as funcionalidades, cabe ao controlador transmitir dados para a camada de visualização. Outra funcionalidade é enviar requisições para os objetos de acordo com a necessidade, por exemplo, é necessário cadastrar um usuário para acessar o programa. A camada de visualização chama o método da instância de controle responsável pelo cadastro de usuário. O objeto de controle chama uma instância do usuário. Na instância de usuário está implementado o método de inclusão de usuário. Este método é então executado com os dados necessários para a inclusão e o objeto de usuário efetuará o cadastro.

3.3.2 Codificação

Como mencionado anteriormente, a camada de controle é a responsável pelas regras de negócio. Pode-se identificar esta funcionalidade com o código contido no quadro 11.

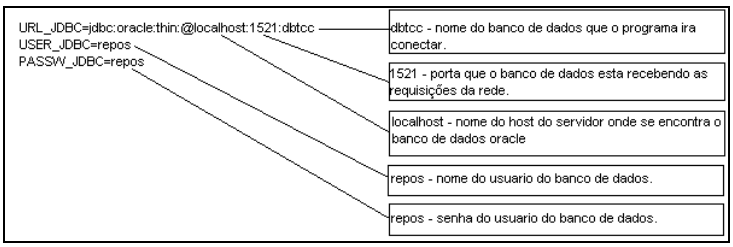
```
/**
 * Efetua o Logon da aplicação verificando de acordo com os objetos.
 * @throws SQLException
 * @throws ClassNotFoundException
 * @throws IOException
 */
public boolean efetuaLogin(String p_login, String p_senha) throws SQLException, ClassNotFoundException, IOException  {

    boolean wrk_valida = false;

    if (m_Repositorio.retornaUsuario(p_login) instanceof Desenvolvedor) {
        Desenvolvedor instancia = (Desenvolvedor) m_Repositorio.retornaUsuario(p_login);
        // Não considera usuario como administrador.
        this.admuser = false;
        // Variavel local recebe instancia
        this.inst_des = instancia;
        if (p_senha.equals(instancia.getDsSenha())){
            wrk_valida = true;
        }
    }else{
        Administrador instancia = (Administrador) m_Repositorio.retornaUsuario( p_login);
        // Considera usuario como administrador.
        this.admuser = true;
        // Variavel local recebe instancia
        this.inst_adm = instancia;
        if (p_senha.equals(instancia.getDsSenha())){
            wrk_valida = true;
        }
    }
    return wrk_valida;
}
```

Quadro 11 - Código fonte logon da aplicação, localizado na camada de controle

A classe *Logon* chama um método para instanciar a classe do controlador. Para efetuar o *login* do usuário, a classe Logon que está na camada de visualização envia uma requisição para o controlador informando o *login* e a senha do usuário.

Conforme o código exibido no quadro 11, o controlador requisita ao objeto do repositório o retorno de um usuário, que pode ser um objeto Administrador ou Desenvolvedor. Existe um tratamento para cada objeto, não há nenhuma conexão com o banco de dados ou comando DML em toda a camada do controlador. Esta camada foi desenvolvida para requisição com os objetos da camada de modelo e retornar os dados para a camada de visualização.

Para conectar-se ao banco de dados antes de executar o programa deverá ser configurado o arquivo connection.inf. Este arquivo contém informações para a conexão com o banco de dados incluindo o usuário e a senha (Quadro 12).



Quadro 12 – Configuração para iniciar a aplicação

Na primeira linha estão as informações para conexão com o banco de dados. Na segunda linha o usuário e na terceira a senha. Com estas informações o programa tcc.jar poderá conectar no banco de dados e verificar se existe um repositório instalado, caso repositório não esteja criado no banco de dados, serão criados as tabelas para manter os códigos fontes que serão criados e também o *backup* dos fontes que serão liberados.

3.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para demonstrar o funcionamento do programa, são apresentadas a seguir algumas funções de utilização do programa, demonstrando um caso real de utilização.

Na figura 10 apresenta-se um exemplo de dois desenvolvedores utilizando o sistema para a codificação dos objetos compartilháveis. Neste caso é demonstrada a compilação normal, trata-se de uma compilação sem validações dos objetos dependentes ou tratamento de retorno do objeto anterior, porém erros de compilação do objeto são demonstrados para o desenvolvedor.

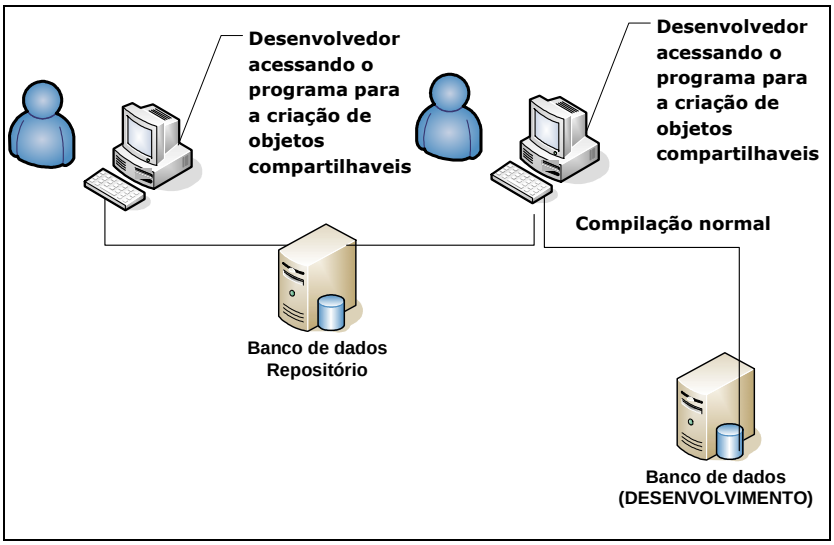


Figura 10 – Esquema de funcionamento “compilação normal”

A seguir na figura 11, encontra-se outro exemplo de procedimento de liberação via agendamento, um agendamento pode ser feito por qualquer usuário do sistema, para isto bastando especificar a senha de acesso no banco de dados de destino. A diferença é que o objeto não será somente compilado, também serão analisadas as dependências dos objetos e em caso de insucesso na liberação, o objeto que foi liberado será retirado do banco de dados, a situação anterior retornará a base tornando o ambiente válido novamente.

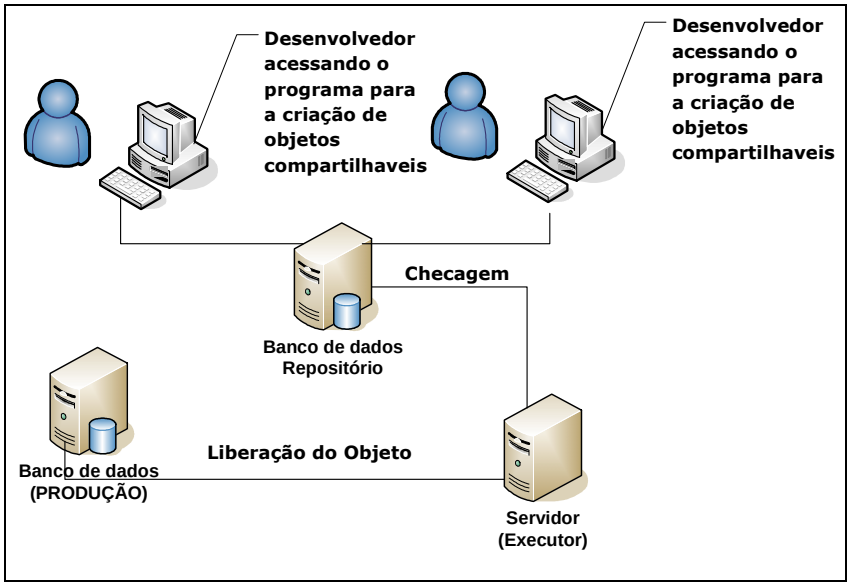


Figura 11 – Esquema de funcionamento “agendamento”

Uma vez configurado o arquivo connection.inf e seguido os passos para criar o usuário no banco de dados, ao iniciar a aplicação tcc.jar a primeira tela que é apresentada é a tela de logon do sistema.

Na tela de *login* (figura 12), é apresentado o campo usuário e o campo senha, ao preencher estes campos e pressionar o botão “OK”, o programa validará os dados digitados e realizará o *logon* do usuário.

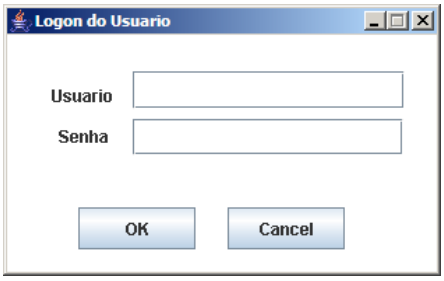


Figura 12 - Tela *login*

Após a tela de *login* será apresentada a tela principal (figura 13). Esta é a tela que o desenvolvedor fará a criação dos novos códigos fontes e ou manutenção de fontes existentes. O usuário possui o menu para trabalhar com o código fonte conforme necessário.

Nesta tela é possível abrir um código fonte que não esteja sendo editado por outro usuário. Caso o usuário tente acessar um código fonte já alocado aparecerá uma mensagem informando da utilização.

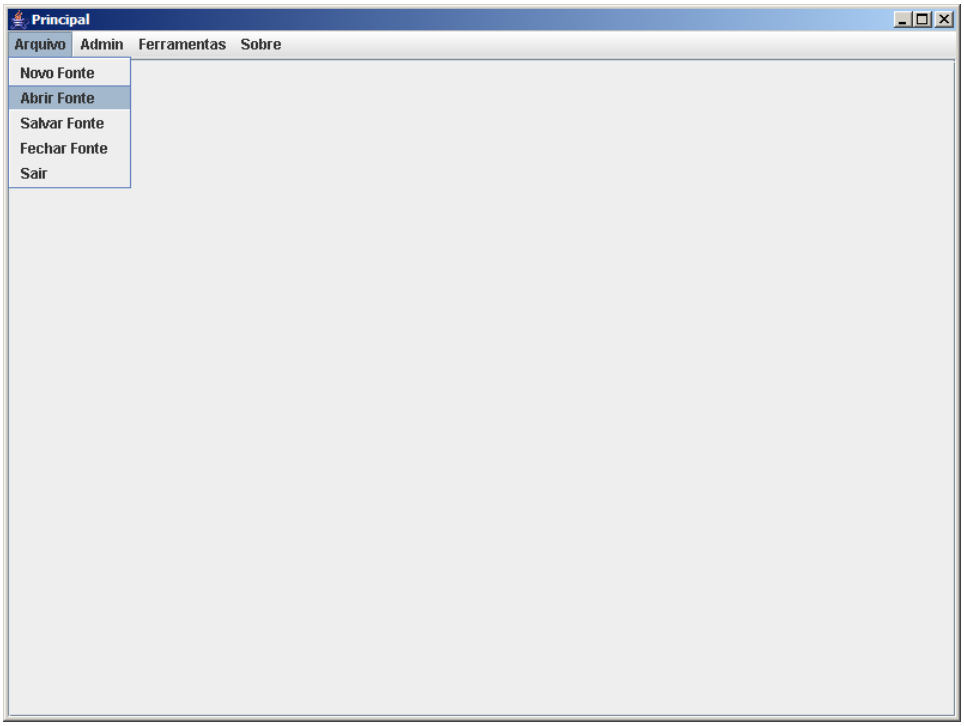


Figura 13 - Tela principal “menu arquivo”

Com esta tela o usuário tem acesso a outros menus. O menu ferramentas mostra a opção de agendamento de entrega dos códigos fontes. Também há a opção de compilação para o código fonte que está sendo editado e uma opção para verificar os agendamentos como

também remover ou retornar um agendamento já realizado. O menu admin, sendo este somente acessado pelo usuário administrador, contém as opções de manter o usuário, gerenciar códigos fontes locados por outro usuários e também a manutenção de fonte de dados, que é onde estão as configurações de conexão para a liberação e compilação dos dados.

Para requisitar a compilação de um objeto, faz-se necessário abrir um código fonte com o “menu de arquivo”. Após aberto é disponibilizada pelo programa a opção de compilação do “menu de ferramentas”, acessando esta opção aparecerá uma janela para informar os dados de conexão (Figura 14).

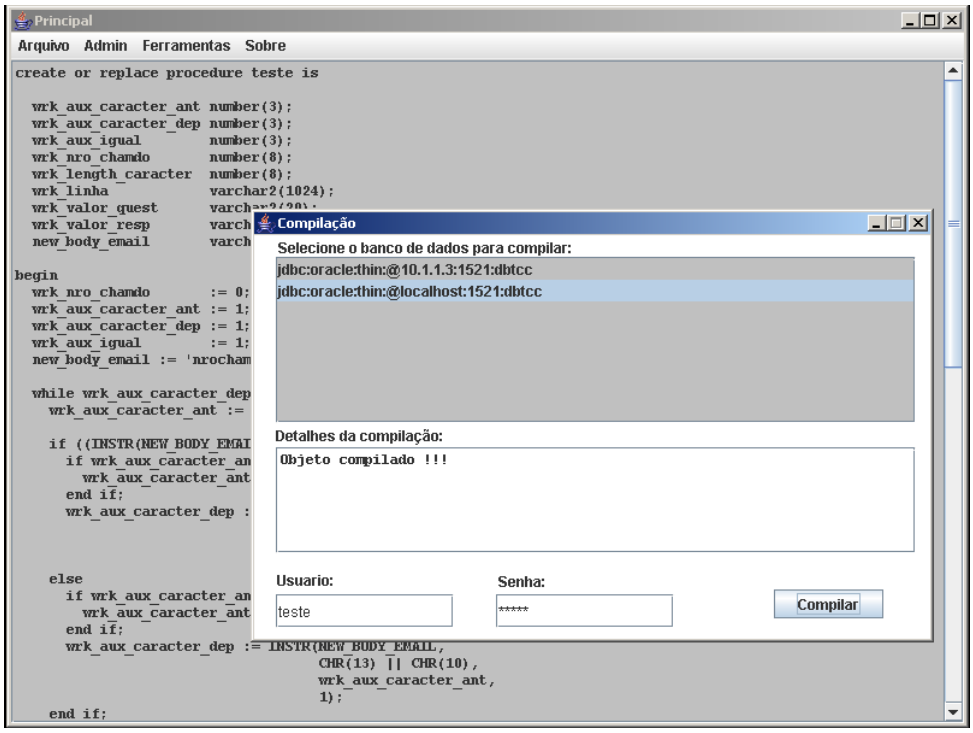


Figura 14 - Tela de compilação

Pressionando o botão de compilação o código fonte será compilado pelo banco de dados. Caso haja problemas com sua compilação os dados serão informados na tela de compilação no campo detalhes da compilação.

Acessando o “menu de ferramentas”, opção de agendamento, o usuário estará acessando a tela para o agendamento das entregas dos códigos fontes (figura 15). Nesta tela deverá ser adicionado o código fonte do usuário clicando em adiciona, o nome do usuário,

senha, banco de dados que serão liberados os objetos, data e hora da liberação, tempo de aguardo para liberar o objeto e com estes dados ao pressionar o botão agendar o agendamento será realizado.

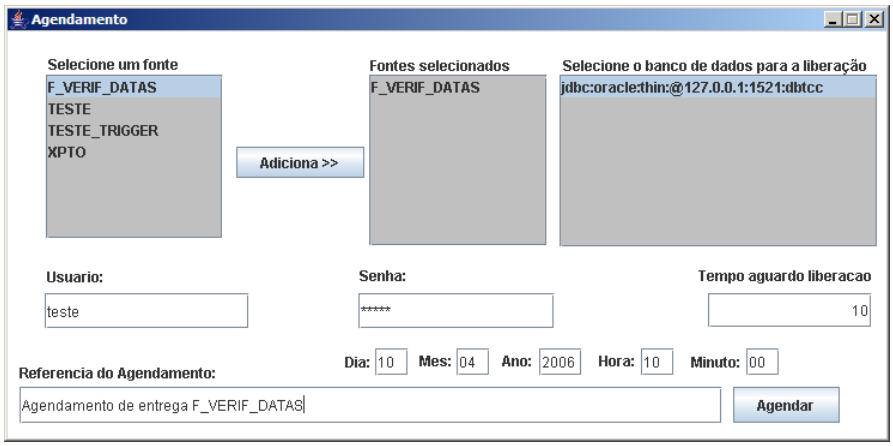


Figura 15 - Tela de agendamento

Também no “menu de ferramentas” é disponibilizada uma opção chamada execuções que é onde se encontram as informações das liberações (figura 16). Para verificar o estado de uma execução clica-se sobre o nome do agendamento. Os dados aparecerão no quadro de informações da liberação.

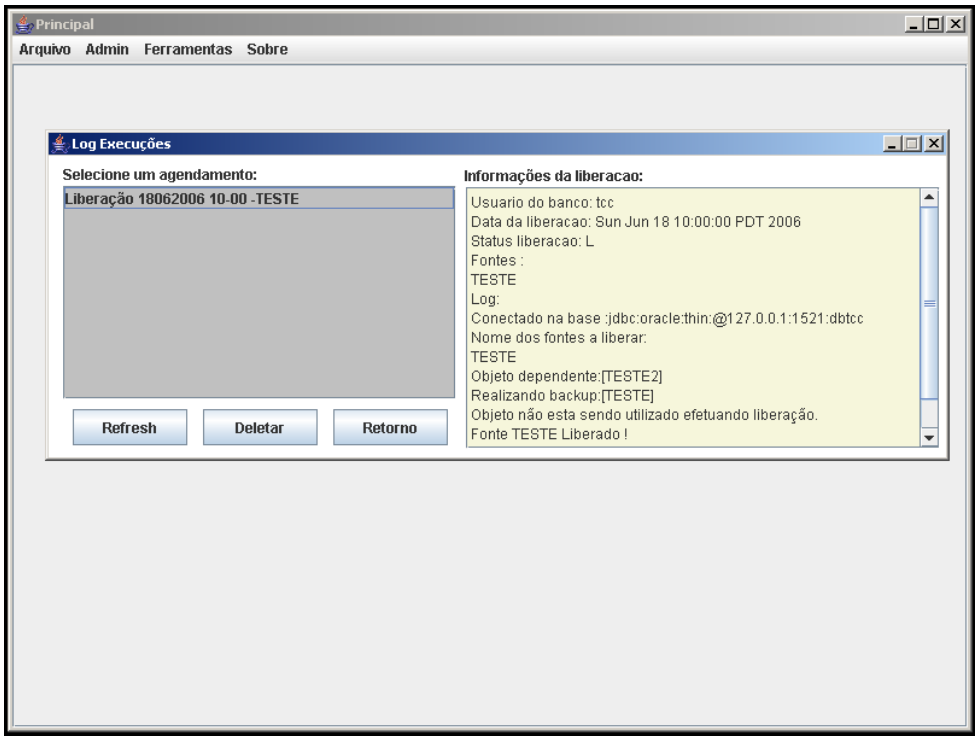


Figura 16 - Tela de log das execuções

Neste exemplo a execução está marcada para efetuar com o usuário de banco “tcc” na data definida. O *status* da liberação é denominado L (Liberada), o objeto TESTE especificado nas informações de liberação foi compilado na base de dados sem erro, tornando desnecessário o retorno da liberação. Também na (figura 16) foi encontrado um objeto dependente chamado TESTE2, no término da liberação este objeto dependente será recompilado. Caso o objeto dependente não consiga ser compilado o objeto liberado será excluído e a situação que estava anteriormente na base de dados será retornada, as dependências serão recompiladas tornando-se válidas novamente.

Os dados referentes à liberação serão inseridos abaixo destas informações listadas (figura 17), a liberação somente será realizada quando o programa tccexecutor.jar for executado e a data previamente definida nesta liberação seja alcançada.

O programa tccexecutor.jar (figura 17) é um modulo separado do programa tcc.jar. No executor é feita a liberação dos objetos compartilháveis. O executor também se utiliza do controlador para executar as liberações.

Figura 17 - Tela tccExecutor

O executor somente necessita da configuração do arquivo connection.inf tendo toda a sua execução baseada no repositório que está na base de dados Oracle.

3.5 RESULTADOS E DISCUSSÃO

Antes do desenvolvimento, foi desenhado o diagrama de classes no *Enterprise Architect* e toda a camada de modelo foi definida desta forma.

Ao iniciar o desenvolvimento do programa, foram avaliadas as opções de ambientes

disponíveis, optando-se pelo Eclipse juntamente com o editor visual. A escolha deu-se pelo ambiente apresentar uma forma agradável para trabalhar com a criação das telas, pois, disponibilizam o código por completo para qualquer tipo de manutenção, assim o código fica com os métodos nomeados conforme é definido, ao contrário de outros softwares que deixam informações estáticas no código e que não podem ser alteradas.

O padrão MVC embora não esteja previsto na proposta, serviu perfeitamente como base para o desenvolvimento do programa. As classes *Singleton* possibilitaram uma forma sutil de controlar a conexão e o tráfego das informações entre as camadas, fazendo com que todo o programa fosse desenvolvido com esta idéia.

A maior preocupação desde o desenvolvimento do programa foi à entrega dos objetos. Toda a ação da entrega e acesso do usuário foram mapeadas para que não ocorressem falhas na entrega e nem com seus objetos dependentes, tornando assim a aplicação confiável.

Uma limitação encontrada no software depois do desenvolvimento mostrou ser o tamanho do código fonte dos objetos. O espaço contido no repositório para o armazenamento não suporta objetos muito grandes.

4 CONCLUSÕES

Este trabalho mostrou-se capaz de exercer a função de liberação de objetos compartilháveis e controle de código fonte, tornando assim o agendamento da liberação possível e também uma forma de manter um histórico de informações pertinentes às liberações.

O uso da ferramenta *Enterprise Architect* foi essencial para a especificação, bem como o uso da JDE Eclipse, através do *pluggin* Editor Visual, que tornou o modo de visualização mais fácil de ser desenvolvido.

Cabe salientar também o uso do padrão MVC, o qual se mostrou em conformidade com as novas tendências no processo de desenvolvimento de software.

Com o trabalho desenvolvido e devidamente testado, não há mais necessidade de agendamento de um DBA para efetuar uma liberação de objetos no banco de dados. Esta contribuição foi de grande valia para as pessoas responsáveis por ambientes de alta utilização e que necessitam realizar manutenções com o objetivo de manter todo o ambiente válido após uma alteração de objeto.

Os objetivos previstos para este trabalho foram atingidos, visto que a ferramenta já está sendo utilizada em banco de dados de produção na empresa em que atuo.

4.1 EXTENSÕES

Nesta seção são apresentadas sugestões e ou alterações para este trabalho, conforme segue:

a) adquirir códigos fontes do banco de dados, importando os códigos no repositório já existente;

b) um identador para os códigos fontes que estão sendo criados, com base na BNF do Pl/Sql, fazendo com que os códigos retornados do banco de dados já sejam identados.

REFERÊNCIAS BIBLIOGRÁFICAS

AULT, Michael. **ORACLE8**: black book. Scottsdale: Coriolis Group Books, 1998.

BUMERANG. **Pesquisa de mercado IPM:** uso da infra-estrutura em TI nas grandes empresas do Brasil. [S.l.], 2005. Disponível em: <http://www.bumeran.com.br/aplicantes/contenidos/zonas/a_articulos.ngmf?IDZONA=7&IDSUBZONA=2&IDART=71120&ZH=0 >. Acesso em: 18 set. 2005.

DAMIANO, Gary. **Oracle**: a beginner's guide. Berkeley: Osborne, 1995.

DATE, C. J. **Introdução a sistemas de bancos de dados**. Rio de Janeiro: Campus, 1991.

GAMMA, Erich. **Padrões de projeto**: soluções reutilizáveis de software orientado a objetos.Porto Alegre: Bookman, 2000, 364p.

LUCHTENBERG, R. **Ferramenta web de monitoração, administração e extração de informações de um SGBD Oracle**. 2002. 56 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

ORACLE. **Oracle9i database summary**. [S.l.], 2002. Disponível em: <http://otn.oracle.com/products/oracle9i/pdf/Oracle9i_Database_summary.pdf>. Acesso em: 18 set. 2005.

APÊNDICE A – Código que realiza a liberação dos objetos.

```
public void efetuaLiberacao(Agendamento p_agen) throws SQLException,
ClassNotFoundException, IOException{

      m_FonteDados =
m_FonteDados.retornaFonteDados(p_agen.getCodFonteDados());
      String logLiberacao = "";
      String[] dependentes = null;
      String[] nome_dependentes = null;
      String[] owner_dependentes = null;
      boolean requerRetorno = true;
      int qtd_dependentes = 0;
      boolean usuariosAcessando= false;
      int erros=0;



      try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con =
DriverManager.getConnection(m_FonteDados.getDsConexao(),
p_agen.getUsuario(), p_agen.getSenha());
            con.setAutoCommit(false);


            logLiberacao += "Conectado na base :"+
m_FonteDados.getDsConexao() +"\n";
            setlog(logLiberacao, p_agen);

            logLiberacao += "Nome dos fontes a liberar:\n";
            setlog(logLiberacao, p_agen);
            int Fontes[] = new int[p_agen.getFontes().length];
            PreparedStatement pst = null;
            ResultSet rs = null;
            Fontes = p_agen.getFontes();

            for (int i = 0; i < Fontes.length; i++) {

                  m_Fontes =
m_Fontes.retornaFonte(m_Fontes.getNmFonte(Fontes[i]));
                  logLiberacao += m_Fontes.getNmFonte()+"\n";
                  setlog(logLiberacao, p_agen);

                  // Verifica as dependencias
                  pst = con.prepareStatement("select " +
                              "count(*) as QTD_DEPENDENTES " +
                              "from DBA_DEPENDENCIES " +
                              "where " +
                              "REFERENCED_NAME = UPPER(?) " +
                              "and " +
                              "REFERENCED_OWNER not in
('CTXSYS','DMSYS','EXFSYS','OLAPSYS','PUBLIC','SYSTEM','XDB') " +
                              "and " +
                              "REFERENCED_TYPE not in ('SYNONYM')");

                  pst.setString(1,m_Fontes.getNmFonte());
                  rs = pst.executeQuery();
```

```
                if (rs.next()){
                      qtd_dependentes += rs.getInt("QTD_DEPENDENTES");

                }

                // Verifica se existem erros no objeto a ser liberado,
caso haja nao efetura liberacao.
                pst = con.prepareStatement("select STATUS from
user_objects where object_name = ?");
                pst.setString(1,m_Fontes.getNmFonte());

                rs =pst.executeQuery();
                if (rs.next()){
                      if
(rs.getString("STATUS").equalsIgnoreCase("INVALID")){
                            requerRetorno = false;
                            erros++;
                      }
                }
           }

           // Caso não hajam erros efetua liberacao
           if (erros == 0){
                dependentes = new String[qtd_dependentes +
p_agen.getFontes().length];
                nome_dependentes = new String[qtd_dependentes +
p_agen.getFontes().length];
                owner_dependentes = new String[qtd_dependentes +
p_agen.getFontes().length];

                // Popula o array de dependencias
                for (int i = 0; i < Fontes.length; i++) {

                      m_Fontes =
m_Fontes.retornaFonte(m_Fontes.getNmFonte(Fontes[i]));
                      pst = con.prepareStatement("select " +
                                  "OWNER    as OWNER " +
                                  ",TYPE    as OBJECT_TYPE" +
                                  ",NAME    as OBJECT_NAME " +
                                  "from DBA_DEPENDENCIES " +
                                  "where " +
                                  "REFERENCED_NAME = UPPER(?) " +
                                  "and " +
                                  "REFERENCED_OWNER not in
('CTXSYS','DMSYS','EXFSYS','OLAPSYS','PUBLIC','SYSTEM','XDB') " +
                                  "and " +
                                  "REFERENCED_TYPE not in ('SYNONYM')");

                      pst.setString(1,m_Fontes.getNmFonte());
                      rs = pst.executeQuery();

                      // Insere no array de dependencias o comando para a
recompilação.
                      int wrk_int = 0;
                      while(rs.next()){
                            logLiberacao += "Objeto dependente:[" +
rs.getString("OBJECT_NAME")+ "]"+"\n";
                            setlog(logLiberacao, p_agen);
                            nome_dependentes[wrk_int] =
rs.getString("OBJECT_NAME");
```

```
                              owner_dependentes[wrk_int] =
rs.getString("OWNER");

                              if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("VIEW")){
                                    dependentes[wrk_int] = "alter view "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";

                              }
                              if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("PACKAGE")){
                                    dependentes[wrk_int] = "alter package "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";
                              }
                              if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("PACKAGE BODY")){
                                    dependentes[wrk_int] = "alter package "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile
body";
                              }
                              if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("PROCEDURE")){
                                    dependentes[wrk_int] = "alter procedure
"+ rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";
                              }
                              if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("FUNCTION")){
                                    dependentes[wrk_int] = "alter function
"+ rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";
                              }
                              if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("TRIGGER")){
                                    dependentes[wrk_int] = "alter trigger "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";
                              }
                              wrk_int++;
                        }
                  }


                  // Realiza o backup dos objetos
                  for (int i = 0; i < p_agen.getFontes().length; i++) {
                        m_Fontes =
m_Fontes.retornaFonte(m_Fontes.getNmFonte(p_agen.getFontes()[i]));

                        // Em caso de package, package body, function,
procedure backup da user_source
                        pst = con.prepareStatement("select TEXT, NAME, TYPE
" +
                                    "from user_source " +
                                    "where " +
                                    "name = ?");

                        pst.setString(1, m_Fontes.getNmFonte());
                        rs = pst.executeQuery();
                        if (rs.next()){
                              logLiberacao += "Realizando backup:[" +
rs.getString("NAME")+ "]"+"\n";
                              setlog(logLiberacao, p_agen);
                              insereBackupFontes(p_agen.getCodAgendamento(),
rs.getString("NAME"), "create or replace " +  rs.getString("TEXT"));
```

```
                        }

                       // Em caso de view realiza backup da user_views
                        pst = con.prepareStatement("select TEXT from
user_views where VIEW_NAME = ?");
                        pst.setString(1, m_Fontes.getNmFonte());
                        rs = pst.executeQuery();

                        while (rs.next()){
                              logLiberacao += "Realizando backup:[" +
m_Fontes.getNmFonte()+ "]"+"\n";
                              setlog(logLiberacao, p_agen);

                              InputStream ascii_data = rs.getAsciiStream
(1);
                              int c;
                              ByteArrayOutputStream os =new
ByteArrayOutputStream ();
                              String texto = "";

                              while ((c = ascii_data.read ()) != -1)
                                    os.write(c);
                                //texto += c;

                              texto = os.toString();

                              int codigoagendamento =
p_agen.getCodAgendamento();
                              String view_name = m_Fontes.getNmFonte();

                              insereBackupFontes(codigoagendamento,
                                          view_name,
                                          "create or replace view " +
                                          view_name +
                                          " as " +
                                          texto
                                          );
                        }

                        // Em caso de trigger realiza backup da
user_triggers
                        pst = con.prepareStatement("select TRIGGER_BODY from
user_triggers where TRIGGER_NAME = ?");
                        pst.setString(1, m_Fontes.getNmFonte());

                        rs = pst.executeQuery();

                        while (rs.next()){

                              logLiberacao += "Realizando backup:[" +
m_Fontes.getNmFonte()+ "]"+"\n";
                              setlog(logLiberacao, p_agen);
                              ByteArrayOutputStream os =new
ByteArrayOutputStream ();
                              InputStream ascii_data = rs.getAsciiStream
(1);
                              int c;
                              String texto = "";

                              while ((c = ascii_data.read ()) != -1)
                                os.write(c);
```

```
                              texto = os.toString();

                              insereBackupFontes(p_agen.getCodAgendamento(),
                                          m_Fontes.getNmFonte(),
                                          "create or replace trigger "
                                          + texto);
                        }
                  }

                  // Verifica os acessos nos objetos da liberação
                  int qtdAcessos = 1; // caso o usuario nao tenha colocado
o periodo de aguardo.
                  int qtdChecagem = p_agen.getAguardaLiberacao();

                  while (qtdChecagem > 0){
                        qtdAcessos = 0;
                        for (int i = 0; i < p_agen.getFontes().length; i++)
{
                              m_Fontes =
m_Fontes.retornaFonte(m_Fontes.getNmFonte(p_agen.getFontes()[i]));

                              pst = con.prepareStatement("select count(*) as
QTD_ACESSOS " +
                                          "from v$access a, v$session s,
v$process p " +
                                          "where object like upper(?) " +
                                          "and a.sid = s.sid " +
                                          "and p.addr = s.paddr");

                              pst.setString(1, m_Fontes.getNmFonte());
                              rs = pst.executeQuery();
                              if (rs.next()){
                                    qtdAcessos += rs.getInt("QTD_ACESSOS");
                              }
                        }
                        if (qtdAcessos > 0){
                              logLiberacao += "Objeto utilizado,
aguardando!"+"\n";
                              setlog(logLiberacao, p_agen);
                              qtdChecagem--;
                              try {
                                    Thread.sleep(1000);
                              } catch (InterruptedException erro) {
                                    System.out.println("Erro - " +
erro.getMessage());
                              }
                        }else{
                              qtdChecagem = qtdAcessos;
                        }
                  }

                  if (qtdAcessos == 0){
                        boolean necessitaReenviar = false;
                        usuariosAcessando = false;

                        logLiberacao += "Objeto não esta sendo utilizado
efetuando liberação."+"\n";
                        setlog(logLiberacao, p_agen);
                        // Realiza a liberação dos objetos
```

```
                        for (int i = 0; i < p_agen.getFontes().length; i++)
{
                              m_Fontes =
m_Fontes.retornaFonte(m_Fontes.getNmFonte(p_agen.getFontes()[i]));

                              // Liberando objetos
                                    try{
                                          logLiberacao +=
m_Fontes.liberaFonteBase(m_FonteDados.getDsConexao(), p_agen.getUsuario(),
p_agen.getSenha());
                                          setlog(logLiberacao, p_agen);
                                          pst = con.prepareStatement("select
OBJECT_NAME " +
                                                      "from user_objects " +
                                                      "where status = ? " +
                                                      "and
upper(object_name) = upper(?) ");

                                          pst.setString(1, "INVALID");
                                          pst.setString(2,
m_Fontes.getNmFonte());
                                          rs = pst.executeQuery();
                                          if (rs.next()){
                                                necessitaReenviar = true;
                                          }
                                    }catch(Exception erro){
                                          logLiberacao += "Erro ao liberar
objeto :"+m_Fontes.getNmFonte()+", será recompilando posteriormente.
\n"+erro.getMessage()+"\n";
                                          setlog(logLiberacao, p_agen);
                                          necessitaReenviar = true;
                                    }

                              logLiberacao += "Fonte " +
m_Fontes.getNmFonte() + " Liberado !"+"\n";
                              setlog(logLiberacao, p_agen);
                        }


                        if (necessitaReenviar){
                              logLiberacao += "Reenviando objeto(s) pois
ficaram invalidos: \n";
                              setlog(logLiberacao, p_agen);
                              // reenviar objetos liberados com erro.
                              for (int i = 0; i < p_agen.getFontes().length;
i++) {
                                    m_Fontes =
m_Fontes.retornaFonte(m_Fontes.getNmFonte(p_agen.getFontes()[i]));

                                   // Liberando objetos
                                    try{
                                          logLiberacao =
m_Fontes.liberaFonteBase(m_FonteDados.getDsConexao(), p_agen.getUsuario(),
p_agen.getSenha());
                                          setlog(logLiberacao,
p_agen);
                                          pst =
con.prepareStatement("select OBJECT_NAME " +
                                                      "from
user_objects " +
```

```
                                                "where status =
? " +
                                                "and
upper(object_name) = upper(?) ");

                                          pst.setString(1, "INVALID");
                                          pst.setString(2,
m_Fontes.getNmFonte());
                                          rs = pst.executeQuery();
                                          if (rs.next()){
                                                erros++;
                                          }else{
                                                logLiberacao += "Fonte
" + m_Fontes.getNmFonte() + ".. OK !"+"\n";
                                                setlog(logLiberacao,
p_agen);
                                          }
                                    }catch(Exception erro){
                                          logLiberacao += "Erro ao
reliberar objeto :"+m_Fontes.getNmFonte()+"\n"+erro.getMessage()+"\n";
                                          setlog(logLiberacao,
p_agen);
                                          erros++;
                                    }
                              }
                        }

                        // Recompila objetos dependentes.
                        for (int i = 0; i < dependentes.length; i++) {
                              if (!(dependentes[i] == null)){
                                    pst =
con.prepareStatement(dependentes[i]);
                                    try{
                                          pst.execute();
                                    }catch(Exception erro){
                                          erro.printStackTrace();
                                          erros++;
                                    }
                                    pst = con.prepareStatement("select
STATUS from DBA_OBJECTS where owner = ? and object_name = ?");
                                    pst.setString(1, owner_dependentes[i]);
                                    pst.setString(2, nome_dependentes[i]);
                                    rs = pst.executeQuery();
                                    // Caso o objeto esteja invalido, conta
como erro.
                                    if ((rs.next()) &&
(rs.getString("STATUS").equals("INVALID"))){
                                          erros++;

                                    }
                              }
                        }
                  }else{
                        logLiberacao += "Erro: Usuários acessando
objetos.\n";
                        setlog(logLiberacao, p_agen);
                        erros++;

                  }
            }
```

```
            if (erros == 0){
                  p_agen.setLiberadoBase();
                  logLiberacao += "Objeto (s) entregue (s) com sucesso
!!!"+"\n";
                  setlog(logLiberacao, p_agen);
            }else{
                  logLiberacao += "Liberação abortada !!!"+"\n";
                  setlog(logLiberacao, p_agen);
                  if (usuariosAcessando || !requerRetorno){
                    logLiberacao += "Liberação abortada sem necessidade de
retorno !!!"+"\n";
                    setlog(logLiberacao, p_agen);
                    p_agen.setErroLiberadoBase();
                  }else{
                    efetuaRetorno(p_agen);
                  }
            }

            pst.close();
            rs.close();
            con.close();

      }catch(Exception erro){
            erro.printStackTrace();
            logLiberacao += "Erro na liberação:"+
m_FonteDados.getDsConexao() +" "+erro.getMessage() + "\n";
            setlog(logLiberacao, p_agen);
            if (requerRetorno){
                  efetuaRetorno(p_agen);
            }
            p_agen.setErroLiberadoBase();

      }
}
```

APÊNDICE B – Código para o retorno dos objetos em caso de erro na liberação.

```
public void efetuaRetorno(Agendamento p_agen) throws SQLException,
ClassNotFoundException, IOException{
	String logLiberacao = "";
	String[] dependentes = null;
	int qtd_dependentes = 0;
	Fontes fonte = new Fontes();
	m_FonteDados =
m_FonteDados.retornaFonteDados(p_agen.getCodFonteDados());

	try{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connect = ConnectionManager.getInstance().getCon();
		Connection con =
DriverManager.getConnection(m_FonteDados.getDsConexao(),
p_agen.getUsuario(), p_agen.getSenha());
		con.setAutoCommit(false);

		logLiberacao += getLog(p_agen);
		logLiberacao += "Conectado na base para retornar objetos:"+
m_FonteDados.getDsConexao() +"\n";
		setlog(logLiberacao, p_agen);

		int Fontes[] = new int[p_agen.getFontes().length];
		PreparedStatement pst;
		PreparedStatement wrk_pst;
		ResultSet rs;
		Fontes = p_agen.getFontes();

		// realiza retorno do backup dos objetos.
		pst = connect.prepareStatement("select " +
				"CD_AGENDAMENTO , NM_FONTE, SRC " +
				"from BACKUP " +
				"where " +
				"CD_AGENDAMENTO = ?");

		pst.setInt(1, p_agen.getCodAgendamento());
		rs = pst.executeQuery();

		while (rs.next()){
			wrk_pst =
con.prepareStatement(rs.getString("SRC").replaceAll("\r\n","
").replaceAll("\t"," "));
			wrk_pst.execute();
		}

		for (int i = 0; i < Fontes.length; i++) {

			fonte = fonte.retornaFonte(fonte.getNmFonte(Fontes[i]));
			logLiberacao += fonte.getNmFonte()+"\n";
			setlog(logLiberacao, p_agen);

			// Verifica as dependencias
			pst = con.prepareStatement("select " +
					"count(*) as QTD_DEPENDENTES " +
					"from DBA_DEPENDENCIES " +
					"where " +
```

```
                              "REFERENCED_NAME = UPPER(?) " +
                              "and " +
                              "REFERENCED_OWNER not in
('CTXSYS','DMSYS','EXFSYS','OLAPSYS','PUBLIC','SYSTEM','XDB') " +
                              "and " +
                              "REFERENCED_TYPE not in ('SYNONYM')");

                  pst.setString(1,fonte.getNmFonte());
                  rs = pst.executeQuery();
                  if (rs.next()){
                        qtd_dependentes += rs.getInt("QTD_DEPENDENTES");

                  }

            }

            dependentes = new String[qtd_dependentes];

            // Popula o array de dependencias
            for (int i = 0; i < Fontes.length; i++) {

                  fonte = fonte.retornaFonte(fonte.getNmFonte(Fontes[i]));
                  pst = con.prepareStatement("select " +
                              "OWNER    as OWNER " +
                              ",TYPE    as OBJECT_TYPE" +
                              ",NAME    as OBJECT_NAME " +
                              "from DBA_DEPENDENCIES " +
                              "where " +
                              "REFERENCED_NAME = UPPER(?) " +
                              "and " +
                              "REFERENCED_OWNER not in
('CTXSYS','DMSYS','EXFSYS','OLAPSYS','PUBLIC','SYSTEM','XDB') " +
                              "and " +
                              "REFERENCED_TYPE not in ('SYNONYM')");

                  pst.setString(1,fonte.getNmFonte());
                  rs = pst.executeQuery();

                  // Insere no array de dependencias o comando para a
recompilação.
                  int wrk_int = 0;
                  while(rs.next()){
                        logLiberacao += "Objeto dependente:[" +
rs.getString("OBJECT_NAME")+ "]"+"\n";
                        setlog(logLiberacao, p_agen);
                        if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("VIEW")){
                              dependentes[wrk_int] = "alter view "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";

                        }
                        if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("PACKAGE")){
                              dependentes[wrk_int] = "alter package "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";
                        }
                        if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("PACKAGE BODY")){
                              dependentes[wrk_int] = "alter package "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile
body";
```

```
                        }
                        if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("PROCEDURE")){
                              dependentes[wrk_int] = "alter procedure "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";
                        }
                        if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("FUNCTION")){
                              dependentes[wrk_int] = "alter function "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";
                        }
                        if
(rs.getString("OBJECT_TYPE").equalsIgnoreCase("TRIGGER")){
                              dependentes[wrk_int] = "alter trigger "+
rs.getString("OWNER") + "." + rs.getString("OBJECT_NAME") + " compile";
                        }
                        wrk_int++;
                  }
            }

            logLiberacao += "Objeto (s) retornado (s) com sucesso
!!!"+"\n";
            setlog(logLiberacao, p_agen);
            p_agen.setErroLiberadoBase();

            pst.close();
            rs.close();
            con.close();

      }catch(Exception erro){
            erro.printStackTrace();
            logLiberacao += "Erro na liberação:"+
m_FonteDados.getDsConexao() +" "+erro.getMessage() + "\n";
            setlog(logLiberacao, p_agen);
            p_agen.setErroLiberadoBase();
      }

}
```