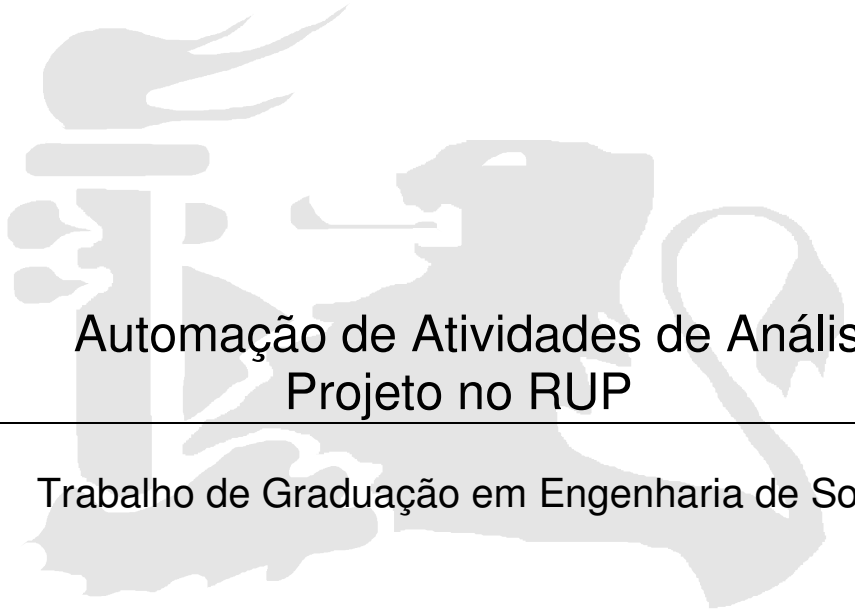
UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA



Automação de Atividades de Análise e Projeto no RUP

Trabalho de Graduação em Engenharia de Software

Aluno: Márcio Eduardo Leal Bezerra (melb@cin.ufpe.br).
Orientador: Augusto César Alves Sampaio (acas@cin.ufpe.br).

Recife
2007.2

Assinaturas

Este Trabalho de Graduação é resultado dos esforços do aluno Márcio Eduardo Leal Bezerra, sob a orientação do professor Augusto César Alves Sampaio, sob o título de “Automação de Atividades de Análise e Projeto no RUP”. Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

_____

Augusto César Alves Sampaio (orientador)

_____

Márcio Eduardo Leal Bezerra

Resumo

O Rational Unified Process® (RUP) é um processo de engenharia de software bastante difundido na indústria por utilizar-se de boas práticas de desenvolvimento como a modelagem através da conceituada Unified Modeling Language® (UML). Aliado a isso, várias ferramentas fornecem suporte ao processo, principalmente à disciplina de análise e projeto. Apesar de ser a modelagem do sistema suportada por ferramentas e facilitada por UML, muitos dos passos ainda são cansativos e repetitivos, os quais são realizados manualmente. Sendo assim, esse trabalho identifica passos passíveis de automação de atividades da disciplina de análise e projeto do RUP e visa à validação das regras de automação através da construção de um *plugin* denominado RADAS para a ferramenta Rational Rose®. As regras de automação são referentes às atividades Analisar Caso de Uso nos passos "Descrever responsabilidades para as classes identificadas" e "Descrever atributos e associações para as classes identificadas", Projetar Arquitetura no passo "Identificar e mapear elementos de projeto" e Projetar Caso de Uso.

Palavras-chave:

RUP, UML, CASE, Processo, Atividade, Automação.

Agradecimentos

Em primeiro lugar, agradeço a Deus por me proporcionar o cumprimento de mais uma importante etapa da minha vida.
À minha família por ter me garantido a minha formação educacional e moral, sendo a principal fonte dos meus princípios e valores, e também, de minha felicidade.
Aos meus amigos, pelas palavras de apoio, pelos momentos de descontração e união.
À minha namorada, Carolina, por sua compreensão, carinho e amor, seus incentivos e contribuições diretas quais me deram motivação para a realização das minhas atividades.
Ao meu orientador Professor Augusto Sampaio por suas valiosas orientações e ajuda no desenvolvimento deste trabalho.
Ao meu colega de curso Joabe Jesus, que me ajudou a formar a idéias importantes para este trabalho.
Aos demais colegas, aos professores e funcionários do Centro de Informática da UFPE, a família Consist e a todas as outras pessoas que de alguma forma contribuíram, ao longo de minha vida, para que eu alcançasse mais este objetivo.

Muito obrigado.

Sumário

1 Introdução

O software é uma parte indispensável do nosso mundo moderno [2], sendo a economia mundial cada vez mais dependente dele. Os tipos de software tecnologicamente possíveis e a demanda da sociedade estão aumentando em tamanho, complexidade, distribuição e importância [6]. Tais variáveis pressionam a indústria para novos patamares de qualidade e produtividade conjugados com rápido desenvolvimento e distribuição, gerando assim novos problemas organizacionais e técnicos.

Para auxiliar na resolução desses problemas surgem processos (ou metodologias) de engenharia de software como o Rational Unified Process® (RUP).

O RUP tem como objetivo assegurar a produção de software com qualidade, satisfazendo as necessidades de seus usuários finais dentro do prazo definido. Sua vasta adoção por diferentes setores da indústria se deve ao fato, dentre outros, de aplicar boas práticas de desenvolvimento de software modernas.

Dentre as boas práticas fundamentadas pela disciplina de análise e projeto do RUP estão o desenvolvimento de uma arquitetura baseada em componentes e o uso de modelagem gráfica do sistema através de uma linguagem padrão e bem difundida chamada de Unified Modeling Language® (UML).

A proposta da disciplina de análise e projeto é traduzir os requisitos do sistema em uma especificação que descreva como implementar o sistema [6], visando definir uma arquitetura robusta, fácil de compreender, construir e evoluir.

A modelagem através de UML é utilizada para descrever a arquitetura e o comportamento do sistema, evidenciando sua importância durante todo o desenvolvimento, principalmente nas etapas iniciais, garantindo uma arquitetura que suporte o sistema.

Assim como as demais disciplinas do RUP, a disciplina de análise e projeto é suportada por ferramentas CASE (*Computer-aided Software Engineering*), dentre elas o Rational Rose®. Ainda que a disciplina seja suportada por ferramentas de software e os passos de suas atividades sejam bem detalhados no processo, alguns desses passos ainda são repetitivos e cansativos para analistas e projetistas. Sendo assim, seria necessário um nível maior de automação por parte das ferramentas.

Partindo da afirmação de que “processos de software são softwares” [8] alguns passos do processo são passíveis de automação, demandando uma revisão do RUP visando à identificação de tais passos.

Durante sua constante evolução, o RUP assimilou processos de outras empresas e trabalhos relacionados. O que vai de encontro à possibilidade de extensão de ferramentas de suporte por terceiros, como o Rational Rose® que disponibiliza uma interface para extensão.

Devido aos fatos expostos, o corrente trabalho revisou a disciplina de análise e projeto do RUP visando identificar passos de atividades passíveis de automação, propondo meios de automatizá-los (regras de automação) em uma ferramenta CASE, além de construir um *plugin* para o Rational Rose®,

denominado de RADAS, que automatiza os passos identificados como forma de validação dos resultados desse trabalho.

As regras de automação são referentes às atividades Analisar Caso de Uso, Projetar Arquitetura e Projetar Caso de Uso, todas da disciplina Análise e Projeto. As regras para a atividade Analisar Caso de Uso inferem relacionamentos e operações das classes envolvidas através de um diagrama de interação. Para a atividade Projetar Arquitetura, as regras refinam o diagrama de pacotes e sugere uma nova forma de mapear classes de análise em classes de projeto. As regras para a atividade Projetar Caso de Uso dizem respeito a como atualizar o modelo de projeto a partir do modelo de análise de forma automática.

No Capítulo 2 apresentamos uma breve introdução ao RUP com ênfase na disciplina de Análise e Projeto, no Capítulo 3 é descrito alguns conceitos e artefatos da UML utilizados nesse trabalho, no Capítulo 4 são analisadas atividades da disciplina de Análise e Projeto e inferida as regras de automação, no Capítulo 5 apresentamos algumas ferramentas avaliadas para a implementação do RADAS, no Capítulo 6 é demonstrado o RADAS com suas funcionalidades referentes às regras de automação e resultados de testes, no Capítulo 7 apresentamos as conclusões e comentários de trabalhos relacionados e futuros.

2 Rational Unified Process

O Rational Unified Process (RUP) é um processo de engenharia de software que fornece uma abordagem disciplinada para sistematizar tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a produção de software de alta qualidade, satisfazendo as necessidades de seus usuários finais dentro do prazo e orçamento previsíveis [6].

O RUP é um produto criado pela Rational Software, adquirida pela IBM, atual mantenedora. Além de material didático do processo a IBM disponibiliza um conjunto de ferramentas proprietárias que suportam o RUP como o RequisitePro para o gerenciamento de requisitos, o TestManager para o gerenciamento de testes, o ClearCase para o gerenciamento de configuração, o Rose para modelgem, entre outros.

Outras empresas e a comunidade *open source* disponibilizam ferramentas que oferecem suporte ao RUP sendo, algumas delas analisadas no presente trabalho.

O RUP é um processo que pode ser adaptável às necessidades de cada empresa, ou seja, instâncias do RUP podem ser estendidas ou reduzidas deixando a organização livre para adequar os componentes do processo ao ambiente da mesma.

A utilização de práticas de desenvolvimento de software consolidadas foi um dos fatores que levou o RUP a ser largamente adotado pela indústria do software. São elas:

- Desenvolvimento iterativo
- Gerenciamento intenso de requisitos
- Uso de arquiteturas baseadas em componentes
- Modelagem gráfica do sistema através de Unified Modeling Language (UML)
- Controle integrado de qualidade e mudanças

Outro ponto importante é que o RUP é *dirigido a casos de uso*. Os casos de uso guiam todo o processo de desenvolvimento onde os mesmos tentam capturar as funcionalidades do sistema.

Como já citado, o RUP pode ser estendido e em sua própria evolução traz consigo diferentes extensões como o RUP para sistemas em tempo real e o RUP para comércio eletrônico. Ambas as extensões não serão abordadas neste trabalho.

No próximo capítulo são descritos alguns aspectos da UML utilizados pelo RUP.

2.1 A Estrutura do Processo

O RUP é definido em duas estruturas: Uma estrutura dinâmica que trata de conceitos de iterações, ciclo de vida do projeto e outros fatores do processo referentes ao acompanhamento e gerenciamento do projeto, e uma estrutura estática, mais fortemente ligada ao tema desse trabalho, pois trata de conceitos-chave como papéis, atividades, artefatos, entre outros que serão necessários para o entendimento da disciplina de análise e projeto.

2.1.1 A Estrutura Estática

Seguem os quatro elementos básicos da estrutura estática do processo:

- Papéis: Quem faz algo.
- Atividades: Como fazer.
- Artefatos: O que fazer.
- Fluxos: Quando fazer.

Um papel designa quais comportamentos e responsabilidades um indivíduo (ou um grupo deles) da equipe de desenvolvimento tem em uma ou mais atividades coerentes, ou seja, que sejam melhores executadas por determindado perfil. Vendo por outro lado, alguns papéis são relacionados a tipos de artefatos.

Vale salientar que um papel não é necessariamente exercido por uma única pessoa da equipe e muito menos durante todo o processo. Pessoas podem acumular papéis, como também trocar de papéis em diferentes partes do projeto.

Uma atividade é uma unidade de trabalho que é desempenhada por um papel específico com entradas e saídas bem definidas. Geralmente as entradas são artefatos e as saídas, que são os obejtivos das atividades, são novos artefatos ou atualizações do mesmo.

Uma atividade pode ser repetida durante o processo, nas fases ou iterações do mesmo, com intuito de criar novos artefatos ou atualizar os mesmos.

Artefatos são os produtos tangíveis do projeto: trata-se daquilo que é produzido ou usado pelo projeto, enquanto o mesmo trabalha para o produto final [2]. Eles podem ser um modelo ou um elemento desse modelo, um documento, código-fonte e seus executáveis.

A impressão dos artefatos não é encorajada pelo RUP, pois o controle sobre tal artefato é perdido, dificultando o acesso da equipe ao mesmo. O que o RUP prega é que os artefatos ou relatórios sejam impressos, caso necessário, apenas para apresentação oportuna e depois, descartados. Sendo assim, os artefatos devem ser mantidos e atualizados pelas ferramentas onde foram criados e gerenciados por um controle de versão.

Tal abordagem vai de acordo com o propósito desse trabalho, pois os artefatos utilizados pela disciplina de análise e projeto são gerados por ferramentas CASE (do inglês, *Computer-Aided Software Engineering*), sendo

as mesmas o veículo para a automatização dos passos do processo como no caso do corrente trabalho onde foi utilizada a Rational Rose.

Os artefatos ainda são divididos e organizados por conjuntos de informação: gerência, requisitos, projeto, implementação e distribuição. Desses, o mais importante para esse trabalho é o conjunto de projeto, pois nele estão organizados os artefatos produzidos na disciplina de análise e projeto, como o modelo de projeto, descrição da arquitetura e o modelo de testes. Os mesmos são descritos no tópico da disciplina de análise e projeto.

Um fluxo é composto pela relação lógica e temporal das atividades, como também evidencia a relação entre os papéis com a finalidade de obter resultados esperados do processo, ou seja, artefatos, atividades e papéis precisam ser relacionados para constituir um processo.

O RUP adota uma organização lógica de um conjunto de atividades em fluxos chamados de disciplinas.

Disciplinas de engenharia:

- Modelagem de negócios
- Requisitos
- Análise e Projeto
- Implementação
- Testes
- Distribuição

Disciplinas de suporte:

- Gerenciamento de Projeto
- Gerenciamento de Configuração e Mudança
- Ambiente

As disciplinas são geralmente representadas em UML por diagramas de atividades que demonstram como se dá o fluxo das atividades na mesma. Cada estado de atividade no diagrama representa um conceito do RUP chamado de detalhes do fluxo. O mesmo é composto por uma ou mais atividades da disciplina em questão.

2.2 A Arquitetura

O RUP foca na concepção e construção de modelos e estes são imprescindíveis para compreender tanto o problema como a solução viabilizada pelo software a ser desenvolvido. Modelos constituem muitos dos artefatos do conjunto de projeto, que será explicado na seção posterior.

Modelos são simples representações da realidade e não constroem ou implementam um sistema. Os modelos apenas ajudam a guiar a equipe do projeto, mostrando o que é o sistema e como funciona, possibilitando reuso de componentes e extensão.

O RUP é um processo centrado em uma arquitetura, ou melhor, em modelos e documentos e em suas diferentes visões. Define-se arquitetura como sendo:

- A organização de um sistema de software [6]

- A seleção de elementos estruturais e as interfaces das quais o sistema é composto, junto com seu comportamento, como especificado na colaboração entre os elementos [6]
- A composição destes modelos em subsistemas progressivamente maiores [6]

Segundo a definição do RUP, a arquitetura carrega muita informação em seus diferentes modelos e estes são de interesse de muitos *stakeholders* como: analista de sistemas, arquitetos, usuários e clientes, gerentes de projeto, empresas terceirizadas, entre outros. Todos possuem diferentes expectativas do projeto.

Sendo assim, nem toda informação, ou nem todos os modelos interessam a determinada parte. Por isso o RUP apresenta 5 diferentes visões da arquitetura:

- A Visão de Casos de Uso contém casos de uso e cenários que abrangem comportamentos significativos em termos de arquitetura, classes ou riscos técnicos. Ela é um subconjunto do modelo de casos de uso [12].
- A Visão Lógica contém as classes de design mais importantes e sua organização em pacotes e subsistemas em camadas. Ela contém algumas realizações de caso de uso. É um subconjunto do modelo de design [12].
- A Visão de Implementação contém uma visão geral do modelo de implementação e sua organização em termos de módulos em pacotes e camadas. A alocação de pacotes e classes (da Visão Lógica) nos pacotes e módulos da Visão de Implementação também é descrita. Ela é um subconjunto do modelo de implementação [12].
- A Visão de Processos contém a descrição das tarefas (processo e threads) envolvidas, suas interações e configurações, e a alocação dos objetos e classes de design em tarefas. Essa visão só precisará ser usada se o sistema tiver um grau significativo de simultaneidade. No RUP, ela é um subconjunto do modelo de design [12].
- A Visão de Distribuição contém a descrição dos vários nós físicos da maior parte das configurações comuns de plataforma e a alocação das tarefas (da Visão de Processos) nos nós físicos. Essa visão só precisará ser usada se o sistema estiver distribuído. Ela é um subconjunto do modelo de implantação [12].

Os artefatos da visão de casos de uso servirão de entrada para as atividades da disciplina de análise e projeto. A visão lógica, de processos e de distribuição (ou implantação) são trabalhadas na disciplina de análise e projeto, estando a visão lógica enfatizada nesse trabalho.

Uma instância do RUP poderá possuir mais visões da arquitetura dependendo da necessidade da organização e do projeto, como por exemplo: visão de segurança, de dados, de interface com o usuário, entre outras.

Em projetos de software menores e simples, algumas visões do RUP geralmente não são utilizadas como as visões de distribuição e de processos,

pois geralmente possuem apenas um processo e funcionam em um único equipamento com o mínino de comunicação com outros sistemas.

2.3 Casos de Uso

O RUP adota a técnica de modelo de casos de uso para modelar o problema para o qual o sistema deve apresentar uma solução e expressar os requisitos. Como veremos, a disciplina de análise e projeto tenta modelar uma solução que serve como entrada para a construção do sistema, mas para isso é imprescindível que o problema esteja bem especificado, ou seja, para resolvermos um problema, primeiro precisamos saber qual é o problema e o seu contexto.

Sendo assim, é preciso compreender alguns conceitos-chave do RUP:

- Caso de Uso: sucessão de ações executadas por um sistema, que rende um resultado observável de valor a um ator em particular [6].
- Ator: alguém ou algo fora do sistema, que interage com o sistema [6].

Nesse contexto entende-se por ação como sendo um procedimento iniciado por um ator ou disparado por um evento de tempo (através de um dispositivo temporizador) que pode repercutir em outros atores ou não, sendo a mesma atômica.

Os casos de uso representam funcionalidades do sistema, os quais devem acrescentar valor ao usuário (neste caso representado por um ator) e por isso os casos de uso geralmente têm o nome da funcionalidade respectiva.

As seqüências de ações podem ser vistas como um fluxo de eventos que descrevem como o sistema interage com os atores. Para representar uma funcionalidade do sistema existem diferentes cenários nos quais os atores e o sistema podem estar em estados diferentes, o que leva um caso de uso a ser representado por vários fluxos de eventos: um principal e outros alternativos.

Caso de Uso: Efetuar Login
Descrição: Este caso de uso é responsável por autenticar um usuário do sistema.
Pré-condição: Nenhuma.
Pós-condição: Um usuário válido é logado e sua sessão é registrada no sistema.
Fluxo Principal: 1. O cliente informa login e senha. 2. O sistema verifica se o login e a senha são válidos (verifica-se se o login e senha pertencem a uma conta). 3. O sistema registra o início de uma sessão de uso.
Fluxos Secundários: No passo 2, se o login e a senha forem inválidos, o sistema exibe uma mensagem e volta ao passo 1.

Quadro 2.1: Exemplo de Fluxo de Caso de Uso.

A descrição do fluxo de eventos é o conteúdo mais importante de um caso de uso. O mesmo é definido em linguagem natural simples, geralmente com um vocabulário reduzido e consistente. Existem outras técnicas para descrever o fluxo de eventos do caso de uso que não são cobertos pelo RUP, mas podem ser encontrados em [17].

Com base nos conceitos acima é possível entender o modelo é o conjunto de todos os casos de uso e todos os atores participantes, tendo a finalidade de descrever todas as funcionalidades do sistema. O modelo é definido em UML que veremos no capítulo seguinte.

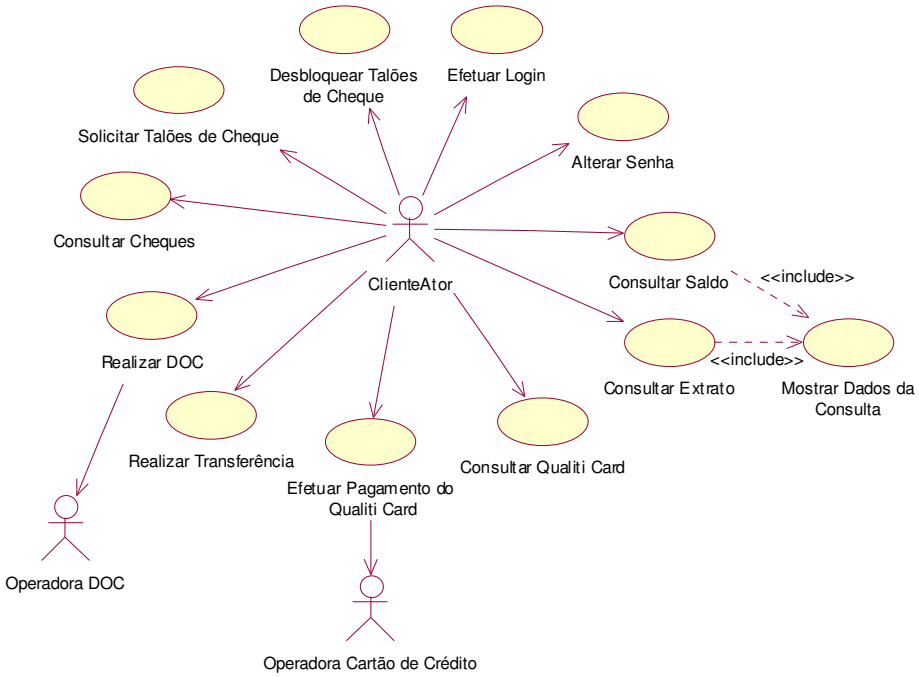


Figura 2.1: Exemplo de Modelo de Caso de Uso para um Sistema Bancário.

O modelo representa apenas os requisitos funcionais do sistema sendo um modelo *caixa-preta* do sistema, exibindo o comportamento do sistema sem preocupação de como as funcionalidades estão implementadas. É de responsabilidade da disciplina da análise e projeto atender a alguns dos requisitos não-funcionais e descrever como o sistema está estruturado internamente.

Como já fora citado, o RUP é um processo dirigido a casos de uso, ou seja, os casos de uso guiam todo o processo de desenvolvimento. O modelo tem origem na disciplina de requisitos a partir dos requisitos do sistema, servindo como um contrato entre equipe de desenvolvimento, clientes e usuários. Na disciplina de análise e projeto o modelo de caso de uso atua como principal entrada para a realização dos casos de uso, dando origem ao modelo de projeto.

Na disciplina de testes os casos de uso são a principal entrada para a extração dos cenários de teste do sistema. Na disciplina de gerenciamento de projeto, as fases e iterações são definidas baseadas nos casos de uso. Ainda na disciplina de gerenciamento, os mesmos servem para mensurar riscos, estimativas de esforço, entre outros.

2.4 Disciplina de Análise e Projeto

A disciplina de análise e projeto tem como principais metas traduzir os requisitos do sistema em um modelo capaz de descrever como o sistema deve ser construído e definir uma arquitetura estável e robusta capaz de suportar a solução escolhida.

A cada iteração no fluxo de atividades da disciplina, podemos separá-la em duas fases, uma de análise e outra de projeto. Na análise o foco está nos requisitos funcionais do sistema, ou seja, no problema, definindo um modelo mais simples para o sistema, através dos casos de uso.

Já a fase de projeto possui foco na solução escolhida, sendo a mesma dirigida pelos requisitos não-funcionais do sistema, na tentativa de atender o ambiente de implementação, onde o modelo gerado está mais próximo do código que será produzido.

Os papéis envolvidos nas atividades da disciplina são:

- Arquiteto: responsável por atividades técnicas e construção de artefatos do projeto. Possui uma visão abrangente e superficial do sistema. O mesmo decompõe a arquitetura nas visões do RUP, agrupa elementos de projeto e define as principais interfaces entre elementos do sistema.
- Analista de sistema: define as responsabilidades, operações, atributos e relações de uma ou várias classes, e determina como eles deveriam ser ajustados ao ambiente de implementação [6]. Sendo assim, o analista faz realização dos casos de uso levando em consideração a arquitetura definida.

Papéis opcionais:

- Projetista de banco de dados: Define a estrutura de dados da aplicação, como tabelas, índices, visões, *triggers*, entre outros. Faz-se necessário quando o sistema inclui um banco de dados.
- Projetista de cápsulas: assegura que o sistema possa responder a eventos de maneira pontual, por meio da utilização apropriada de técnicas coordenadas de projeto [6]. Tal papel é típico para sistemas em tempo real.
- Revisores de arquitetura e revisor de projeto: responsáveis pela revisão de artefatos da disciplina.

Dentre os papéis acima, o presente trabalho cobrirá apenas algumas das atividades relacionadas aos papéis do Arquiteto e do Analista de sistema, pois os mesmos são os mais importantes.

As atividades executadas pelo projetista de banco de dados não foram contempladas nesse trabalho pelo fato de existirem *frameworks* de mapeamento objeto-relacional largamente difundidos nos projetos de softwares atuais. Tais *frameworks* geram a estrutura do banco de dados a partir do conjunto de classes persistentes definido no modelo de projeto.

Os artefatos principais da disciplina são o documento de arquitetura e o modelo de análise e projeto (e outros artefatos que os compõe).

O documento de arquitetura é considerado padrão pelo RUP e define todas as visões arquiteturais com suas ressalvas vistas em tópico anterior. Muitas organizações dispensam a confecção de tal artefato pelo costume de utilizarem sempre a mesma arquitetura na sua linha de produtos de software ou porque adotam padrões de arquitetura bastante difundidos. Em caso de alteração em algum aspecto do padrão de arquitetura adotado, é aconselhado que a mesma seja documentada.

Por se tratar de um documento e não de um modelo em UML, este não fora contemplado no corrente trabalho.

O modelo de análise e projeto é considerado em dois artefatos distintos pelo RUP: um modelo de análise e outro de projeto. No entanto, como o modelo de projeto é apenas um refinamento do modelo de análise, é comum tratá-los como um único artefato.

O modelo de análise é considerado opcional pelo RUP, mas esse trabalho o adotou para a revisão dos processos da disciplina, melhorando a compreensão para o modelo de projeto. O modelo de análise contém apenas uma abstração simples do sistema visando o controle das funcionalidades extraídas dos casos de uso.

As classes do modelo de análise são mapeadas através de artefato específico em elementos do modelo de projeto. Tal mapeamento é útil quando a organização opta por manter os dois modelos para verificar a consistência entre os modelos. O mapeamento é um dos pontos tratados no capítulo 4.

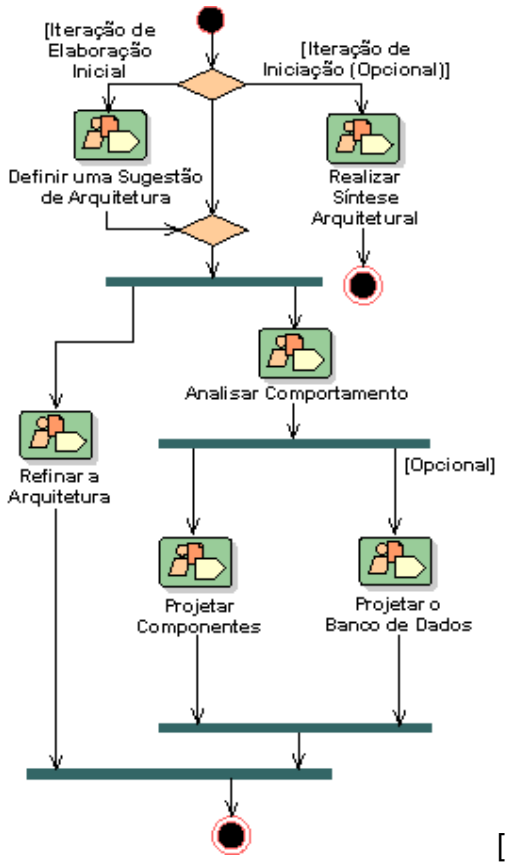
O modelo de projeto é um modelo de objeto que descreve a realização dos casos de uso e serve como uma abstração do modelo de implementação e seu código-fonte (artefato do conjunto de artefatos de implementação). O modelo de design é usado como base para atividades de implementação e teste [12].

De forma mais detalhada, o modelo de projeto consiste em colaborações de classes que podem ser agregadas em pacotes e subsistemas para ajudar a organizar o modelo e fornecer blocos compostos da construção dentro do modelo. Uma classe é a descrição de um conjunto de objetos que compartilham as mesmas responsabilidades, relações, operações, atributos e semântica [6].

Um pacote é um agrupamento lógico de classes, apenas para fins de organização do modelo e fica a critério do Arquiteto.

Um subsistema é um tipo de pacote, consistindo em um conjunto de classes que agem em unidade para fornecer comportamentos específicos [6]. Os conceitos inerentes ao modelo de projeto serão detalhados no capítulo 3.

Uma iteração típica da disciplina é demonstrada pelo seguinte diagrama de atividade:



[12]

Figura 2.2: Diagrama de Atividade da disciplina de Análise e Projeto.

O trabalho evidenciou apenas os estados de atividades Analisar Comportamento, Projetar Componentes e Refinar a Arquitetura. Cada estado de atividade possui um detalhe de fluxo de trabalho do estado, onde é exibido artefatos, atores, atividades, entre outros, e suas relações. Atividades podem se repetir em estados de atividades distintos.

As atividades pertinentes ao fluxo de análise e projeto são analisadas no capítulo 4.

3 UML

A UML é uma linguagem gráfica que auxilia na especificação, visualização e documentação de modelos de sistemas de software, incluindo sua estrutura e projeto. UML também é utilizada para a modelagem de negócios e quaisquer outros sistemas (de software ou não), através de seus perfis e das várias ferramentas disponíveis no mercado. A UML possibilita a analise dos requisitos de uma aplicação futura e projetar uma solução compatível, sendo a solução representada em um dos 13 tipos de diagramas da versão 2.0 da UML.[14]

A UML é uma linguagem mantida e definida pela *Object Management Group* (OMG), um consórcio internacional da indústria da computação sem fins lucrativos [7].

Um dos papéis centrais da OMG e suas forças tarefa para o desenvolvimento de software é a especificação de uma arquitetura dirigida a modelos, a *Model-Driven Architecture* (MDA). A MDA é uma tentativa de padronizar o desenvolvimento de software a partir da criação de modelos independentes de plataforma (PIM, do inglês *Platform Independent Model*), os quais podem ser mapeados para modelos de plataformas específicas (PSM, do inglês *Platform Specific Model*)[9].

No contexto da OMG a UML desempenha um papel crucial, pois é utilizada para descrever os padrões MDA e seus modelos. A UML também é utilizada para sua autodefinição e extensão.

Dentre os objetivos estabelecidos para a UML pela OMG estão:

- Fornecer mecanismos de extensão e especialização para estender os seus principais conceitos.
- Suportar especificações independentes de processos ou linguagem de programação.
- Suportar conceitos de mais alto nível como componentes.
- Fomentar criação de ferramentas de modelagem com objetos [9].

O fato de UML fornecer mecanismos para sua extensão vai de encontro com a possibilidade de extensão do RUP, o que permite que instâncias do RUP, como o RUP para sistema em tempo real, constituam novos perfis para UML.

A UML, pelo fato de não estar vinculada a nenhuma arquitetura ou linguagem de programação e aceitar conceitos de mais alto nível, não sofre de problemas de migração e atualização de seus modelos, o que é um problema dos sistemas legados modelados para plataformas específicas.

A UML também não define processos para criação de artefatos, como os diagramas, classes, pacotes e outros. A UML é independente de processo, definindo apenas os artefatos de modelagem, como os artefatos serão utilizados depende da metodologia adotada, que no contexto deste trabalho é o RUP.

3.1 Conceitos de Orientação a Objetos

Antes de prosseguir com os recursos da UML é necessário explicar alguns dos conceitos de orientação a objetos dando prosseguimento aos que foram citados no capítulo 2, como o conceito de classe.

Objetos são elementos físicos ou abstratos, ou seja, tangíveis ou não, passíveis de descrição, manipulação e destruição. O objeto é uma representação de uma entidade específica no mundo real, como por exemplo, um carro que acabara de sair de uma linha de montagem ou o cargo de gerente em uma empresa. Inserir figura exemplificando os modelos.

Complementando a definição de classe, uma classe que represente os carros teria como atributos o número do chassi, da placa, tipo de combustível, entre outros; como operações o comando de frear, acender os faróis, engatar a marcha, etc.

As classes (e conseqüentemente os objetos) necessitam se relacionar para que o modelo do mundo real realize ou represente algum comportamento. Sendo assim, existem três tipos de associações entre as classes: associação simples, agregação e composição. Adiante veremos suas definições e como são representadas em UML.

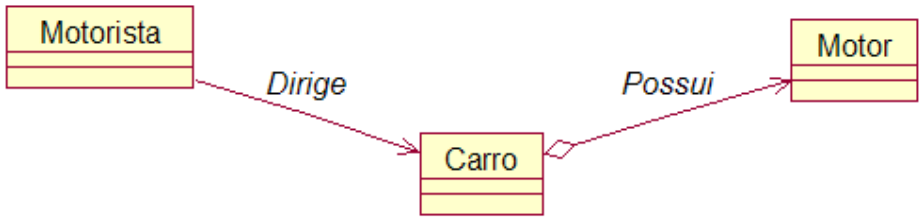


Figura 3.1: Exemplo de Relacionamento entre Classes.

Num contexto simples do ato de dirigir, um motorista dirige um carro. Na modelagem desse contexto, a classe motorista conhece a classe carro se associando a mesma pelo ato de dirigir. A mesma é uma associação simples e também pode ser direcionada no sentido da classe motorista para a classe carro pelo fato de não ser necessário que o carro conheça alguma informação do motorista.

Uma agregação é exemplificada pela relação entre a mesma classe carro do exemplo acima e a classe que representa os motores dos carros. Pode-se dizer que a relação entre a classe carro e a classe motor, no sentido carro-motor, é uma agregação pelo fato de o motor (e outras eventuais partes) em conjunto darem origem ao carro.

Uma composição é uma agregação mais restritiva em relação da parte com o todo, onde uma parte da relação de agregação não faz sentido de existir no modelo fora do todo que representa a agregação. Tal associação não foi utilizada pelo RADAS.

Todos os exemplos de relações acima dependem do contexto em que se encontram, ou seja, dependem do que esteja sendo modelado e podem variar entre modelos.

Um dos objetivos da UML é ser extensível e um mecanismo bastante utilizado é o estereótipo. Estereótipos auxiliam na identificação de elementos

nos modelos sem modificar seus fundamentos. A UML define três estereótipos para classes utilizados pela nossa modelagem: fronteira (<<*boundary*>>), controle (<<control>>) e entidade (<<entity>>).

O Estereótipo <<*entity collection*>>, oriundo de uma extensão UML sugerida pela Qualiti Software Process® e Centro de Informática – UFPE, também foi utilizado. Detalhes sobre a utilização e significado dos mesmos serão vistos no capítulo 4.

3.2 Diagramas

Toda a modelagem utilizando-se de UML é exibida graficamente através de um diagrama. Um diagrama ajuda os envolvidos no projeto de software a compreender o problema e a solução de pontos de vista diferente, onde cada tipo de diagrama proporciona uma visão única do sistema.

A UML versão 2.0 (UML 2.0) define 13 tipos de diagramas que podem ser organizados em diagramas estruturais, comportamentais e de interação, como no quadro abaixo:

Diagramas Estruturais
Classes
Objetos
Componente
Estrutura de Composição
Pacotes
Implantação
Diagramas Comportamentais
Caso de Uso
Atividades
Máquina de Estados
Diagramas de Interação
Seqüência
Comunicação (substituiu o de Colaboração da versão UML 1.4)
Tempo
Visão Geral de Interações

Quadro 3.1: Diagramas da UML 2.0

Dentre os diagramas estruturais, foram utilizados dois diagramas no decorrer deste trabalho, o diagrama de pacotes e o de classes.

3.2.1 Diagrama de Pacotes

Os pacotes são meios de organizar logicamente artefatos produzidos durante o projeto, como diagramas, classes, casos de uso, etc. Um diagrama de pacotes exibe as dependências entre pacotes ou outros elementos.

Um diagrama de pacotes pode ser utilizado para identificar dados, identificar a divisão do sistema em subsistemas (os pacotes que identificam subsistemas possuem o estereótipo <<*subsystem*>>), separar fases de um projeto, evidenciar as camadas da arquitetura (como é utilizado nos capítulos 4 e 6), entre outros.

3.2.2 Diagrama de Classes

O diagrama de classes modela os recursos e seus relacionamentos necessários para o funcionamento do sistema. Tais recursos se originam na confecção de outros diagramas (como visto na Atividade Analisar Caso de Uso do capítulo 4) e vão sendo incrementados na medida que novos diagramas e artefatos são gerados.

No mesmo tempo que os recursos do diagrama de classes são originados a partir de outros diagramas, o diagrama de classes também serve de base para os demais diagramas do modelo do sistema, como por exemplo, quando um diagrama de estados detalha os estados de uma classe.

A UML modela uma classe dividindo-a em três compartimentos gráficos, os quais contém informações como nome, atributos (informações que uma determinada classe possui) e operações. O primeiro compartimento exibe o nome como também o estereótipo e o pacote de origem; o segundo, os atributos e seus modificadores; e o terceiro, as operações.

Os relacionamentos nos diagramas de classe podem exibir informações como o nome da relação, o papel de cada uma das classes na relação e a aridade da mesma, como podemos ver na figura 3.2 a seguir:

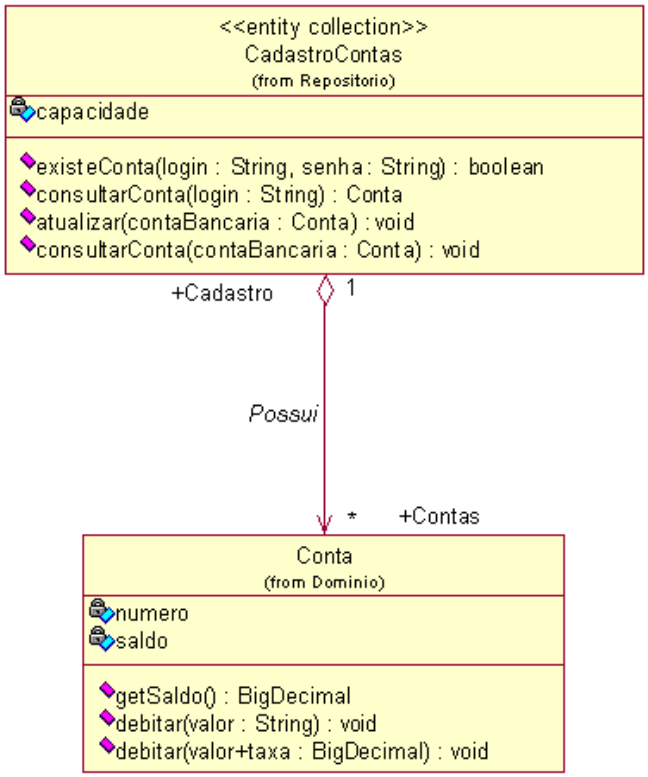


Figura 3.2: Exemplo de Exibição de Classes em um Diagrama de Classes.

No exemplo da figura 3.2 podemos entender a relação da seguinte maneira: um (1, representando parte da aridade) cadastro (+Cadastro) possui (Possui) nenhuma ou várias (*, representando a outra parte da aridade) contas (+Contas). Vale salientar que as classes e os relacionamentos podem ter informações omitidas pelo fato de não serem relevantes para um determinado diagrama.

3.2.3 Diagrama de Seqüência

O diagrama de seqüência faz parte do conjunto de diagramas de interação. Os diagramas de interação se resumiam em apenas dois diagramas na UML 1.4, seqüência e colaboração, evoluindo para os quatro diagramas citados no quadro 3.1. O diagrama de colaboração evoluiu para o diagrama de comunicação.

O CASE Rational Rose® ainda utiliza-se da nomenclatura diagrama de colaboração e o mesmo é citado nos demais capítulos desse trabalho para facilitar o entendimento.

O diagrama de seqüência demonstra a interação entre objetos no tempo, evidenciando a seqüência de mensagens trocadas entre os mesmos. Para que uma mensagem seja enviada para um objeto receptor, o objeto transmissor precisa saber como desse ser a assinatura dessa mensagem. Essa assinatura deve corresponder a uma operação da qual o objeto receptor pertence.

O RUP utiliza o mesmo para representar a execução dos diferentes cenários dos casos de uso, geralmente um para cada fluxo do caso de uso. Sendo assim, o diagrama tem um escopo bastante definido dentro da metodologia do RUP.

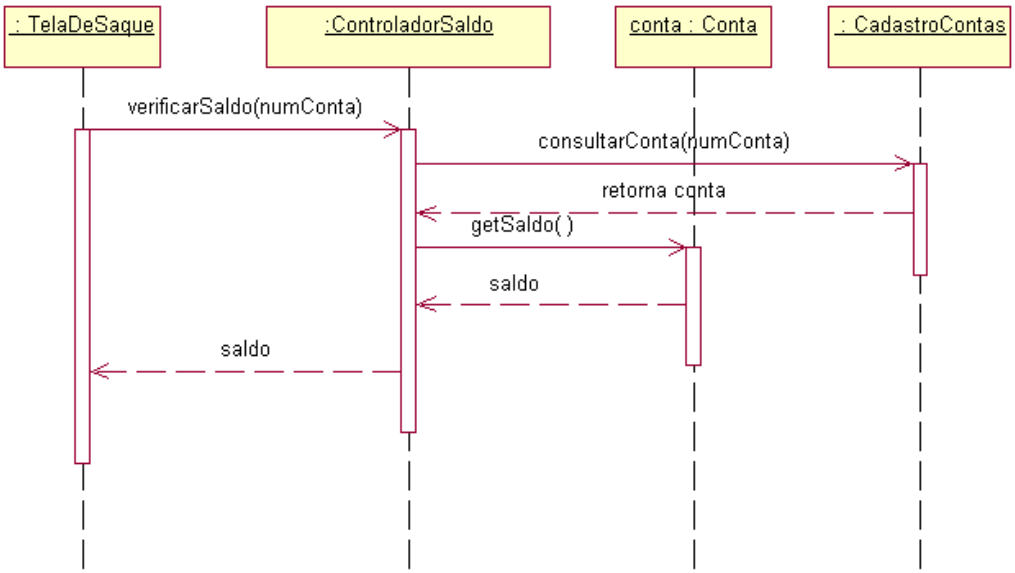


Figura 3.3: Diagrama de Sequência do Caso de Uso Verificar Saldo

No figura 3.3 vemos um diagrama de seqüência para o caso de uso onde o usuário verifica o saldo de sua conta. A tela do sistema envia uma

mensagem ao objeto que controla o fluxo desse cenário solicitando o saldo da conta. O controlador envia uma mensagem para o cadastro de contas solicitando uma conta cujo numero se encontra na mensagem. O cadastro devolve a conta requerida. O controlador envia uma mensagem para a conta solicitando seu saldo, qual é retornado em seguida para o controlador. Por fim, o saldo é enviado a tela.

O diagrama de colaboração e o diagrama de seqüência são equivalentes e poderão ser utilizados como entrada para o RADAS, como descrito no capítulo referente. A diferença do diagrama de colaboração é que esse evidencia a colaboração entre os objetos através de seus relacionamentos.

4 Processos Automatizáveis Identificados

Após uma revisão dos processos de Análise e Projeto do RUP, mais especificamente nos estados de atividades citados no capítulo 2 referentes à figura 2.2, foi possível sugerir formas de automatizar alguns passos de atividades, como também da confecção de artefatos e inferir regras de automação para os mesmos. As regras foram definidas por atividade como segue a organização abaixo.

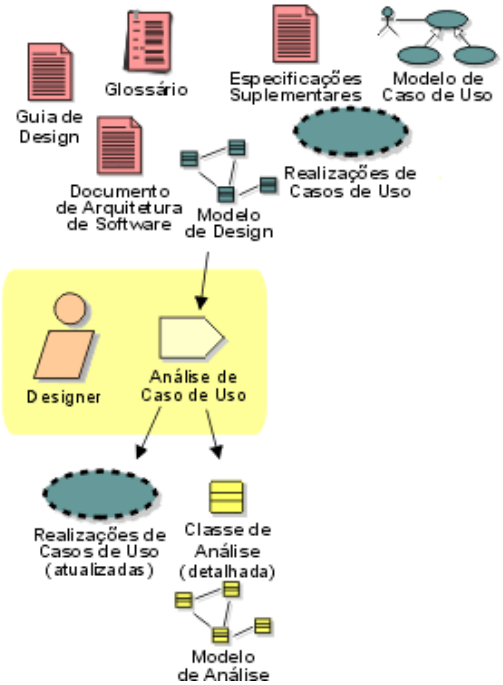
4.1 Atividade Analisar Caso de Uso.

Nessa atividade foram constados vários passos que podem ser totalmente ou parcialmente automatizados. Esta atividade é executada pelo Analista uma única vez para cada Caso de Uso priorizado para a iteração corrente.

As finalidades da atividade são identificar classes de análise capazes de realizar o fluxo de eventos do caso de uso em questão, designando comportamento, distribuindo atribuições, responsabilidades e associações entre as mesmas.

Durante a atividade, o Analista deve se ater apenas a requisitos funcionais relativos ao Caso de Uso, não se preocupando em como será a implementação, pois isso será o foco da atividade Projetar Caso de Uso.

A figura abaixo demonstra alguns artefatos de entrada e saída desta atividade, como também o papel responsável pela mesma.



[12]

Figura 4.1: Detalhe de Fluxo com Atividade Análise de Caso de Uso.

Em resumo os passos da atividade são:

1. Identificar classes de análise.
2. Identificar classes persistentes.
3. Distribuir comportamento entre as classes identificadas.
4. Descrever responsabilidades para as classes identificadas.
5. Descrever atributos e associações para as classes identificadas.

O passo 1 consiste em determinar classes com estereótipos vistos no capítulo de UML. Geralmente cada caso de uso possui apenas uma classe de controle, uma classe de fronteira para cada Ator ou Sistema Externo envolvido no fluxo do caso de uso e classes de entidades extraídas a partir de substantivos relevantes do fluxo do caso de uso.

No apêndice se encontram fluxos de casos de uso utilizados para testar o RADAS os quais contém os substantivos que deram origem às classes de análise dos respectivos casos de uso.

No passo 2 é determinado quais das classes de entidades identificadas precisam ser persistidas no sistema. Tal classificação é verificada quando a entidade precisa ser recuperada (ou armazenada) durante o fluxo do caso de uso.

Além disso, no passo 2 são geradas classes cujos estereótipos não pertencem as versões da UML, que são as classes <<*entity collection*>>. Tais classes representam a noção de uma coleção de classes de entidade, geralmente utilizado em classes de cadastro para uma classe de entidade persistente. As mesmas também podem ser verificadas nos exemplos do Apêndice.

No passo 3 é atribuído comportamento entre classes identificadas para o caso de uso, de forma que as mesmas interajam entre si, colaborando para a execução do fluxo de eventos do caso de uso.

Durante esse passo são construídos os diagramas de interação definidos no capítulo UML. Dadas as classes identificadas, é necessário construir ao menos um diagrama de interação (seqüência, colaboração ou ambos), de preferência quantos forem necessários para demonstrar todo o comportamento das classes nos diferentes fluxos do caso de uso.

É importante salientar que durante a atividade Analisar Caso de Uso não são levadas em consideração classes analisadas em outros casos de uso. Tal fato só ocorre na atividade Projetar Caso de Uso, executada pelo Arquiteto como veremos em seção posterior.

No passo 4 temos de inferir as responsabilidades de cada uma das classes durante a execução do fluxo do caso de uso. Nesse passo faz sentido o início da construção do diagrama de classes para o caso de uso, pois nesse diagrama serão exibidas as responsabilidades sob a forma de operações na representação UML da classe no diagrama.

Outro ponto que deve ser ressaltado é que durante a fase de análise, o analista não deve se preocupar em definir com exatidão os parâmetros e retornos das operações, pois isso é uma preocupação da fase de projeto. Mas como muitas das operações não mudarão (ou mudarão de forma simples) e como também muitas vezes já é sabida com exatidão a definição da operação, é interessante definir a mesma já na fase de análise.

Nesse passo as operações são extraídas a partir das mensagens dos diagramas de interação do caso de uso construído no passo 3. Esse trabalho executado pelo Analista pode ser realizado automaticamente por uma ferramenta pois, baseado na definição da mensagem, é possível extrair ao menos o nome da operação.

Uma sugestão que agrega bastante valor, poupando tempo ao Analista, seria o mesmo definir a mensagem nos diagramas de interação utilizando-se de todo o potencial da notação UML mostrada no capítulo UML, pois através da mesma é possível inferir o tipo de retorno, tipos e nomes dos parâmetros, possibilitando automatização por completo desse passo.

O último passo consiste em definir atributos e relacionamentos para as classes baseado em conhecimentos do negócio da aplicação, dos requisitos, e outros artefatos mostrados na figura 4.1 e representá-los no diagrama de classes seguindo as definições para as associações visto no capítulo UML.

Ainda que o processo de inferir relacionamentos seja bastante subjetivo, regras podem ser inseridas para definição dos relacionamentos. No entanto, deve-se ressaltar que as regras nunca devem prevalecer sobre o conhecimento do Analista.

Abaixo seguem as regras e suas exceções:

1. Para toda mensagem entre instâncias de duas classes distintas é gerada uma associação simples no mesmo sentido da mensagem no diagrama de classes. Caso exista outra mensagem em sentido contrário entre instâncias das mesmas classes anteriores deverá ser gerada uma outra associação no sentido contrário alertando o Analista de que existe uma inconsistência no modelo.
2. Para uma automensagem não será gerada uma auto-associação na classe da instância em questão, a menos que exista a mesma classe definida como um parâmetro ou retorno da operação na operação correspondente a essa automensagem. Tal fato pode ser constatado no caso de uma estrutura de dados do tipo lista ligada.
3. Para toda operação definida em uma mensagem são geradas dependências entre as instâncias das classes envolvidas na mensagem (remetente e receptor) e as classes definidas como parâmetros e retorno da operação em questão.

3.1. Se a classe receptora da mensagem for uma <<*entity collection*>> e uma das classes definidas nos parâmetros ou tipo de retorno for uma entidade (<<*entity*>>) deverá ser gerada uma agregação da classe << *entity collection* >> para a classe de entidade. Caso haja mais de uma classe de entidade definida, deverá ser gerada uma agregação para cada uma e exibida no diagrama de classes evidenciando ao Analista uma inconsistência no modelo gerada por essa regra.

4. No caso de haver uma instância de uma classe sem comunicação com outras instâncias ou mesmo se a classe da instância não possuir nenhum relacionamento com alguma outra classe presente no diagrama, a classe da instância isolada deverá ser exibida no diagrama de classes do caso de uso, evidenciando uma possível inconsistência ou falta no diagrama de interação.

Nota-se que as regras acima podem entrar em conflito e a sugestão que foi acatada no RADAS é que em caso de conflito permanecerá o relacionamento que fora anteriormente definido.

Tal sugestão permite que as funcionalidades do RADAS se apliquem para o modelo de projeto, evitando que uma posterior substituição por um relacionamento diferente gere inconsistências em outros diagramas previamente inspecionados pelo Arquiteto e Analista.

4.2 Atividade Projetar Arquitetura.

Essa atividade também é conhecida como Identificar Elementos de Design [12] e consiste em analisar as interações das classes de análise no modelo de análise com o intuito de identificar elementos de projeto. Essa atividade é desempenhada uma única vez na iteração pelo Arquiteto, o qual leva em consideração o modelo de caso de uso, o documento de arquitetura, o modelo de análise (realizações dos casos de uso), entre outros artefatos.

Como saída dessa atividade temos o modelo de projeto atualizado (muitas vezes uma evolução do modelo de análise como já fora citado), o mapeamento dos elementos de análise em elementos de projeto e uma possível atualização no documento de arquitetura.

A mesma pode ser resumida para este trabalho nos seguintes passos:

1. Identificar e mapear elementos de projeto.
2. Identificar possibilidade de reuso.
3. Organizar os elementos na estrutura da aplicação.

O passo 1 consiste em identificar elementos de projeto como classes de projeto, subsistemas e suas interfaces e documentar a origem dos elementos identificados através de um mapeamento entre as classes de análise e os novos elementos.

O mapeamento é feito pela seguinte regra: Uma dada classe de análise pode gerar nenhum ou vários elementos de projeto.

Mesmo o mapeamento não sendo um artefato obrigatório, este é de extrema importância, uma vez que através dele é possível manter o rastreamento entre os modelos de análise e projeto, permitindo uma revisão das decisões de projeto e retrocesso do modelo se necessário.

Na maioria das instituições os mapeamentos são confeccionados em ferramentas de planilhas ou textos armazenados em documentos separados do modelo de análise e projeto e fora da ferramenta CASE, algo extremamente desestimulado pela Metodologia do RUP, pelos motivos explicados no capítulo RUP. Dessa maneira esse artefato é de pouca utilidade e caiu em desuso.

Este trabalho se propõe a confeccionar o artefato de mapeamento das classes de análise em elementos de projeto na ferramenta CASE e incorpora-lo ao modelo de análise e projeto.

Como veremos no capítulo 7 é possível automatizar parcialmente o processo de atualização os diagramas das realizações de casos de uso na atividade Projetar Caso de Uso (e outras) tendo o mapeamento incorporado ao modelo de análise e projeto.

No passo 2 são identificadas possibilidades de reuso através da inspeção de pacotes, subsistemas e até classes para verificar similaridades com outros componentes. Tais componentes podem estar disponíveis no mercado ou na própria instituição para substituir um desses elementos, como também esses elementos podem dar origem a componentes para utilização em projetos futuros.

O último passo consiste em organizar os elementos de projeto na estrutura da aplicação, definida pelo padrão de arquitetura ou, caso não seja seguido o padrão, a estrutura definida no documento de arquitetura. Se uma estrutura ainda não fora bem definida, esse é o momento adequado para refiná-la e sugerir modificações.

Também é nesse passo que é levado em consideração o meio de armazenamento.

A estrutura da aplicação é representada no modelo através de um diagrama de pacotes e suas dependências, com cada pacote representando uma camada da arquitetura ou apenas uma organização conveniente de um conjunto de elementos.

Organizar os elementos em pacotes é uma tarefa simples, mas identificar e criar as dependências entre os mesmo pode se tornar uma tarefa árdua para um ser humano e sua complexidade depende diretamente do número de elementos e do padrão de arquitetura adotado.

Tal fato pode levar a falta de relacionamentos no diagrama, não detecção de dependências cíclicas ou de dependências que não deveriam existir. Dessa maneira, tais falhas podem passar para o modelo de implementação e conseqüentemente para o código da aplicação.

A regra para a resolução de tal fato é simples e de fácil automatização por uma ferramenta CASE. A regra consiste em gerar uma dependência de um pacote A para um outro pacote B se existir qualquer relacionamento de um elemento contido em A para algum elemento contido em B.

4.3 Atividade Projetar Caso de Uso

A Atividade Projetar Caso de Uso é desempenha pelo Analista e consiste em refinar a realização de caso de uso através da atualização dos diagramas com os novos elementos de projeto.

O Refinamento leva em consideração as decisões de arquitetura como, por exemplo, a decisão sobre a camada de persistência (atualizando os diagramas com as interfaces da mesma camada), os subsistemas derivados de complexas interações nos diagramas (substituindo tais interações por chamadas as interfaces dos novos subsistemas), etc.

A atividade é realizada para cada Caso de Uso gerando um modelo de projeto capaz de suportar a implementação de todas as funcionalidades do sistema, respeitando os requisitos não-funcionais.

Grande parte da atualização dos diagramas poderia ser realizada de forma automática caso o mapeamento de elementos de análise em elementos de projeto fosse incorporado ao modelo de análise e projeto, como sugerido no item anterior (ver Atividade Projetar Arquitetura). No entanto, apenas o

mapeamento não seria suficiente para realizar tais transformações de forma precisa.

5 Ferramentas Analisadas

Um dos objetivos deste trabalho é analisar ferramentas CASE disponíveis atualmente com o intuito de verificar se possuem características que sigam as sugestões e regras definidas no capítulo 4, suportando a UML e os processos do RUP, além de verificar a possibilidade de as sugestões e as regras serem adicionadas às ferramentas.

Uma vasta lista de ferramentas CASE pode ser encontrada em [15] de onde foram extraídos alguns nomes para a análise e outras foram escolhidas pela disponibilidade de acesso as mesmas pelo autor.

As ferramentas avaliadas foram:

- Enterprise Architect[16]
- JUDE/Community[4]
- ArgoUML[13]
- Rational Rose®

A ferramenta Enterprise Architect é uma poderosa ferramenta de modelagem UML que suporta a versão 2.1 da UML, com integração a várias IDE, mas não possui uma interface aberta para construção de *plugin* para a ferramenta. Esse foi um dos fatores que inviabilizou a sua utilização neste trabalho, assim como o fato de a ferramenta estar disponível para avaliação por um curto período de tempo.

JUDE/Community é uma ferramenta mais simples que a Rational Rose® e a Enterprise Architect, mas com todas as funcionalidades necessárias para modelagem UML 1.4. A ferramenta foi desenvolvida em JAVA, o que possibilita a sua portabilidade para outras plataformas.

Muitos grupos de desenvolvimento utilizam-se do JUDE/Community para suportar outras metodologias de desenvolvimento de software como pode ser constatado em [5], além do RUP. Mesmo possuindo características para o desenvolvimento do *plugin* implementasse as regras definidas nesse trabalho, o JUDE/Community não possui uma interface de extensão aberta.

ArgoUML não se resume a uma ferramenta de modelagem UML, sendo o mesmo um projeto de desenvolvimento de código aberto [13]. A ferramenta suporta a UML 1.4 com muitas outras funcionalidades que ajudam a interação com o usuário, com outras novas sendo desenvolvidas.

Apesar de existir ferramenta comercial que estende do projeto ArgoUML, tal fato não refletiu na redução do risco de desenvolvimento do *plugin*, pois a implementação terá de ser feita a partir do código fonte do projeto e não a partir de uma interface bem definida e específica.

5.1 Rational Rose®

A Ferramenta Rational Rose® é parte da família Rational Software da IBM® e auxilia o desenvolvimento de software em análise, modelagem, projeto e construção do sistema. A mesma se baseia na modelagem UML e fornece todo o arcabouço de funcionalidades necessárias para suportar o RUP.

Três fatores foram decisivos para a adoção da ferramenta Rational Rose® o desenvolvimento do RADAS: a disponibilidade de acesso à mesma nos laboratórios do Centro de Informática – UFPE, o fato de a ferramenta ser largamente adotada pelas empresas de desenvolvimento de sistemas e a existência de uma interface de programação da ferramenta Rational Rose® para o desenvolvimento de *plugin*.

5.1.1 Rose Extensibility Interface (REI)

O propósito da Rational Rose® é prover o desenvolvimento de software baseado em componentes. Como era esperado, o aplicativo do Rational Rose® foi desenvolvido sobre componentes, e é definido no modelo do Rose Extensibility Interface como mostrado na figura abaixo.

O modelo da REI é essencialmente um metamodelo do modelo da Rational Rose® , expondo seus pacotes, classes, propriedades e métodos, os quais definem e controlam o aplicativo e todas as suas funcionalidades.

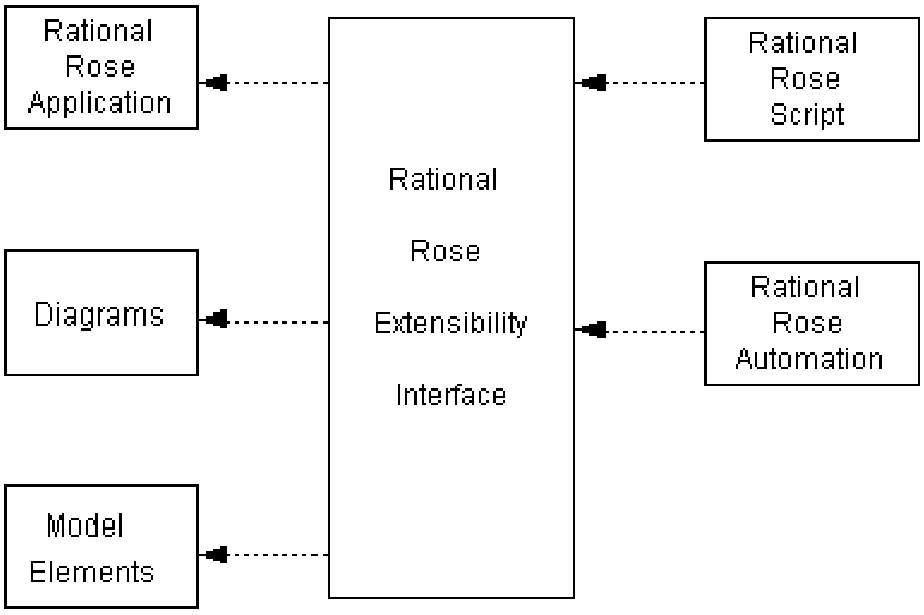


Figura 5.1: Estrutura de Pacotes do Rational Rose®

Através da REI é possível:

- Configurar menus e telas.
- Automatizar funções da ferramenta através do Rational Rose Scripts (para gerar diagramas e classes, atualizar modelos, gerar documentação, entre outros).
- Executar funções da ferramenta através de outra aplicação, utilizando-se do objeto Rational Rose Automation.
- Acessar as classes, propriedades e métodos através de uma IDE, apenas incluindo a biblioteca de tipos da REI nas referências do projeto na IDE.

6 RADAS

RADAS é a concretização deste trabalho na forma de um aplicativo que automatiza algumas da regras e sugestões definidas no capítulo 4, complementando as funcionalidades da ferramenta CASE Rational Rose®. Para utilizar o RADAS é preciso apenas iniciá-lo e juntamente, o Rational Rose® também será iniciado. O usuário deverá operar o Rational Rose® normalmente e quando for necessário utilizar as funcionalidades do RADAS. O usuário deve pressionar o botão *Lock Model.* Após pressionar o botão, o aplicativo Rational Rose® desaparecerá impedindo que o modelo seja modificado, habilitando as funcionalidades do RADAS. Para retornar ao Rational Rose® basta clicar no mesmo botão, agora renomeado como *Unlock Model.*

O RADAS foi construído por funcionalidades e assim modularizado internamente e divido em sua interface gráfica com o usuário. Na interface gráfica encontram-se três grupos de controles formados de acordo com as três diferentes funcionalidades do RADAS, explicadas abaixo.

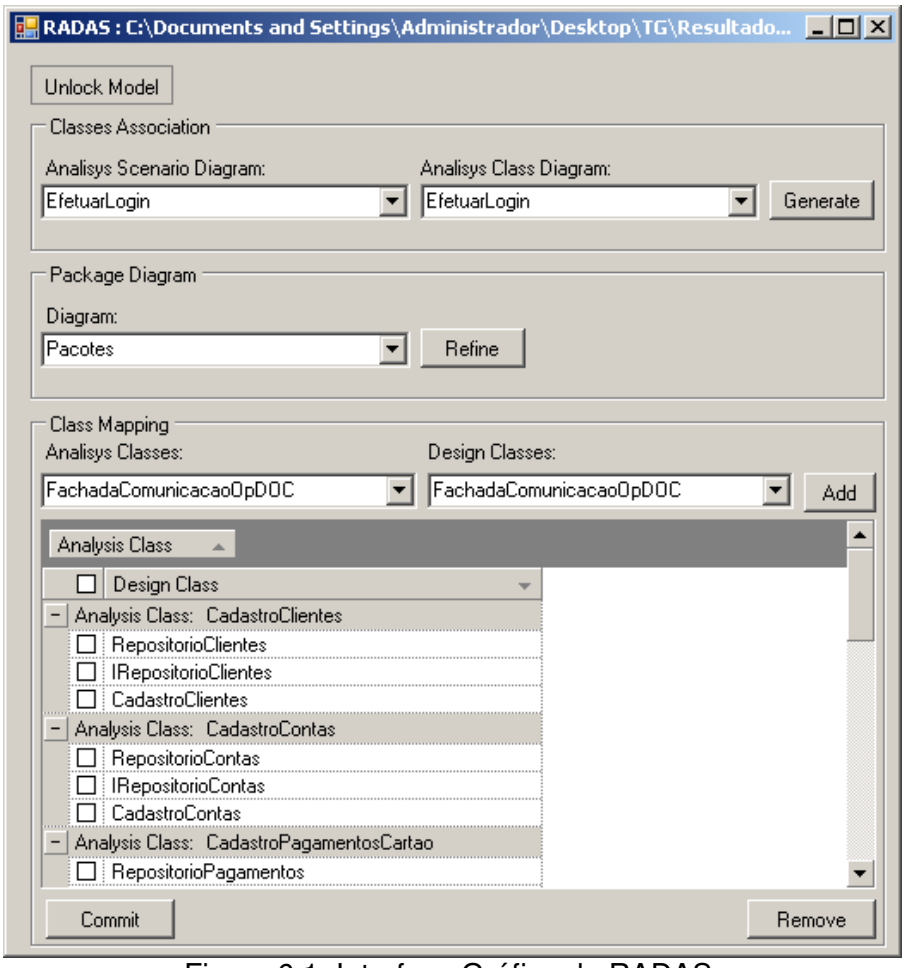


Figura 6.1: Interface Gráfica do RADAS.

6.1 Inferência de Relacionamentos e Operações

Essa funcionalidade identificada pelo grupo de controles *Classes Association* na figura 6.1 acima é responsável pela inferência de relacionamentos e operações através das instancias de objetos e mensagens presentes em um diagrama de interação (seqüência ou colaboração, selecionado na lista *Analisys Scenario Diagram* que exibe os diagramas do pacote *LogicalView* do Rational Rose®).

Os relacionamentos e operações são adicionados às respectivas classes no modelo, as quais serão exibidas em um diagrama de classes (vazio ou não, selecionado através da lista *Analisys Class Diagram* que mostra os diagramas do pacote *LogicalView* do Rational Rose®).

A Inferência de relacionamentos e operações é feita baseada nas regras definidas no capítulo 4, mais especificamente as regras presentes no item Atividade Analisar Caso de Uso.

O conflito entre as regras se dá quando uma das regras deve gerar um relacionamento entres classes e já existe um outro relacionamento entre as mesmas classes. A resolução do conflito é simples e consiste em permanecer o relacionamento mais antigo.

Tal resolução fora adotada para evitar que possíveis substituições gerem inconsistências em outros diagramas já analisados.

Na ocasião da inferência de operações, duas operações idênticas não serão geradas. Para as operações serem consideradas idênticas, estas devem ter o mesmo nome, mesmo tipo de retorno, mesmo número de parâmetros, os quais devem ser dos mesmos tipos e definidos na mesma seqüência.

As operações são inferidas através do texto da mensagem entre duas instâncias do diagrama de interação que estiverem no formato definido pelo padrão UML visto no capítulo 3. Exemplos:

- Operation()
  - Nesse exemplo o tipo de retorno será definido como *void*
- Operation(param1:=Param1Type, param2:=Param2Type):=ReturnType

Após a escolha dos diagramas é necessário apenas pressionar o botão *Generate* para executar essa funcionalidade e logo em seguida será exibido o diagrama de classes modificado no Rational Rose®

6.2 Refinamento do Diagrama de Pacotes

Essa funcionalidade identificada pelo grupo de controles *Package Diagramn* na figura 6.1 é responsável pela inferência de dependências entre os pacotes presentes no diagrama selecionado na lista *Diagram,* a qual exibe os diagramas de pacote da *LogicalView* do Rational Rose®.

Tal funcionalidade é baseada nas regras do capítulo 4, mais especificamente nas regras presentes no item Atividade Projetar Arquitetura.

A funcionalidade em questão é acionada através do botão *Refine*, o que ocasiona um expressivo tempo de processamento, proporcional a quantidade de classes presentes em cada pacote exibido no diagrama, e logo após é

exibido o diagrama no Rational Rose®. Essa também considera todos os pacotes internos aos pacotes existentes no diagrama.

6.3 Mapeamento de Classes

Essa funcionalidade identificada pelo grupo de controles *Class Mapping* na figura 6.1 é responsável pelo mapeamento de classes de análise em classes de projetos presentes no pacote da *LogicalView* do Rational Rose®.

No grupo de controle encontram-se duas listas idênticas: a localizada à esquerda da interface (identificada por *Analisys Classes*) serve para seleção de uma classe de análise e a localizada à direita da interface (identificada por *Design Classes*) serve para seleção de uma classe de projeto.

As duas listas são idênticas porque não existe diferença entre classes de análise e classes de projeto no modelo do Rational Rose®.

Tal funcionalidade foi baseada nas regras do capítulo 4, mais especificamente nas regras presentes no item Atividade Projetar Arquitetura, com algumas ressalvas. As ressalvas implicam em um modo um pouco diferente de mapeamento comparado ao mostrado no Apêndice.

As ressalvas fazem com que o mapeamento seja realizado considerando que se alguma classe de projeto não teve origem a partir de alguma classe de análise, a mesma não pertence ao mapeamento, como também se alguma classe de análise tornar-se uma classe de projeto (em palavras: continuar no modelo de projeto) a mesma deve mapear ela mesma como classe de projeto. Um exemplo ocorre na figura 6.1.

O mapeamento deve ser feito dessa maneira para permitir uma funcionalidade futura que será discutida no capítulo 7 e proposta no capítulo 4, mais especificamente no item Atividade Projetar Caso de Uso.

Para adicionar um mapeamento é necessário selecionar classes em ambas as listas e pressionar o botão *Add* e automaticamente será inserida uma linha na grade exibida na interface representando o mapeamento.

Para remover um ou mais mapeamentos basta selecioná-los na coluna mais a esquerda da grade e depois clicar no botão *Remove*.

Para que o mapeamento persista no modelo é necessário clicar no botão *Commit*.

6.4 Testes

Os testes do RADAS foram realizados com a análise e projeto dos fluxos de caso de uso contidos no Apêndice: Efetuar Login, Realizar DOC e Realizar Pagamento Qualiti Card referentes.

Os testes aconteceram seguindo o diagrama de atividades descrito no capítulo 3, passando apenas pelas duas atividades, Atividade Analisar Caso de Uso e Atividade Projetar Arquitetura vistas no capítulo 4. A seqüência natural das atividades faz com que as funcionalidades do RADAS sejam utilizadas na ordem em que aparecem na interface gráfica com o usuário.

Seguem alguns testes divididos por funcionalidade para o caso de uso Efetuar Login:

6.4.1 Inferência de Relacionamentos e Operações

Analisando o fluxo do Caso de Uso Efetuar Login foram geradas classes de análise e o seguinte diagrama de seqüência abaixo, utilizando-se de instâncias das classes geradas:

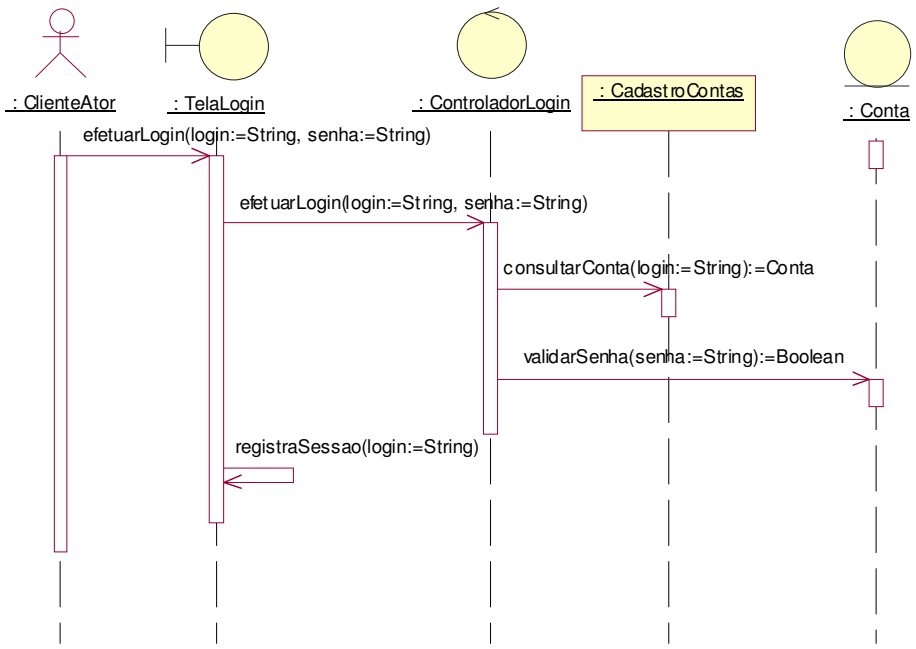


Figura 6.2: Diagrama de Sequência para o Caso de Uso Efetuar Login

A partir do diagrama acima o RADAS gerou o seguinte diagrama de classes:

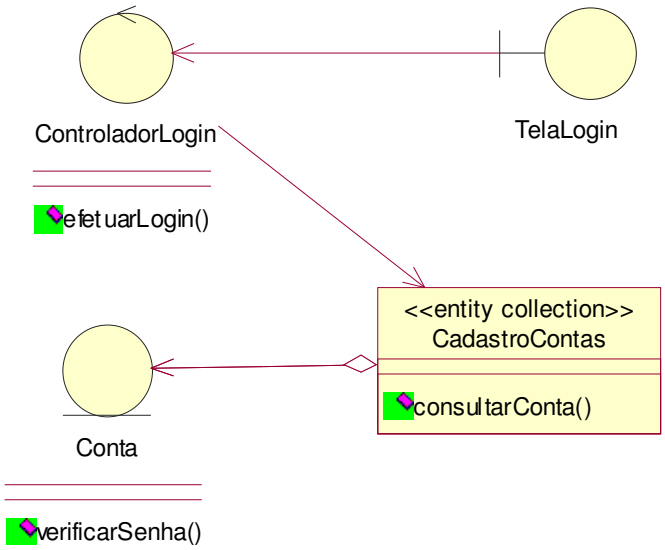


Figura 6.3: Diagrama de Classes resultante do Diagrama da figura 6.2.

6.4.2 Refinamento do Diagrama de Pacotes

Com o uso da funcionalidade Inferência de Relacionamentos e Operações foram gerados os diagramas de classes para os três casos de uso citados acima, e conseqüentemente suas operações e relacionamentos, alterando o diagrama de classes visto no teste anterior. Os diagramas de classe para os demais casos de uso podem ser vistos no Apêndice.

Após gerados, todos os diagramas, todas as classes presentes no pacote *Logical View* do Rational Rose® foram distribuídas nos pacotes: Tela, Comunicação, Controle, Domínio e Repositório.

A distribuição foi realizada com os seguintes critérios:

- As classes de fronteira que representavam interfaces gráficas com o usuário ficaram no pacote Tela.
- As classes de fronteira que representavam comunicação com sistemas externos ficaram no pacote Comunicação.
- As classes de controle de cada caso de uso ficaram no pacote controle.
- As classes de entidade, persistentes ou não, ficaram no pacote Domínio.
- As classes de cadastro, cujos estereótipos são do tipo *<<entity collection>>*, ficaram no pacote Repositório.

Após a distribuição os pacotes foram dispostos no seguinte diagrama de pacotes:

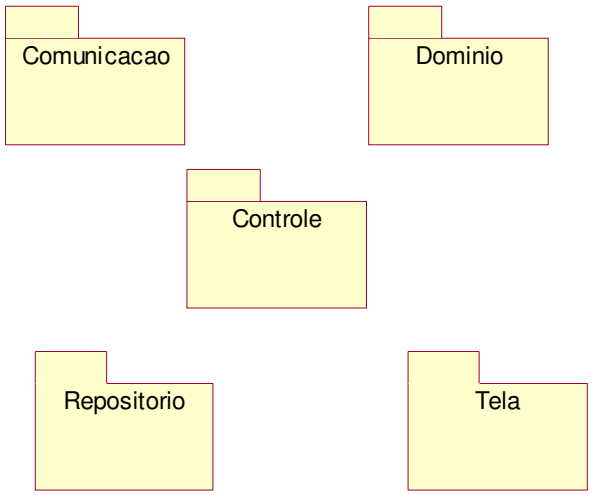


Figura 6.4: Diagrama de Pacotes antes Refinamento.

A aplicação da funcionalidade atualizou o diagrama de pacotes da seguinte maneira:

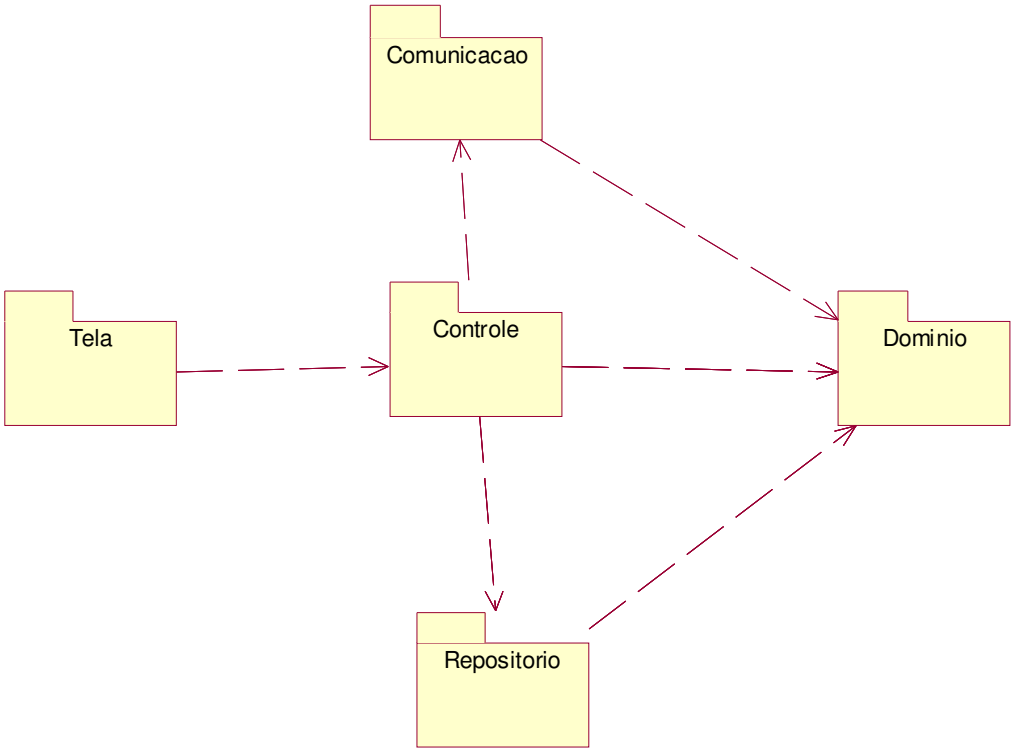


Figura 6.5: Diagrama de Pacotes após Refinamento.

6.4.3 Mapeamento de Classes

A funcionalidade de mapeamento foi realizada com as classes de projeto geradas a partir das classes de análise do diagrama resultante do teste da funcionalidade de Inferência de Relacionamentos e Operações.

As classes de projeto foram distribuídas nos pacotes exibidos no diagrama de pacotes mostrados no teste da funcionalidade Refinamento do Diagrama de Pacotes.

Na figura abaixo vemos as classes de projeto agrupadas no quadro pela classe de análise:

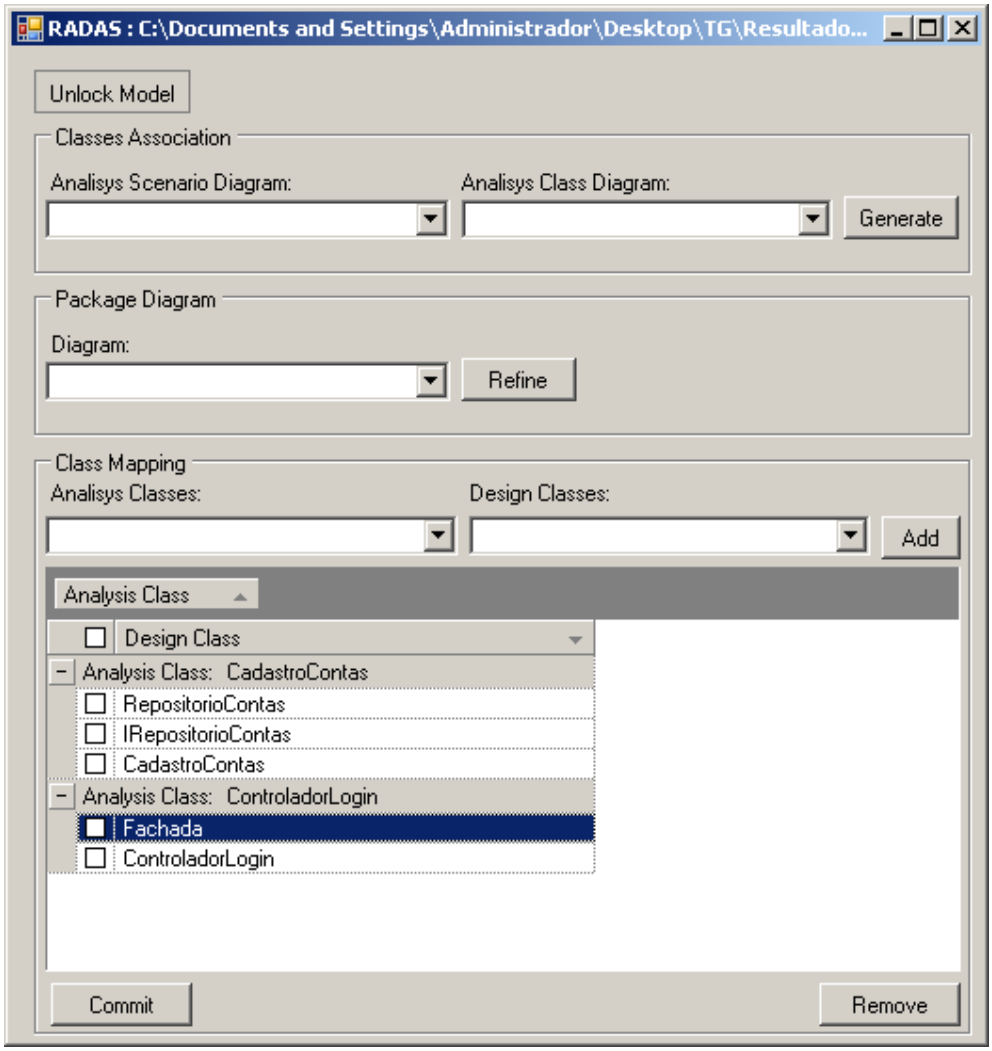


Figura 6.6: Interface Gráfica do RADAS exibindo mapeamento de Classes.

6.5 Detalhes Técnicos

Na proposta inicial deste trabalho o RADAS era desejável para a consolidação das regras e definido como forma de *plugin* para a ferramenta Rational Rose®, mas por problemas técnicos encontrados na Rose Extensibility Interface (REI) não foi possível o desenvolvimento do RADAS em forma de plugin, incorporando funcionalidades diretamente a menus e telas da ferramenta.

O problema consistia em uma combinação de fatores, dos quais o mais importante foi o fato de que a REI apenas se comunica com objetos do tipo *Component Object Model* (COM) utilizados para intercambiar componentes produzidos em diferentes linguagens de programação como Visual Basic (descontinuada pela fabricante e mantenedora Microsoft®) e outras mais antigas. O mesmo fato impede que a REI se comunique com objetos de intercambio das atuais linguagens de programação como Java® e C#.

No capítulo 5 foi constatado que a ferramenta Rational Rose® seria a única ferramenta cabível dentre as analisadas. Sendo assim, a outra maneira encontrada foi utilizar-se do Rose Automation, que permite a uma outra aplicação controlar o Rational Rose®, tendo a mesma interface de comunicação definida na REI.

A mudança técnica na implementação do RADAS trouxe uma melhora, pois o mesmo não precisa ser instalado no computador para ser utilizado, apenas o Rational Rose® deve estar devidamente instalado.

O RADAS foi desenvolvido utilizando a plataforma .NET da Microsoft® e programado na linguagem de programação C# através da IDE (*Integrated Development Enviorement*) Micrisoft Visual Studio 2005®.

Mesmo o RADAS sendo desenvolvido utilizando-se da plataforma .NET, o que permite uma velocidade maior de produção do aplicativo, houve atrasos no desenvolvimento devido à sinergia de dois principais fatores: ausência de documentação ou documentação imprecisa da REI contida nos arquivos de Ajuda da ferramenta Rational Rose®; e impossibilidade de depurar a chamadas ao objeto COM que implementa a REI durante a execução do RADAS.

Tais fatores levaram a testes isolados das declarações da interface para descobrir o correto funcionamento da REI.

7 Conclusões e Trabalhos Futuros

A revisão da disciplina de Análise e Projeto do RUP demonstrou que ainda existe espaço para automação de suas atividades por ferramentas CASE mesmo o RUP sendo um processo bem definido e difundido na indústria, juntamente com ferramentas de suporte consagradas como o Rational Rose®.

Os testes realizados sobre o RADAS com os três casos de uso referenciados no capítulo 6 demonstraram-se satisfatórios para a funcionalidade Inferência de Relacionamentos e Operações, pois reproduziram diagramas de classes consistentes com os diagramas de seqüência correspondentes. Para as demais funcionalidades os resultados demonstram que as respectivas regras de automação estão corretas. O RADAS ficará disponível para os integrantes do Centro de Informática – UFPE possibilitando um retorno contínuo sobre sua eficácia e usabilidade.

As regras de automação, quando incorporadas a uma ferramenta CASE como o Rational Rose®, aceleram o desenvolvimento do sistema uma vez que elimina passos manuais e repetitivos, antes realizados pelos Analistas e Arquitetos.

O modo como as regras foram implementadas no RADAS possibilita que inconsistências e lacunas sejam identificadas precocemente no modelo do sistema, como por exemplo, quando uma instância de uma classe em um diagrama de seqüência não participa de nenhuma troca de mensagens. Ao utilizar-se da funcionalidade de Inferência de Relacionamentos e Operações, a referida classe é adicionada ao diagrama de classes sem nenhum relacionamento, evidenciando a falta de mensagens no diagrama de seqüência ao Analista.

Tais contribuições repercutirão em um aumento da produtividade dos Analistas e Arquitetos, uma vez que os mesmos terão mais tempo para outras atividades e poderão modelar sistemas mais rapidamente e mais complexos.

Outros trabalhos também abordam o tema de automação para o desenvolvimento de sistemas, seja definindo uma nova metodologia ou automatizando processos baseados em metodologias já consolidadas como o RUP[20]. Tais trabalhos não propõem eliminar processos manuais, como aqui foram realizados, mas sistematizar o processo em uma ferramenta ou ambiente que suporte o fluxo do processo, exceto em [22].

A atualização automática do modelo de projeto a partir do modelo de análise sugerido no capítulo 4 na seção referente à atividade Projetar Caso de Uso ficará como trabalho futuro. A proposta do trabalho será identificar um meio de atualizar os diagramas de seqüência e de classes a partir do mapeamento de classes de análise em classes de projeto como sugerido no capítulo 4 na seção referente à atividade Projetar Arquitetura.

O modelo e o mapeamento como entradas para a atualização do modelo não são suficientes para atualizarem o modelo de projeto. As entradas carecem de informações sobre como as novas classes de projeto se relacionam com as classes de análise do mapeamento. Tal fato impede a atualização dos diagramas de classes, pois não são conhecidos quais serão os relacionamentos entre as classes de projeto, como também, entre as classes de projeto e as classes de análise remanescentes.

Também não é possível atualizar os diagramas de seqüência, pois não consta no mapeamento quais classes de projeto são responsáveis pelas operações da classe de análise mapeada e a ordem das mensagens a serem trocadas no diagrama de seqüência.

O desafio do trabalho futuro será encontrar um formato para as informações requeridas citadas acima. Uma sugestão é estabelecer um formato de entrada que possua informações sobre os padrões de projeto adotados no projeto do sistema. Um modo de representar tal entrada seria modelar o padrão através de UML e referenciá-lo nas entradas do mapeamento de classes onde o padrão se aplica.

Como exemplo de utilização podemos citar o padrão *Facade*[18] e o padrão *Persistent Data Collections* (PDC)[19] aplicado na transformação das classes de cadastro utilizado no teste do RADAS exibido no mapeamento da figura 6.6. O trabalho[21] propõe a criação de um repositório de padrões de projetos comumente utilizados que possam ser aplicados ao processo de desenvolvimento do RUP.

8 Referências

1. ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023: informação e documentação: referências**: elaboração. Rio de Janeiro, 2002.

2. BOOCH G. Leaving Kansas. **IEEE Software,** v.15, n.1, 1998. p. 32-35.

3. **Introdução ao fluxo de A&P do RUP**. Disponível em: <http://www.cin.ufpe.br/~if718> Acesso em: 10 set. 2007.

4. **Jude System Design Tool.** Disponível em: < http://jude.change-vision.com/jude-web/index.html> Acesso em: 20 out. 2007.

5. **Jude Users Community.** Disponível em: <http://jude-users.com/en/>Acesso em: 20 out. 2007.

6. KRUCHTEN, Philippe. **Introdução ao RUP: rational Unified Process**. 2ª ed. Rio de Janeiro: Editora Ciência Moderna Ltda, 2003.

7. **Object Management Group**. Disponível em: <http://www.omg.org> Acesso em: 20 set. 2007.

8. OSTERWEIL L. Software processes are software too. **Proceedings of the Ninth International Conference on Software Engineering**, 1987. p.2-13.

9. PENDER, Tom. **UML: A Bíblia**. Rio de Janeiro: Campus, 2004.

10. **Projetar Arquitetura: classes de projetos, subsistemas e recuso.** Disponível em: <http://www.cin.ufpe.br/~if718> Acesso em: 10 set. 2007.

11. **Projetar casos de uso**. Disponível em: <http://www.cin.ufpe.br/~if718> Acesso em: 10 set. 2007.

12. **Rational Unified Process: Visão geral**. Disponível em: <http://www.wthreex.com/rup/> Acesso em: 20 set. 2007.

13. **Tigris.org Open Source Software Engineering Tools.** Disponível em: <http://argouml.tigris.org/> Acesso em: 20 out. 2007.

14. **Unified Modeling Language**. Disponível em: <http://www.uml.org> Acesso em: 20 set. 2007.

15. **Wikipedia: The Free Encyclopedia (List of UML Tools).** Disponível em: < http://en.wikipedia.org/wiki/List_of_UML_tools> Acesso em: 20 set. 2007.

16. **Wikipedia: The Free Encyclopedia (Sparx Enterprise Architect).** Disponível em: <http://en.wikipedia.org/wiki/Sparx_Enterprise_Architect > Acesso em: 20 set. 2007.

17. SOMMERVILLE, Ian. **Software Engineering**. 8ª edição. Harlow: Editora Addison Wesley, 2007.

18. GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software:** Addison-Wesley, 1994.416p.

19. MASSONI, T.;ALVES, V.;SOARES, S.;BORBA, P. **PDC:*The persistent data collections pattern***. In: FIRSTLATIN AMERICAN CONFERENCE ON PATTERN LANGUAGES OF PROGRAMMING, 2001, Rio de Janeiro,p.1-16.

20. FILHO, L.; ANNA, N.; JÚNIOR, M.; GENVIGIR, E. **Análise e Modelagem para Automação de Processos de Gerenciamento de Projetos**.
Disponível em: <http://hermes2.dpi.inpe.br:1905/col/lac.inpe.br/lucio/2002/11.08.15.13/doc/BorregoFilhoLF.pdf>
Acesso em: 20 set. 2007.

21. MARINHO, F.; SANTOS, M.; PINTO, R.; ANDRADE, R. **Uma Proposta de um Repositório de Padrões de Software Integrado ao RUP.**
Disponível em: <http://www.cin.ufpe.br/~sugarloafplop/articles/spa/spa_RepositorioPadroesSoftwareRUP.pdf>
Acesso em: 20 set. 2007.

22. CUNHA, P.; CARNEIRO, R.; PIRES, C.; BANDEIRA, L.; DONEGAN, P.; MAIA, C.; MATOS, C. **Automação de Testes Sistêmicos Funcionais**.
Disponível em: <http://www.mct.gov.br/upd_blob/0005/5018.pdf>.
Acesso em: 20 set. 2007.

9 Apêndice Casos de Uso Relacionados

Caso de Uso: Efetuar Login
Descrição: Este caso de uso é responsável por autenticar um usuário do sistema.
Pré-condição: Nenhuma.
Pós-condição: Um usuário válido é logado e sua sessão é registrada no sistema.
Fluxo Principal: 4. O cliente informa login e senha. 5. O sistema verifica se o login e a senha são válidos (verifica-se se o login e senha pertencem a uma conta). 6. O sistema registra o início de uma sessão de uso.
Fluxos Secundários: No passo 2, se o login e a senha forem inválidos, o sistema exibe uma mensagem e volta ao passo 1.

Caso de Uso: Realizar Pagamento do Qualiti Card
Descrição: Este caso de uso é responsável por realizar o pagamento do Qualiti Card com a operadora de cartão de crédito. Cada cliente possui apenas um cartão como titular, estando vinculado a apenas uma operadora.
Pré-condição: O cliente deve estar conectado ao sistema (ter efetuado o login).
Pós-condição: O valor do pagamento é debitado da conta do cliente, o pagamento é enviado à operadora do cartão de crédito e a transação é registrada no sistema.
Fluxo Principal: 1. O cliente informa os dados necessários para efetuar o pagamento do cartão: o código de barras da fatura que deseja efetuar o pagamento e o valor que deseja pagar. 2. O sistema recupera a conta bancária do cliente logado. 3. O sistema verifica se o saldo da conta do cliente é suficiente para realizar o pagamento. 4. O sistema debita da conta do cliente. 5. O sistema envia o pagamento à operadora de cartão de crédito. 6. O sistema registra a transação de pagamento e emite um comprovante da mesma para o usuário. A transação registrada contém os dados da conta do cliente, o código de barras da fatura, data, hora e valor do pagamento.
Fluxos Secundários: No passo 3, se o saldo disponível na conta do cliente for menor que o valor do pagamento, o sistema informa que o saldo é insuficiente e retorna para o passo 1. No passo 5, se a operadora de cartão de crédito estiver inativa, o sistema exibe uma mensagem e cancela a operação. Em qualquer momento o usuário pode cancelar a operação.

Caso de Uso: Realizar DOC

Descrição:

Este caso de uso é responsável por realizar a transferência de valores entre uma conta deste banco para uma conta de um outro banco. A transferência pode ocorrer entre contas com mesmo CPF ou CPFs distintos..

Pré-condição:

O cliente deve estar conectado ao sistema (ter efetuado o login).

Pós-condição:

O valor da transferência foi debitado da conta do cliente, o DOC foi enviado à operadora de DOC e a transação foi registrada.

Fluxo Principal:

1. O cliente informa os dados necessários para a realização do DOC: banco, agência e conta destino, CPF do favorecido e valor do DOC.
2. O sistema recupera a conta bancária do cliente logado.
3. O sistema verifica se o saldo da conta do cliente é suficiente para a realização do DOC.
4. O sistema envia o DOC à operadora.
5. Debita-se o valor da conta.
6. O QIB registra a ocorrência desta transação (um DOC).
7. Emite-se um comprovante da mesma para o usuário, contendo os dados da conta de origem e destino, assim como a data e valor do DOC.

Fluxos Secundários:

No passo 3, se o saldo disponível na conta do usuário for menor que o valor do DOC, o sistema informa que o saldo é insuficiente e retorna ao passo 1 do fluxo principal de eventos.

No passo 4, se a operadora de DOC estiver inativa ou se ocorrer algum erro de comunicação que impeça a efetivação da transação, o sistema emite uma mensagem para o cliente e aborta a transação.

Em qualquer momento o usuário pode cancelar a operação.

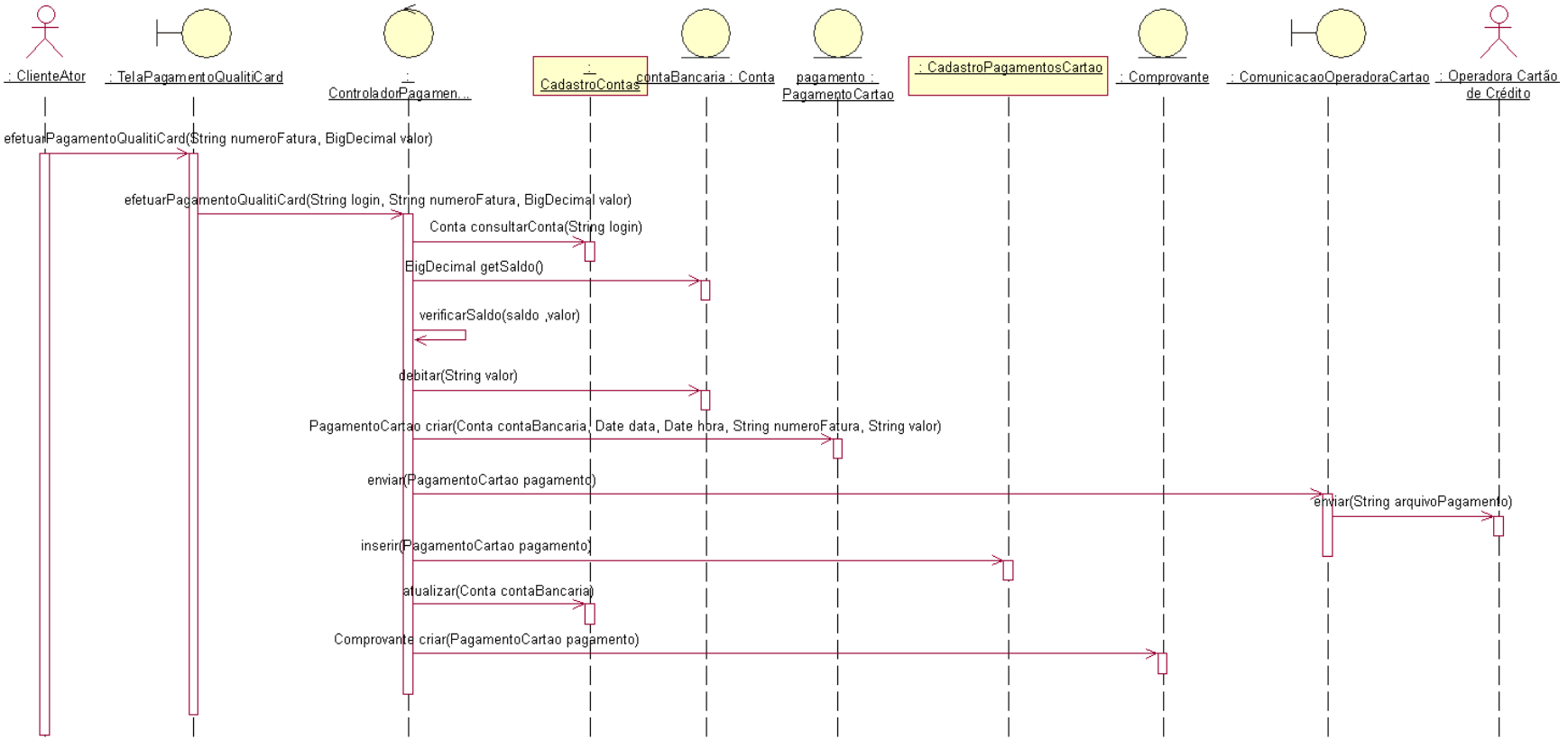


Diagrama de Seqüência do Caso de Uso Efetuar Pagamento do Qualiti Card

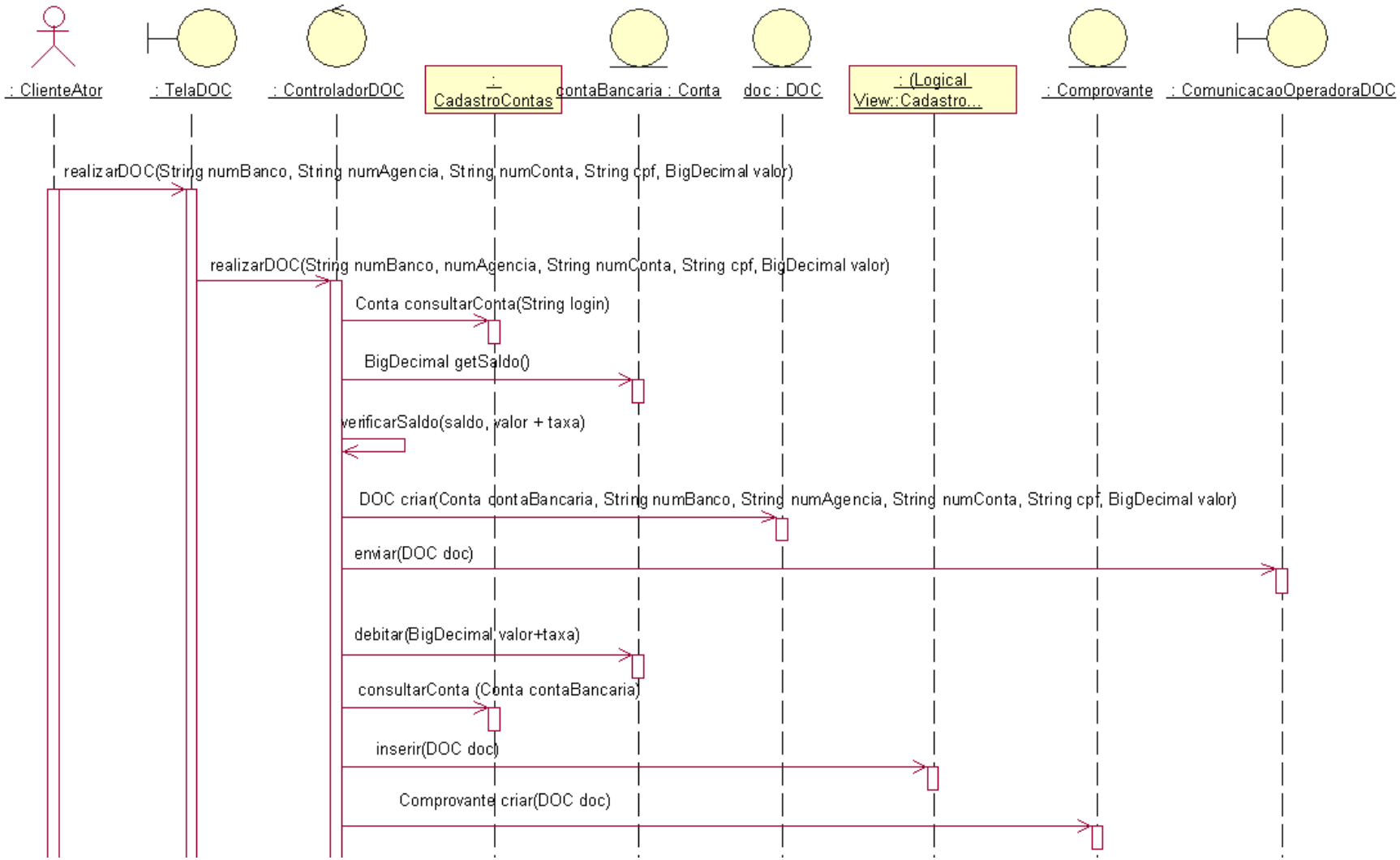


Diagrama de Seqüência do Caso de Uso Realizar DOC.

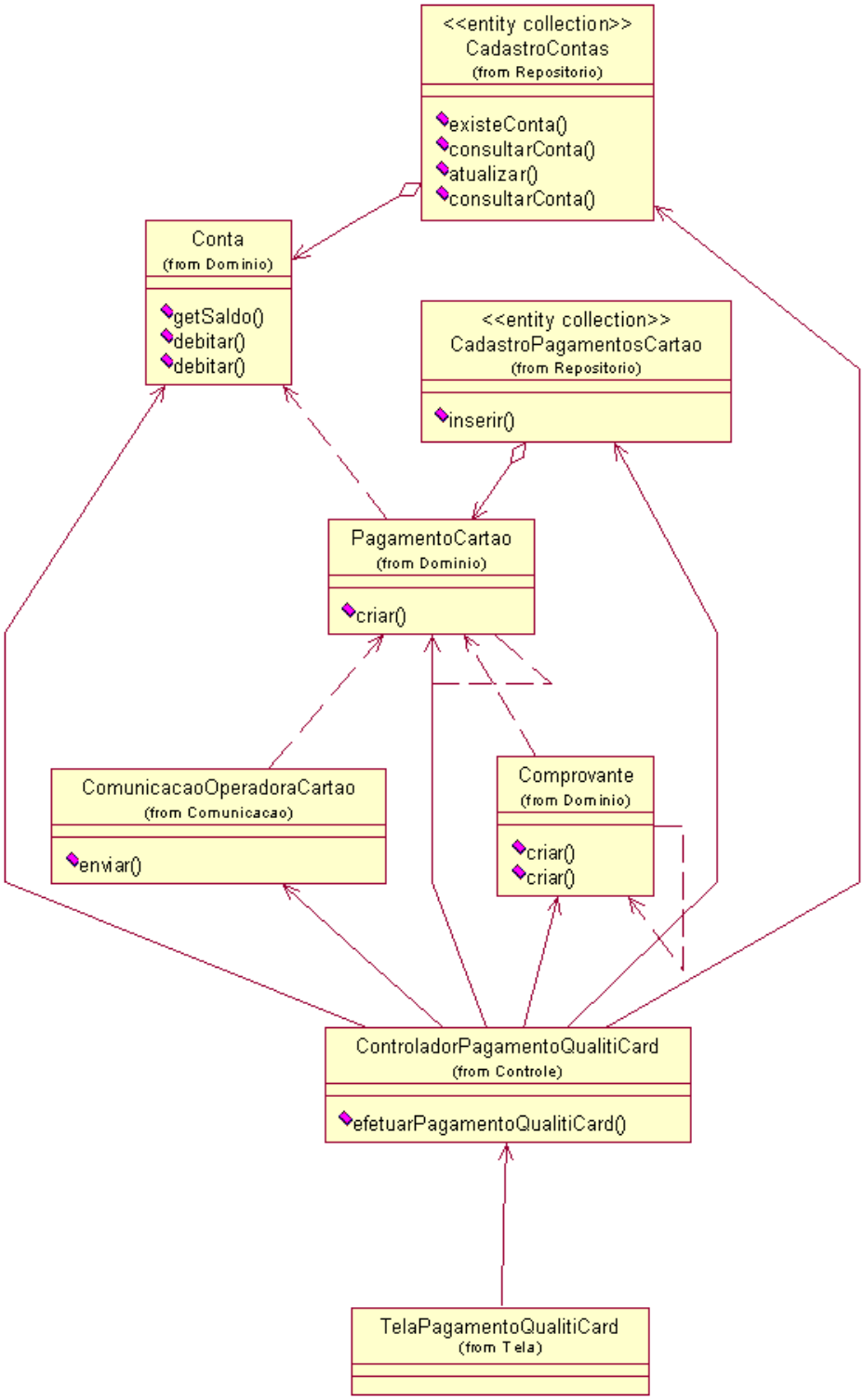


Diagrama de Classes do Caso de Uso Efetuar Pagamento do Qualiti Card

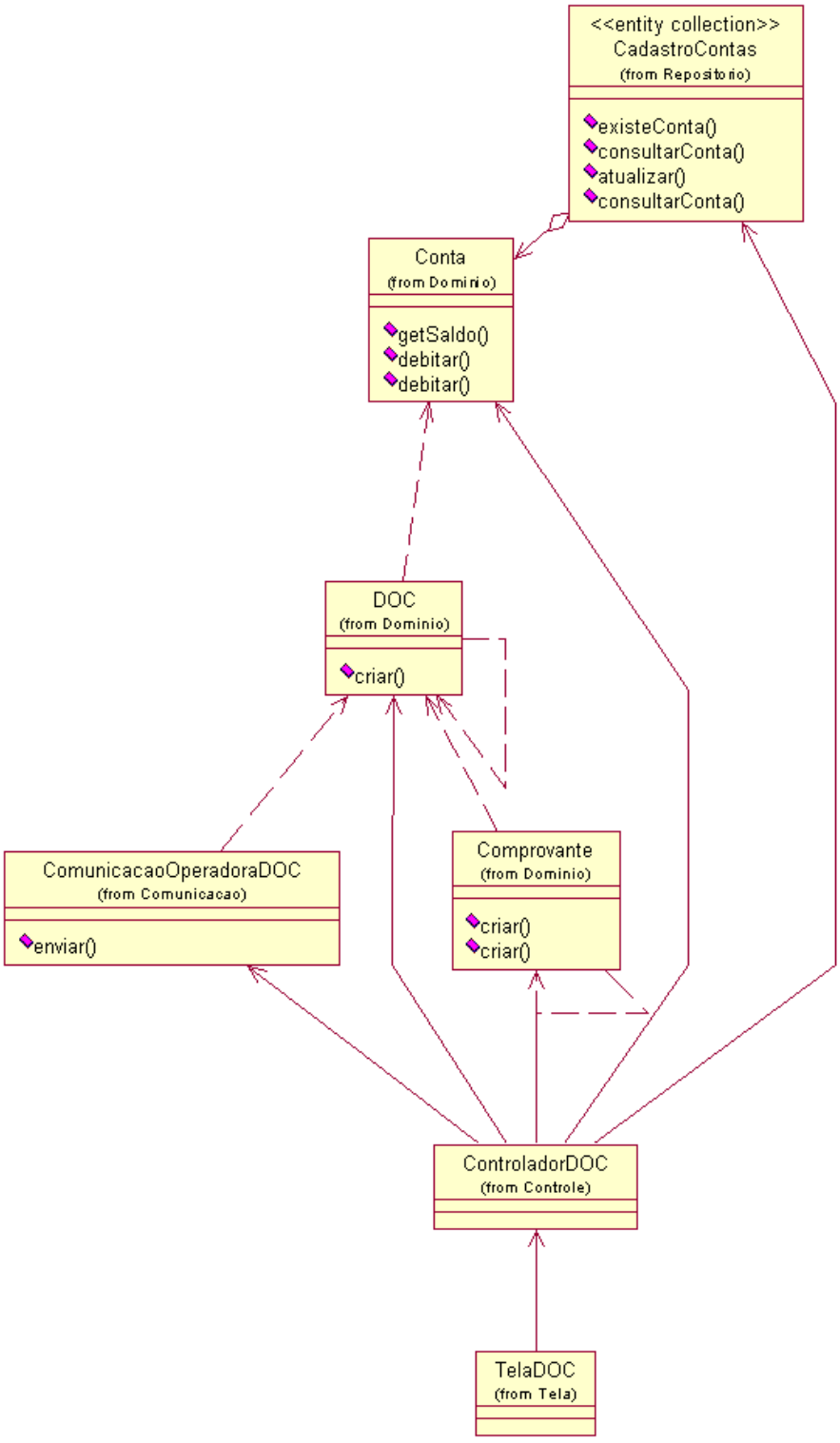


Diagrama de Classes do Caso de Uso Realizar Caso de Uso