UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO
n° 0021/2010

Projeto com UML e implementação de serviços em SOA

Thaíssa Diirr
Flávio Faria
Leonardo Azevedo
Flávia Santoro
Fernanda Baião

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

Projeto de Pesquisa

Grupo de Pesquisa Participante

Relatórios Técnicos do DIA/UNIRIO, No. 0021/2010
Editor: Prof. Sean W. M. Siqueira Dezembro, 2010

Projeto com UML e Implementação de serviços em SOA

Thaíssa Diirr, Flávio Faria, Leonardo Azevedo, Flávia Santoro, Fernanda Baião

Núcleo de Pesquisa e Prática em Tecnologia (NP2Tec)
Departamento de Informática Aplicada (DIA) – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{thaissa.medeiros, flavio.faria, azevedo, flavia.santoro, fernanda.baiao}@uniriotec.br

Abstract. The management of service life-cycle introduces the need for an approach different from those used for traditional system development, due to the introduction of new roles and tasks of architectural development. In Souza *et al*. [2010], the identification and analysis services from a business process model were realized, following heuristics proposed by Azevedo *et al*. [2009a, 2009b, 2009c, 2009d, 2009e]. After analysis, the next phases of life cycle services are the design and implementation, which are the focus of this work. Thus, this work represents a sequel of the case study done by Souza et al. [2010]. All steps carried out were detailed; the services have been designed and implemented.

Keywords: Service modeling, service specification, service design, service implementation.

Resumo. O gerenciamento do ciclo de vida de serviços introduz a necessidade de uma abordagem diferenciada das utilizadas para os sistemas tradicionais, devido à introdução de novos papéis arquiteturais e tarefas de desenvolvimento. Em Souza *et al*. [2010], foram realizadas a identificação e análise de serviços a partir de um modelo de processos de negócio, seguindo heurísticas propostas por Azevedo *et al*.[2009a, 2009b, 2009c, 2009d, 2009e]. Após a análise, as próximas fases do ciclo de vida de serviços são o projeto e a implementação, as quais são o foco deste trabalho. Logo, este trabalho corresponde a uma continuação do estudo de caso realizado por Souza *et al.* [2010]. Todos os passos realizados foram detalhados, os serviços foram projetados e implementados.

Palavras-chave: Modelagem de serviços, especificação de serviços, projeto de serviços, implementação de serviços.

Sumário

Figuras

Tabelas

1 Introdução

Os sistemas orientados a serviços são desenvolvidos diferentemente dos sistemas tradicionais, através da composição de serviços compartilhados entre as organizações. SOA busca maior flexibilidade, agilidade e reuso de aplicações. O desenvolvimento de serviços requer preocupações adicionais ao desenvolvimento de software tradicional, como cooperação entre os papéis arquiteturais de SOA (provedores, consumidores e brokers), bom entendimento dos modelos de negócio e relacionamento entre os parceiros de negócio, como lidar com requisitos conflitantes, como alinhar os requisitos de negócio com soluções de TI, como distribuir serviços através das fronteiras da organização, como produzir serviços de forma a possibilitar o reuso dos mesmos etc. Dessa forma, o ciclo de vida de um software tradicional não se aplica diretamente a uma abordagem SOA. Vale enfatizar os novos papéis arquiteturais e tarefas de desenvolvimento, além de priorizar uma visão holística sobre as demandas de TI. Assim, é necessária uma abordagem diferenciada para o gerenciamento do ciclo de vida de serviços [Gu e Lago, 2007].

Em Azevedo *et al*.[2009a, 2009b, 2009c, 2009d, 2009e] são propostas heurísticas para identificação e análise de serviços em uma abordagem SOA, a partir de modelos de processos de negócio. Após a identificação dos serviços candidatos e a análise dos serviços, os serviços analisados devem ser projetados e implementados .

O objetivo deste documento é apresentar um conjunto de passos seguidos para especificar os serviços e implementá-los fisicamente. Este trabalho é uma continuação do estudo de caso apresentado em Souza *et al*. [2010] onde foram realizadas a identificação e análise de serviços a partir de um modelo de processos de negócio.

Esse relatório está organizado em 5 capítulos, sendo o capítulo 1 a presente introdução. O capítulo 2 apresenta passos seguidos em um exemplo de projeto de serviços e o capítulo 3 a implementação dos mesmos. O capítulo 4 apresenta nossas conclusões e o capítulo 5 apresenta as referências bibliográficas utilizadas para elaboração da nossa proposta.

2 Projeto de serviços

Nesta seção apresentamos o projeto dos serviços candidatos mais indicados a serem desenvolvidos fisicamente a partir do que foi identificado no estudo de caso apresentado por Souza *et al.*[2010].

Erl [2005] define um serviço candidato como sendo uma abstração (não implementada) de serviço a qual, durante a fase de projeto em um modelo de ciclo de vida de serviço, pode ser escolhida para ser implementada como um serviço ou como uma funcionalidade de uma aplicação.

Os serviços candidatos são classificados como serviços candidatos de dados (apenas executam operações de CRUD) ou serviços candidatos de lógica (encapsulam uma regra de negócio). No entanto, nos casos em que há acesso a dados e para os casos em que fique difícil de definir se a regra de negócio está mais ligada ao negócio ou se está mais ligada ao dado, o serviço pode ser classificado como serviço de lógica e executar operação CRUD [Azevedo *et al*., 2009a].

2.1 Serviços candidatos resultantes da fase identificação e análise

Souza *et al.* [2010] realizaram a identificação e análise de serviços candidatos a partir da modelagem de processos de negócio. O modelo de processos utilizado como insumo para o estudo de caso é apresentado em Dirr *et al.* [2010]. Nesse estudo de caso, uma lista de serviços candidatos foi identificada e, após a realização de operações sobre eles, os serviços mais indicados a se tornarem físicos foram obtidos.

Os próximos passos em um ciclo de vida de desenvolvimento de serviços são o projeto e a implementação destes serviços, que são tratados neste trabalho. A partir da lista de serviços candidatos a se tornarem físicos, os serviços de dados foram separados dos serviços de lógica em projetos diferentes. O agrupamento dos serviços de dados foi ajustado unindo-se serviços cujas funcionalidades manipulavam entidades em comum. Neste caso, o serviço *Tratar créditos e taxas* corresponde à junção dos serviços *Tratar créditos concedidos*, *Tratar taxa de juros*, *Determinar taxa de juros* e *Valor limite máximo para a taxa de juros* (Tabela 1). A Tabela 1 apresenta os serviços candidatos identificados por Souza *et al.* [2010] incluindo o nome do serviço resultante do agrupamento realizado, o nome do serviço candidato, o objeto do modelo de processos de negócio a partir do qual o serviço foi identificado e a descrição do serviço candidato, a qual corresponde diretamente ao objeto do modelo de processos. Detalhes dos objetos do modelo de processo e das notações utilizadas podem ser obtidos em Dirr *et al.* [2010].

Tabela 1 - Serviços a serem implementados fisicamente

Nome do serviço	Serviços candidatos que o compõem	Objeto do modelo de processo a partir do qual foi identificado	Descrição do objeto
Tratar créditos e taxas	Tratar créditos concedidos	Cluster[1] "Créditos concedidos"	Representa as informações dos créditos concedidos aos clientes, contendo para cada concessão de crédito: - código da proposta de crédito, - CPF do cliente, - valor total concedido, - número de parcelas, - taxa de juros, - situação de crédito (comprometido, cancelado pelo analista de crédito, aprovado pelo analista de crédito, rejeitado pelo cliente, aprovado pelo cliente), - para cada parcela: - data a ser realizado o pagamento, - valor a ser pago, - juros correspondentes à multa por atraso, - valor pago, - data em que o pagamento foi realizado.
	Tratar taxa de juros	Cluster "Taxa de juros"	Representa as taxas de juros cadastradas, contendo para cada faixa de valor de financiamento: - valor total mínimo de financiamento, - valor total máximo de financiamento, - número de parcelas mínimo, - número de parcelas máximo, - taxa de juro.
	Determinar taxa de juros	Regra de negócio "Determinar taxa de juros"	A taxa de juros deve ser determinada de acordo com o valor total solicitado para financiamento e o número de parcelas. Isto é feito através da comparação do valor total e do número de parcelas com limites estabelecidos nas taxas de juros cadastradas.
	Valor limite máximo para a taxa de juros	Regra de negócio "Valor limite máximo para a taxa de juros"	O valor limite máximo para a taxa de juros é de 12%.

[1] Representa um conjunto de informações (estruturadas ou não) gerado ou consumido durante a execução do processo.

Nome do serviço	Serviços candidatos que o compõem	Objeto do modelo de processo a partir do qual foi identificado	Descrição do objeto
Tratar proposta de contrato	Tratar proposta de contrato	Cluster "Proposta de contrato"	Representa a proposta de contrato, contendo: - código da proposta de crédito, - nome do cliente, - CPF, - endereço do cliente, - telefone do cliente, - lista de peças, - valor a ser financiado, - número de parcelas, - taxa de juros, - valor taxas do contrato, - para cada parcela: - data a ser realizado o pagamento, - valor a ser pago, - juros correspondentes à multa por atraso, -situação.
Tratar proposta de crédito	Tratar proposta de crédito	Cluster "Proposta de crédito"	Representa as informações do solicitante e dos produtos a serem financiados necessárias para solicitação de crédito, contendo: - código da proposta de crédito, - nome do solicitante, - CPF, - identidade, - telefone do solicitante, - endereço do solicitante, - lista de peças, - valor a ser financiado, - número de parcelas, - renda, - data de cadastro, - responsável pelo cadastro, - resultado da verificação de crédito (aprovado ou reprovado).
Verificar cadastro do cliente	Verificar cadastro cliente desatualizado	Regra de negócio "Cadastro de cliente desatualizado"	O cadastro do cliente está desatualizado se o telefone, endereço e renda informados na proposta de crédito forem diferentes das informações do cliente existentes na base de dados.
	Verificar cliente novo	Regra de negócio "Cliente novo"	Um cliente é novo se não existir cadastro de cliente com mesmo CPF que o informado na proposta de crédito.
Tratar cliente	Tratar cadastro do cliente	Cluster "Cadastro do cliente"	Representa as informações do cadastro do cliente, contendo: - nome, - CPF, - identidade, - telefone, - endereço, - CEP, - renda, - data da última atualização, - data de cadastro.
Verificar limite de crédito	Verificar limite de crédito	Regra de negócio "Verificar limite de crédito"	O limite de crédito do cliente é igual a 20% da sua renda menos o valor mensal referentes às parcelas ainda em aberto dos créditos anteriormente concedidos ao cliente.
	Aprovar crédito	Regra de negócio "Aprovação de crédito"	Se limite de crédito do cliente é maior ou igual ao valor da parcela a ser pago para cada parcela da proposta de crédito, então o crédito é aprovado, senão, o crédito não é aprovado.
Calcular taxas do contrato	Enviar mensagens para calcular taxas do contrato	Atividade de múltiplas instâncias "Calcular taxas do contrato"	Este serviço emite mensagem para atividade de múltipla instância Calcular taxas do contrato
	Calcular taxas do contrato_RqN	Requisito de negócio "Calcular taxas do contrato"	O sistema Crédito Direto calcula as taxas do contrato de acordo com as regras de negócio “Cálculo da taxa de alteração do contrato”, “Cálculo da taxa de entrega”, “Cálculo da taxa de impressão” e “Cálculo do IOF”.
	Calcular taxa de impressão	Regra de negócio "Cálculo da taxa de impressão"	A taxa de impressão é de R$3,17 por contrato impresso.
	Calcular taxa de entrega	Regra de negócio "Cálculo da taxa de entrega"	A taxa de entrega do contrato é calculada da seguinte forma: km de distância X R$0,01. Onde km de distância significa a quantidade de quilômetros que o CEP do destinatário está de distância.
	Calcular IOF	Regra de negócio "Cálculo do IOF"	A valor do IOF é de 0,0014% do montante que consta no pedido de crédito.
	Calcular taxa de alteração do contrato	Regra de negócio "Cálculo da taxa de alteração do contrato"	O valor da taxa de alteração do contrato é de R$0,33 x o número de alterações feitas.
	Consolidar mensagens de calcular taxas do contrato	Atividade de múltiplas instâncias "Calcular taxas do contrato"	Este serviço consolida o produto da atividade de múltipla instância Calcular taxas do contrato
	Calcular taxas do contrato	Atividade de múltiplas instâncias "Calcular taxas do contrato"	O sistema Crédito Direto calcula as taxas do contrato. As informações necessárias são: créditos concedidos e cadastro do cliente. A informação gerada é: taxas do contrato.
Tratar taxa de contrato	Tratar taxas do contrato	Cluster "Taxas do contrato"	Representa o valor calculado das taxas do contrato, contendo: - código da proposta de crédito, - taxa de IOF, - taxa de impressão, - taxa de entrega, - taxa de alteração do contrato.

Nome do serviço	Serviços candidatos que o compõem	Objeto do modelo de processo a partir do qual foi identificado	Descrição do objeto
Verificar contrato de risco	Identificar contrato de risco	Regra de negócio "Identificação de contrato de risco"	Um contrato é de risco se o valor mensal a ser pago pelo cliente é maior do que 90% do seu limite de crédito e o cliente possui mais de dois financiamentos em andamento com data de conclusão maior do que 6 meses.
	Ajustar proposta de crédito	Regra de negócio "Ajuste da proposta de crédito"	A proposta de crédito deve ser alterada se o analista considerar que é possível reduzir o risco da proposta de contrato com a redução do valor financiado. Caso contrário, o contrato deve ser cancelado.
	Alterar proposta de crédito	Regra de negócio "Alteração da proposta de crédito"	Quando for necessário reduzir o risco da proposta de crédito, o valor a ser financiado, o número de parcelas e a taxa de juros devem ser alterados.
Gerar proposta de contrato	Gerar proposta de contrato	Requisito de negócio "Gerar proposta de contrato"	O sistema Crédito Direto deve gerar a proposta de contrato, contendo: - código da proposta de crédito, - nome do cliente, - CPF, - endereço do cliente, - telefone do cliente, - lista de peças, - valor a ser financiado, - número de parcelas, - taxa de juros, - valor das taxas do contrato, - para cada parcela: - data a ser realizado o pagamento, - valor a ser pago, - juros correspondentes à multa por atraso; e notificar o analista de crédito sobre a geração do contrato;

2.2 Modelo Canônico

O modelo de dados canônico é um conjunto de dados comum, independente das aplicações e compartilhado por elas. Assim, as aplicações não precisam negociar em formatos de mensagens entre si, podendo, simplesmente, utilizar o formato de dados canônico existente [Woolf, 2006].

Hewitt [2009] descreve duas possibilidades para definição de esquemas de dados:

- Maximização do reuso através da eliminação de tipos redundantes: São definidos esquemas canônicos para representar os tipos principais e mais gerais e também podem ser definidos esquemas específicos para serviços. Porém, a alteração de um tipo principal gera a necessidade de teste de todos os serviços que utilizam aquele tipo. Apesar disso, é apresentado que, apesar das mudanças nos esquemas canônicos devam ocorrer através de governança SOA, elas são raras, pois as entidades do negócio não mudam sempre.
- Minimização da interdependência: São definidos esquemas de dados separadamente para cada serviço, gerando um contrato totalmente customizado para cada serviço e permitindo que os serviços evoluam independentemente. Porém, isto pode significar muita redundância e necessidade de mudar todos os esquemas se um tipo principal definido diferentemente em cada um for alterado.

Apesar de apresentar estas possibilidades, Hewitt [2009] defende a eliminação de tipos redundantes com o projeto de esquemas para o modelo de dados canônico separadamente dos serviços, além da definição de esquemas genéricos ou específicos como documentos independentes que serão incluídos pelo WSDL. O autor também advoga a reutilização do esquema com o modelo canônico sempre que possível ao definir esquemas de dados para os serviços.

Segundo Erl [2007] a definição de esquemas de dados específicos para determinados serviços pode levar a projetos de esquemas eficientes, de forma a só representarem os dados relevantes às capacidades do serviço. Porém, este abordagem diminui a reutilização e, se os serviços possuírem representações distintas para os mesmos da-

dos e precisarem trocar mensagens, será necessária a transformação de dados em tempo de execução a cada troca de dados, dificultando a interoperabilidade pelo aumento da sobrecarga. Além disso, é necessário definir as lógicas de mapeamento entre os esquemas, além de precisar manter as camadas de transformação.

Assim, Erl [2007] também defende o projeto e definição dos esquemas de dados de forma separada das operações dos serviços que os utilizam, possibilitando a representação dos dados de forma padronizada e sua reutilização em diferentes serviços. O padrão de projeto *Schema Centralization* (centralização de esquema) é recomendado e defende a definição de um esquema oficial para cada conjunto de informações a ser compartilhado pelos serviços. Dessa forma, os contratos de serviço podem compartilhar os esquemas centralizados (não excluindo a possibilidade de utilização de esquemas complementares com tipos de dados específicos), melhorando a eficiência com que ocorre a troca de informações sobre serviços. Porém, a normalização da representação dos dados pode resultar em desafios para gerir os padrões e os próprios esquemas, apesar de ser viável dentro de um ambiente controlado. Esta normalização torna-se mais complexa quando os dados são compartilhados entre organizações, devido à utilização de esquemas industriais ou indisposição das organizações em concordar sobre os mesmos esquemas.

Na Figura 1, é apresentado o modelo canônico definido a partir das entidades identificadas a partir do modelo de processo de negócio "Analisar pedido de crédito". Ao modelar e implementar os serviços, devem ser consideradas as entidades presentes no modelo canônico.

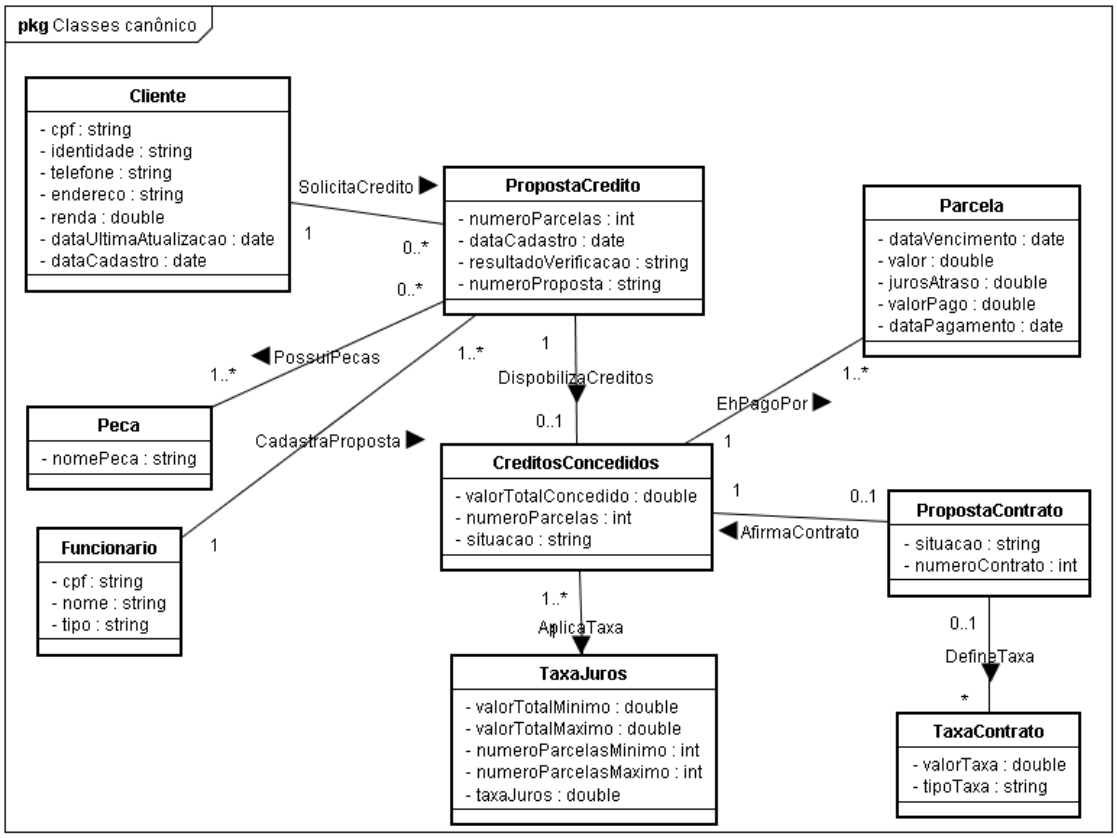


Figura 1 - Modelo canônico

2.3 Decisões sobre operações dos serviços

Como existem serviços (Tabela 1) que abrangem mais de um conceito do negócio, após checar a funcionalidade dos serviços candidatos através da consulta à descrição dos objetos do modelo de processo que os geraram (nas tabelas de identificação dos serviços), foi decidido se cada serviço candidato que compõe o serviço seria uma operação, ou se mais de um serviço candidato comporia uma mesma operação, ou se alguns itens seriam decompostos em mais de uma operação do serviço. Abaixo são listadas as decisões tomadas:

- O serviço "Tratar créditos e taxas" será composto das seguintes operações:
    - Operações CRUD para a entidade Créditos Concedidos (identificadas pelo item ("Tratar créditos concedidos")
    - Operações CRUD para a entidade Taxa de juros (identificadas pelo item "Tratar taxa de juros"). O serviço candidato "Valor limite máximo para taxa de juros" deve ser contemplado nas operações de inserção e alteração da taxa de juros, pois sozinho não caracteriza uma operação
    - Operação "Determinar taxa juros do cliente" (identificada a partir do item de nome correspondente)
- O serviço "Tratar proposta de contrato" é composto de operações CRUD para a entidade Proposta de contrato;
- O serviço "Tratar proposta de crédito" é composto de operações CRUD para a entidade Proposta de crédito;
- O serviço "Manter cadastro do cliente" será composto das seguintes operações:
    - Operações CRUD para a entidade Cliente (identificadas pelo item "Tratar cadastro do cliente")
    - Operação "Verificar cadastro cliente" que representa uma fusão dos serviços candidatos "Verificar cliente novo" e "Verificar cadastro do cliente". Esta operação verifica se o cliente já existe a partir do seu CPF. Se o cliente não estiver cadastrado, ele será incluído. Se o cliente já estiver cadastrado, é verificado se o cadastro está desatualizado e, caso esteja, o registro é alterado.
- O serviço "Verificar limite de crédito" será composto das seguintes operações:
    - Operações "Verificar limite crédito" e "Aprovar crédito" (identificadas a partir dos serviços candidatos de nome correspondente)
- O serviço "Calcular taxas do contrato" será composto das seguintes operações:
    - Operações privadas "Calcular taxa impressão", "Calcular taxa entrega", "Calcular IOF" e "Calcular taxa alteração" (identificadas a partir dos serviços candidatos de nome correspondente)
    - Operação "Calcular taxas contrato" que representa uma fusão dos serviços candidatos "Enviar mensagens para calcular taxas do contrato", "Calcular taxas do contrato_RqN", "Consolidar mensagens de calcular taxas do contrato" e "Calcular taxas do contrato". Todos estes serviços candidatos tratam da realização do cálculo das taxas de contrato, por isso não necessitam ser uma operação cada um. Esta operação realiza o cálculo das taxas invocando as operações privadas do item anterior.

    - Operações CRUD para a entidade Taxas do contrato (identificadas no serviço candidato "Tratar taxas do contrato")
- O serviço "Verificar contrato de risco" será composto das seguintes operações:
    - Operação "Identificar contrato risco" (identificada a partir do serviço candidato de nome correspondente)
    - Operação "Reduzir risco contrato" que representa uma fusão dos itens "Ajustar proposta de crédito" e "Alterar proposta de crédito". Estes dois serviços candidatos descrevem a alteração da proposta com o objetivo de reduzir o risco do contrato, e por isso podem ser unificados em uma só operação.
- O serviço "Gerar proposta de contrato" terá apenas a operação "Gerar proposta contrato" que registra o contrato, notificando a geração.

2.4 Modelagem dos serviços

Com a definição de quais operações cada serviço deveria conter, foi feita a modelagem dos serviços com UML (Unified Modeling Language). Esta modelagem foi baseada na proposta de Rahmani *et al*.[2006], que define a utilização de:

- Classes com estereótipos "ValueObject" (Objetos de valor) para representar as entidades.;
- Interfaces ligadas a estes objetos de valor contendo operações CRUD realizadas sobre eles;
- Uma interface representando o serviço, ligada a uma ou mais interfaces mencionadas no item anterior. Dessa forma, a interface do serviço engloba as interfaces que manipulam os objetos de valor.

Porém, neste trabalho foram consideradas apenas classes para as entidades e uma interface para cada serviço. Assim, a modelagem dos serviços foi realizada de forma que, para cada serviço:

- Foi criada uma interface com o mesmo nome daquele contido na lista de serviços físicos para manipular as entidades correspondentes e disponibilizar as operações do serviço
    - Para o melhor entendimento do que deveria ser realizado pelas operações do serviço, as informações dos elementos do modelo de processo que originaram o serviço foram consultadas.
    - Os parâmetros de entrada e saída do serviço foram obtidos a partir das descrições dos elementos do modelo de processo.
- Estas interfaces receberam uma associação de dependência com as entidades que necessitam manipular.

O Anexo I apresenta os modelos de classe dos serviços. Diagramas de atividades foram criados para as operações dos serviços mais complexas. Estes diagramas foram elaborados a fim de apresentar para o desenvolvedor, em um mesmo local, todas as informações necessárias para a implementação dos serviços. Dessa forma, o desenvol-

vedor não tem que acessar o modelo de processos para realizar a implementação. Isto possibilita o desenvolvimento dos serviços por pessoas que não tenham conhecimento de modelagem de processos. Por outro lado, vale ressaltar que todos os modelos de atividades foram elaborados baseados nas informações do processo, de modo que em um ambiente onde os desenvolvedores são bem treinados em modelagem de processos, a elaboração destes diagramas de atividades não seria necessária. O Anexo II apresenta os diagramas de atividades modelados para os serviços.

O próximo passo após o projeto dos serviços é a implementação dos mesmos cujos detalhes são apresentados na seção 3.

3 Implementação dos Serviços

Esta seção apresenta os detalhes da implementação dos serviços especificados e apresentados na Tabela 1. Os serviços foram implementados utilizando a tecnologia Java Enterprise Edition[2] versão 6.

Esta seção está estruturada da seguinte forma: a seção 3.1 apresenta como as entidades foram mapeadas a partir do modelo canônico. Em seguida, a seção 3.2 apresenta os detalhes de implementação dos serviços, que foram classificados em serviços de lógica e serviços de dados, apresentados nas seções 3.2.1 e 3.2.2 respectivamente. Por fim, a seção 3.3 apresenta a aplicação cliente "Analisar Pedido de Crédito". Esta aplicação implementa os casos de uso descritos no Anexo III, que foram descritos durante a fase de análise deste trabalho. A Figura 2, Figura 3 e Figura 4 apresentam os diagramas de sequência correspondentes a estes casos de uso. Estes diagramas foram elaborados para facilitar o entendimento por parte do programador do fluxo de chamadas de serviços.

[2] http://www.oracle.com/technetwork/java/javaee/tech/index.html

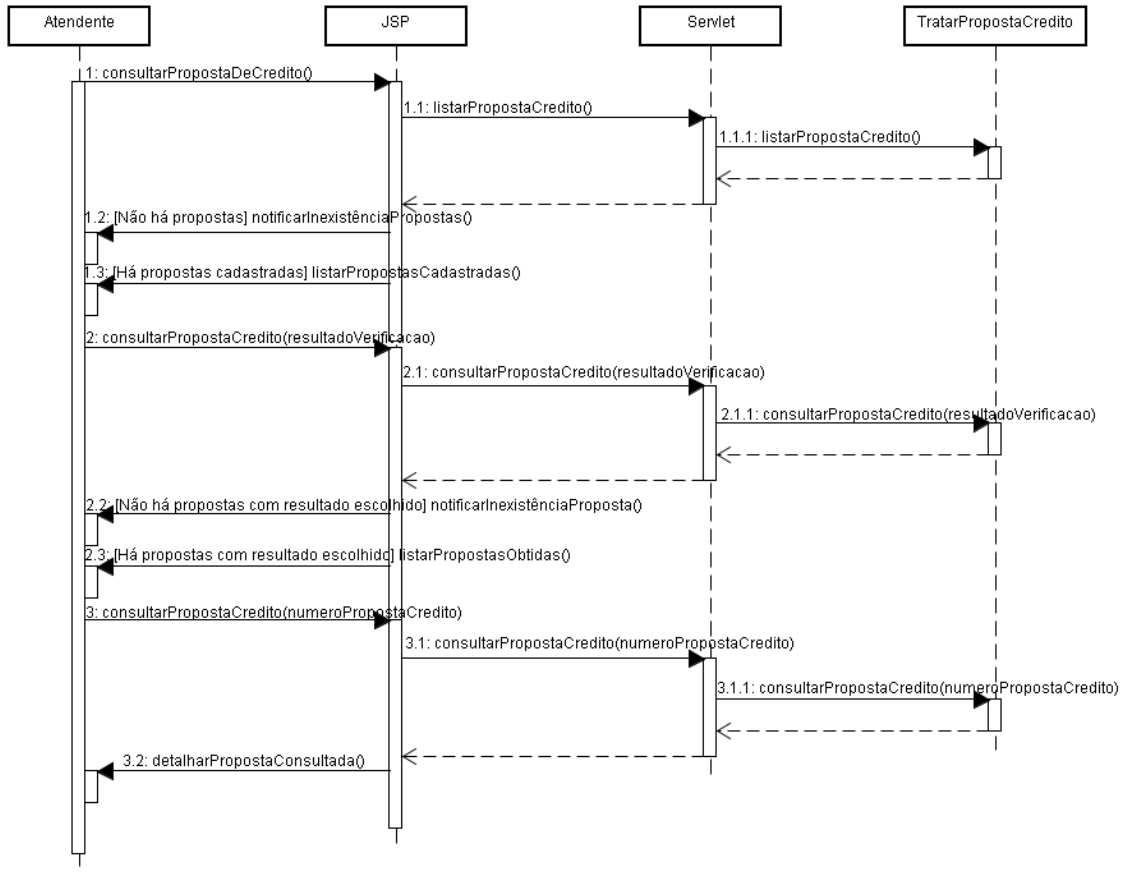


Figura 2 - Diagrama de sequência do caso de uso "Consultar Proposta de crédito"

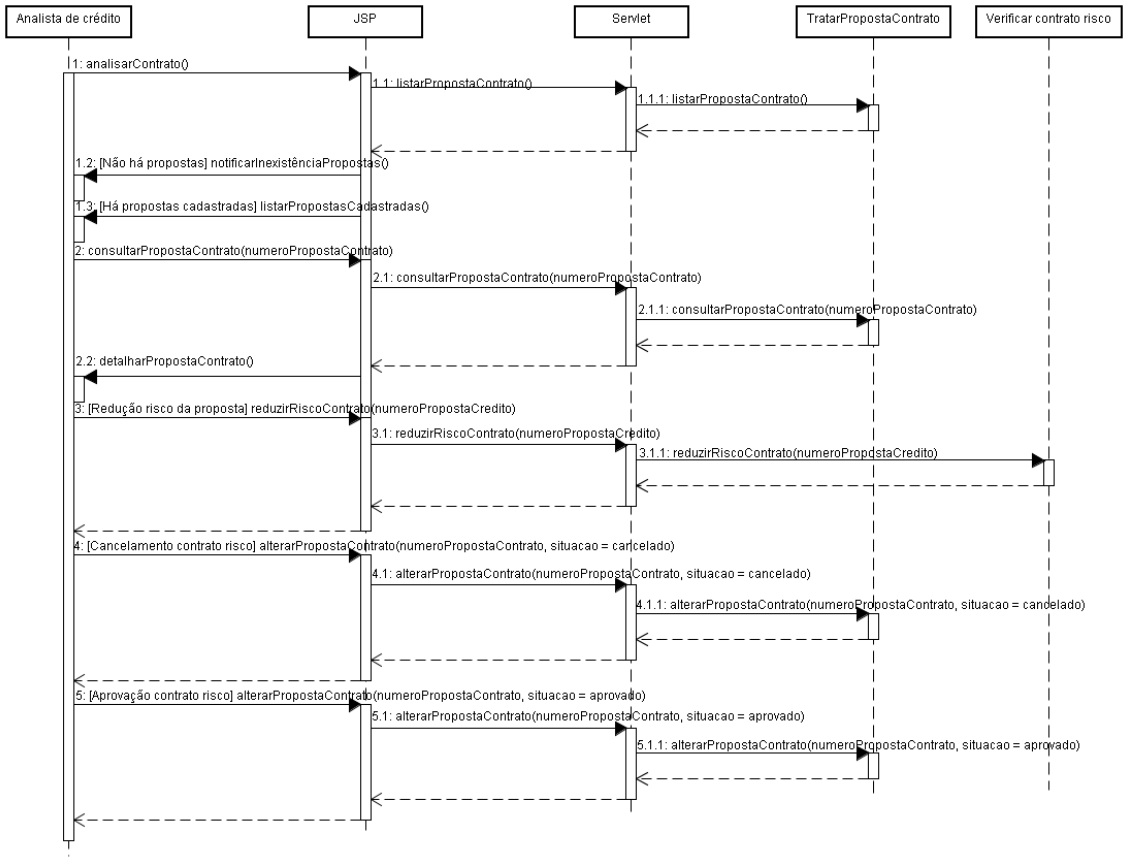


Figura 3 - Diagrama de sequência do caso de uso "Analisar contrato"

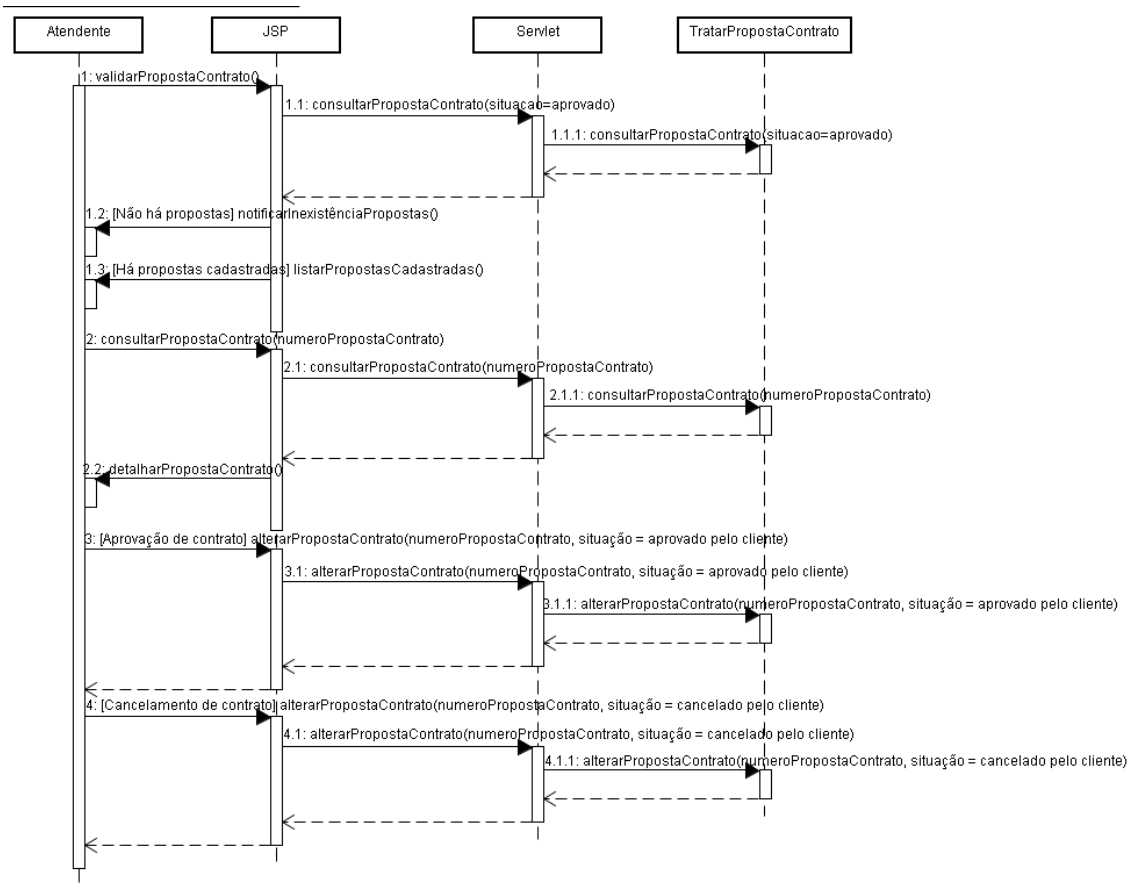


Figura 4 - Diagrama de sequência do caso de uso "Validar proposta de contrato"

3.1 Implementação das entidades do negócio

A partir do modelo canônico (Figura 1), o qual representa a estrutura das entidades do negócio, foi criada uma classe POJO para cada uma das entidades com seus respectivos atributos. Richardson [2006] apresenta que POJO (Plain Old Java Object) é um objeto Java simples que não implementa nenhuma interface especial Java, como aquelas definidas pelo framework EJB [Burke, 2006]. O nome foi definido por Fowler, Rebbecca Parsons e Josh MacKenzie para nomear de uma melhor forma objetos Java regulares[3]. A Figura 5 apresenta um exemplo de classe POJO que representa a entidade cliente que faz parte do domínio do negócio.

[3] http://www.martinfowler.com/bliki/POJO.html

```java
package br.np2tec.soa.entidades;

import java.util.Date;

public class Cliente {

    private String nome;
    private String cpf;
    private String identidade;
    private String telefone;
    private String endereco;
    private Double renda;
    private String email;
    private Date dataUltimaAtualizacao;
    private Date dataCadastro;

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getCpf() {
        return cpf;
    }
```

Figura 5 - Exemplo de classe POJO

As entidades do domínio são manipuladas pelos serviços de negócio. A fim de permitir um maior reuso das classes POJO que representam as entidades do negócio, elas foram implementadas em um único projeto (Figura 6). Este projeto é referenciado pelos projetos correspondentes à implementação dos serviços.

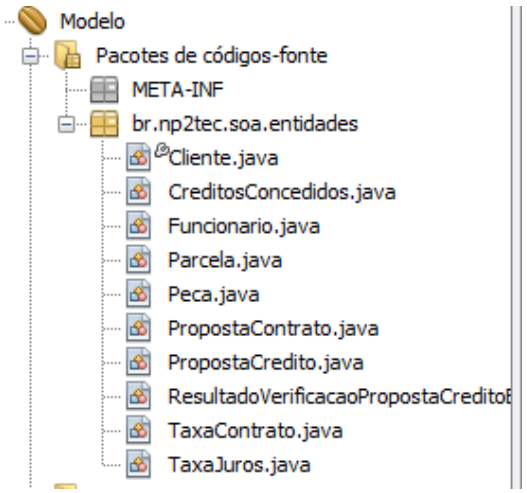


Figura 6 – Projeto que contém as entidades do negócio

3.2 Implementação dos serviços

A classificação dos serviços candidatos em serviços de dados e serviços lógica foi seguida na implementação dos mesmos. Em outras palavras, os serviços de dados e os serviços de lógica foram implementados separadamente.

Erl [2005] apresenta que *web services* é a principal tecnologia para implementação de serviços. Logo, neste trabalho, os serviços foram implementados utilizando esta tecnologia.

As APIs[4] utilizadas para desenvolvimento dos serviços (dados e lógica) foram JAX-WS [Kotamraju, 2009a] e JAX-B [kawaguchi *et al.*, 2010]. Em especial, a API JPA [DeMichiel, 2009] foi utilizada também no desenvolvimento dos serviços de dados.

A especificação JAX-WS [Kotamraju, 2009a] oferece uma maneira simplificada de implementar serviços através do uso de anotações [Mordani, 2009] em objetos POJO. Na Figura 7 apresentamos como a classe *CalcularTaxaContratoService* pode ser exposta como um serviço web com o uso da anotação *@Webservice*. Maiores detalhes sobre implementação de *web services* utilizando podem ser encontrados em [Kotamraju, 2009b].

```
package br.np2tec.soa.servicos;

import javax.jws.WebService;
import javax.xml.ws.WebServiceRef;

@WebService()
public class CalcularTaxaContratoService {
```

Figura 7 - Exemplo serviço web

O ambiente integrado de desenvolvimento (IDE – *Integrated Development Environment*) utilizado para desenvolver os serviços foi o *Netbeans*[5]. Este ambiente permitiu automatizar algumas etapas do processo de desenvolvimento dos serviços como, por exemplo, a geração do WSDL[6] e XSD[7], necessários para o consumo dos serviços pelos clientes. As Figura 8 e Figura 9 apresentam, respectivamente, o WSDL[8] e XSD gerados pelo *Netbeans* correspondente ao serviço *CalcularTaxaContratoService*. Esta forma é mais simples para implementação. No entanto, não é a melhor abordagem. Erl [2005] apresenta que a geração automática desses artefatos pode contribuir para a falta de padronização dos contratos de serviços.

[4] http://java.sun.com/reference/api/

[5] http://netbeans.org/

[6] http://www.w3.org/TR/wsdl

[7] http://www.w3.org/XML/Schema

[8] Parte do arquivo foi suprimida para facilitar sua apresentação.

```
<definitions
    targetNamespace="http://servicos.soa.np2tec.br/"
    name="CalcularTaxaContratoServiceService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://servicos.soa.np2tec.br/"
    schemaLocation="http://localhost:8080/CalcularTaxaContrato/CalcularTaxaContratoServiceService?xsd=1">
            </xsd:import>
        </xsd:schema>
    </types>
    ...
    <portType name="CalcularTaxaContratoService">
        <operation name="incluirTaxaContrato">
            <input message="tns:incluirTaxaContrato"></input>
            <output message="tns:incluirTaxaContratoResponse"></output>
        </operation>
        <operation name="alterarTaxaContrato">
            <input message="tns:alterarTaxaContrato"></input>
            <output message="tns:alterarTaxaContratoResponse"></output>
        </operation>
        <operation name="excluirTaxaContrato">
            <input message="tns:excluirTaxaContrato"></input>
            <output message="tns:excluirTaxaContratoResponse"></output>
        </operation>
        <operation name="listarTaxaContrato">
            <input message="tns:listarTaxaContrato"></input>
            <output message="tns:listarTaxaContratoResponse"></output>
        </operation>
        <operation name="calcularTaxaContrato">
            <input message="tns:calcularTaxaContrato"></input>
            <output message="tns:calcularTaxaContratoResponse"></output>
        </operation>
    </portType>
...
</definitions>
```

Figura 8 - WSDL do serviço CalcularTaxaContratoService

```
<xs:schema xmlns:tns="http://servicos.soa.np2tec.br/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://servicos.soa.np2tec.br/">

    <xs:element name="calcularTaxaContrato" type="tns:calcularTaxaContrato"></xs:element>

    <xs:element name="calcularTaxaContratoResponse" type="tns:calcularTaxaContratoResponse"></xs:element>

    <xs:complexType name="calcularTaxaContrato">
        <xs:sequence>
            <xs:element name="arg0" type="xs:int" minOccurs="0"></xs:element>
            <xs:element name="arg1" type="xs:double" minOccurs="0"></xs:element>
            <xs:element name="arg2" type="xs:double" minOccurs="0"></xs:element>
            <xs:element name="arg3" type="xs:int" minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="calcularTaxaContratoResponse">
        <xs:sequence></xs:sequence>
    </xs:complexType>
</xs:schema>
```

Figura 9 - XSD do serviço CalcularTaxaContratoService

Serviços podem ser construídos a partir da invocação de outros serviços existentes [Papazoglou *et al.*, 2007]. Esta abordagem é conhecida como composição de serviços, apontada como um dos principais aspectos da computação orientada a serviço que contribuí para o reuso dos serviços [Erl, 2005]. Obviamente, ao fazer com que um serviço invoque outro serviço, é criada uma dependência entre os serviços envolvidos na composição. Uma das formas para resolver a dependência entre serviços utilizando a tecnologia Java é utilizar a anotação *@WebServiceRef*.

A Figura 10 apresenta um exemplo de um serviço que invoca outro serviço. Neste caso, o serviço *VerficarContratoRiscoService* tem uma dependência para o serviço *Tra-*

tarPropostaCreditoService[9], representada na anotação *@WebServiceRef*. O atributo *wsdlLocation* contém o endereço para o serviço que será invocado. É importante ressaltar que o uso dessa anotação implica na publicação do serviço em um servidor de aplicação que seja capaz de interpretá-la através do uso da técnica conhecida como injeção de dependência [Fowler, 2004].

```
package br.np2tec.soa.servicos;

import java.util.Iterator;
import java.util.List;
import javax.jws.WebService;
import javax.xml.ws.WebServiceRef;

@WebService()
public class VerificarContratoRiscoService {

    @WebServiceRef(wsdlLocation = ""
    + "http://localhost:8080/TratarPropostaCreditoDataServiceService/TratarPropostaCreditoDataService?WSDL")
    TratarPropostaCreditoDataServiceService tratarPropostaCreditoDataService;
```

Figura 10 - Exemplo de dependência entre serviços

A Figura 11 apresenta um exemplo de invocação do serviço *TratarPropostaCredito* para alterar a proposta de crédito. Inicialmente, é obtido o *port type* do serviço e, em seguida, é invocada a operação *alterarPropostaCredito*.

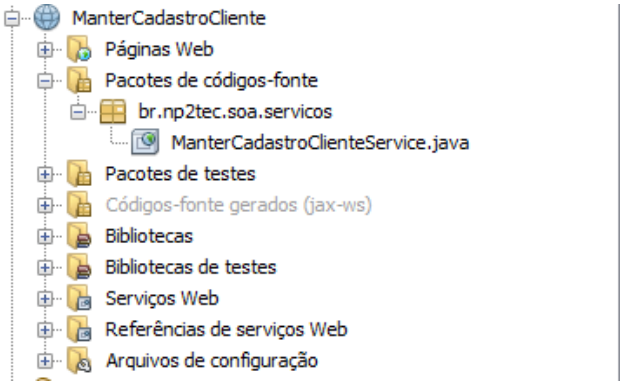
```
public void reduzirRiscoContrato(PropostaCredito propostaCredito) {

    TratarPropostaCreditoDataService port = tratarPropostaCreditoDataService.getTratarPropostaCreditoDataServicePort();
    port.alterarPropostaCredito(propostaCredito.getValorFinanciado(), propostaCredito.getNumeroParcelas(),
            propostaCredito.getResultadoVerificacao(), propostaCredito.getPecas(), propostaCredito.getNumeroProposta());
```

Figura 11 - Exemplo invocação operação através *por type* injetado pela anotação *@WebServiceRef*

3.2.1 Implementação dos serviços de lógica

Conforme apresentado anteriormente, os serviços de lógica encapsulam as regras de negócio. Cada serviço de lógica foi implementado em um projeto distinto, visando uma melhor organização e reuso dos serviços. Além disso, essa abordagem faz com que cada serviço seja empacotado e distribuído separadamente. A Figura 12 apresenta um exemplo de projeto que implementa o serviço de lógica *ManterCadastroClienteService*.



[9] O *Netbeans* inclui a terminação *Service* durante a geração automática do WSDL e das classes *stubs*. Como as classes que implementam os serviços possuem o padrão <*nome_do_serviço*>*Service*, isso faz com que as referências para os serviços sejam alteradas para <*nome_do_serviço*>*ServiceService*.

Figura 12 - Estrutura do projeto de lógica *ManterCadastroClienteService*

A Figura 13 apresenta o serviço de lógica *ManterCadastroClienteService* e um exemplo de lógica implementada através da operação *verificarCadastroCliente.* A operação *verificarCadastroCliente* inclui um cliente (se ele não existir) ou atualiza os dados cadastrais (se o cliente já existir mas os dados estiverem desatualizados).

```
package br.np2tec.soa.servicos;

import br.np2tec.soa.servico.Cliente;
import br.np2tec.soa.servico.TratarClienteDataServiceService;
import javax.jws.WebService;
import javax.xml.ws.WebServiceRef;

@WebService()
public class ManterCadastroClienteService {

    @WebServiceRef(wsdlLocation = ""
    + "http://localhost:8080/TratarClienteDataServiceService/TratarClienteDataService?WSDL")
    TratarClienteDataServiceService clienteDataService;

    public void verificarCadastroCliente(String cpf, String endereco, Double renda,
                String telefone, String identidade, String nome) throws Exception {

        Cliente cliente = clienteDataService.getTratarClienteDataServicePort().consultar(cpf);

        if (cliente == null) {

            clienteDataService.getTratarClienteDataServicePort().incluirCliente(cpf, endereco,
                    renda, telefone, identidade, nome);

        } else {
            if (!renda.equals(cliente.getRenda()) || !endereco.equals(cliente.getEndereco())
                    || !telefone.equals(cliente.getTelefone())) {

                cliente.setRenda(renda);
                cliente.setEndereco(endereco);
                cliente.setTelefone(telefone);

                clienteDataService.getTratarClienteDataServicePort().alterar(cliente);
            }
        }
    }
```

Figura 13 - Exemplo de serviço de lógica

O serviço de lógica referencia um serviço de dados para realizar as operações de persistência. No exemplo da Figura 13, o serviço *ManterCadastroClienteService* utiliza o serviço de dados *TratarClienteDataService* para consultar, incluir e alterar informações do cliente. Na próxima seção serão apresentadas as características de implementação dos serviços de dados.

3.2.2 Implementação dos serviços de dados

Como apresentado anteriormente, os serviços de dados são responsáveis por implementar as operações de acesso a dados e persistência, como por exemplo, operações CRUD. Neste trabalho, foi decidido utilizar um framework de mapeamento Objeto-Relacional [Keith e Schincariol, 2006] para manipular os dados, e a API de persistência Java ou simplesmente JPA [DeMichiel, 2009; Keith e Schincariol, 2006] foi a API utilizada.

JPA utiliza um modelo de persistência baseado em POJO e o mapeamento objeto relacional é feito, normalmente, através de anotações (também é possível utilizar *metadados* XML). JPA oferece suporte a diferentes provedores de persistência, como por

exemplo, Hibernate[10], EclipseLink[11], TopLink[12] e etc. Para implementação dos serviços apresentados na Tabela 1 foi utilizado o provedor de persistência TopLink. A primeira coisa que deve ser feita para utilizar JPA em uma aplicação Java é a criação de uma unidade de persistência, que define o conjunto de classes que serão gerenciadas pelo contexto de persistência e configurações adicionais, como a fonte de dados e o provedor de persistência utilizados. Essas configurações são definidas no arquivo *persistence.xml*.

Um exemplo do arquivo *persistence.xml* é apresentado na Figura 14. O elemento *persistence-unit* é composto pelo atributo *name e* pelo atributo *transaction-type.* O atributo *name* define o nome da unidade de persistência que será utilizado pelo atributo *unitName* da anotação *@PersistenceContext* (Figura 16) para obter a referência ao *EntityManager*[13], que é uma interface utilizada para interagir com o contexto de persistência para criar, alterar, remover e consultar entidades persistentes. O atributo *transaction-type* é utilizado para definir o tipo de transação que será utilizado pela aplicação. As transações podem ser gerenciadas pelo servidor de aplicação, utilizando a API JTA [Cheung e Matena, 2002] ou pela própria aplicação. O atributo *provider* contém o nome qualificado da classe do provedor de persistência utilizado pela unidade de persistência. O atributo *jta-data-source* especifica a referência JNDI[14] para a localização da fonte de dados (*datasource*) onde serão persistidos os dados. O elemento *properties* é utilizado para especificar tanto propriedades padrão da especificação como propriedades específicas de cada fabricante de provedor de persistência. A propriedade *toplink.ddl-generation* define a estratégia de geração do banco de dados automaticamente durante a publicação (*deploy*) da aplicação. A Tabela 2 apresenta as opções possíveis para a geração do banco de dados e tabelas utilizando o provedor de persistência *Toplink*.

Tabela 2 - Opções de geração do banco de dados utilizando o Toplink

Valor	Descrição
none	Opção padrão. Nenhuma tabela é criada ou apagada do banco de dados.
create-tables	Tabelas são criadas a cada publicação da aplicação.
drop-and-create-tables	As tabelas geradas automaticamente pela última publicação serão apagadas e recriadas.

A especificação JPA (Java Persistence API) define um modo de mapear POJOs a um banco de dados, através de metadados de persistência Java, de forma que possam ser inseridos e carregados do banco de dados sem que o desenvolvedor escreva códigos de conexão com o banco de dados. JPA também define a linguagem de consulta JPQL (Java Persistence Query Language) que tem características comparáveis às de SQL, mas é adaptada para trabalhar com objetos Java ao invés de esquema relacional puro [Burke e MonsoHaefel, 2006].

[10] http://www.hibernate.org/

[11] http://www.eclipse.org/eclipselink/

[12] http://www.oracle.com/technetwork/middleware/toplink/overview/index.html

[13] http://download.oracle.com/javaee/5/api/javax/persistence/EntityManager.html

[14] http://www.oracle.com/technetwork/java/index-jsp-137536.html

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

  <persistence-unit name="ModeloPU" transaction-type="JTA">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <jta-data-source>jdbc/servicos</jta-data-source>
    <properties>
      <property name="toplink.ddl-generation" value="create-tables"/>
    </properties>
  </persistence-unit>
</persistence>
```

Figura 14 - Arquivo *persistence.xml*

A primeira alteração a ser feita no projeto é alterar as classes de entidade, que estão no projeto do modelo (Figura 5) para permitir o uso de JPA. Desta forma, são incluídas anotações nas classes POJO a fim de que estas classes possam ser manipuladas pelo contexto de persistência. A Figura 15 apresenta um exemplo do uso de anotações JPA em um POJO. Neste caso, foram incluídas as anotações *@Entity* para indicar que tais classes representavam entidades, *@Id* que define um atributo como chave primária e @Temporal é utilizada para mapear datas especificando o tipo[15] desejado (data, hora e data e hora). Maiores detalhes sobre JPA podem ser encontrados [DeMichiel, 2009; Keith e Schincariol, 2006; Burke, 2006].

```
@Entity
public class Cliente implements Serializable {


    @Id
    private Long id;

    private String nome;
    private String cpf;
    private String identidade;
    private String telefone;
    private String endereco;
    private Double renda;

    @Temporal(TemporalType.DATE)
    private Date dataUltimaAtualizacao;

    @Temporal(TemporalType.DATE)
    private Date dataCadastro;
```

Figura 15 - Classe de entidade Cliente

O próximo passo foi a implementação dos serviços de dados para manipular as classes de entidade anotadas no passo anterior. Para isto, é necessário definir, no serviço de dados, um provedor de persistência, o que é feito através da anotação *@PersistenceContext* a qual referencia a unidade de persistência definida no arquivo *persistence.xml* (Figura 14). A partir deste provedor é recuperado o *EntityManager* res-

[15] http://download.oracle.com/javaee/5/api/javax/persistence/TemporalType.html

ponsável por realizar a persistência dos objetos correspondentes às classes de entidade. A Figura 16 apresenta um exemplo de implementação de serviço de dados utilizando JPA. Vale ressaltar que o uso da anotação *@PersistenceContext* requer o uso do servidor de aplicação que é responsável por gerenciar o contexto transacional da aplicação. A Figura 17 apresenta um exemplo de persistência do objeto Cliente através da invocação do método *persist* da interface *EntityManager*.

```
@WebService()
@Stateless()
public class TratarClienteDataService {

    @PersistenceContext(unitName = "TratarClienteDataServicePU")
    EntityManager em;
```

Figura 16 - Exemplo de serviço de dados com JPA

```
public void incluirCliente(String cpf, String endereco, Double renda,
        String telefone, String identidade, String nome){

    Cliente cliente = new Cliente();
    cliente.setNome(nome);
    cliente.setCpf(cpf);
    cliente.setEndereco(endereco);
    cliente.setTelefone(telefone);
    cliente.setRenda(renda);
    cliente.setIdentidade(identidade);
    cliente.setDataUltimaAtualizacao(new Date());
    cliente.setDataCadastro(new Date());

    em.persist(cliente);
}
```

Figura 17 - Exemplo de persistência utilizando JPA

Como mencionado anteriormente, JPA possui uma linguagem orientada a objetos de consulta e manipulação de dados, chamada JPQL. A Figura 18 apresenta um exemplo de uso da linguagem. Neste exemplo um cliente é consultado através do seu CPF. O método *createQuery* da interface *EntityManager* cria uma instância do objeto *Query* a partir de uma consulta JPQL. Em seguida, o método *setParameter* faz a ligação entre o parâmetro *:cpfParam* e o argumento *cpf* recebido pelo método *consultar*. Por fim, o método *getSingleResult* da interface *Query* executa o comando *select*. Vale observar que caso o comando *select* não retorne informações uma exceção do tipo *NoResultException* será lançada. Da mesma forma, uma exceção do tipo *NonUniqueResultException* será lançada caso o comando *select* retorne mais de um resultado, neste caso o comando *getResultList* deve ser utilizado.

```
public Cliente consultar(String cpf) {
    Cliente cliente = null;
    Query q = em.createQuery("select c from Cliente c where c.cpf = :cpfParam");
    q.setParameter("cpfParam", cpf);

    try {
        cliente = (Cliente) q.getSingleResult();
    } catch (javax.persistence.NoResultException nre) {}

    return cliente;
}
```

Figura 18 – Exemplo consulta JPQL

3.3 Implementação da aplicação cliente

Com objetivo de validar o uso dos serviços implementados foi desenvolvida uma aplicação para consumir os serviços. Esta aplicação implementa os requisitos de negócio definidos no processo Analisar Pedido de Crédito, apresentado por Diirr *et al.* [2010]. Na fase de análise foram especificados três casos de uso apresentados no Anexo III, que foram implementados pela aplicação *AnalisarPedidoCredito*. A aplicação foi implementada utilizando tecnologia Java em uma arquitetura *web*. A aplicação faz uso dos serviços desenvolvidos na seção 3.2 e 3.3 para implementar suas funcionalidades.

Para a geração das classes clientes do serviço (também chamadas stubs ou proxies), o *Netbeans* utiliza o utilitário *wsimport*[16] (que vem com a distribuição do Java Development Kit). A Figura 20 apresenta o comando *wsimport* sendo executado e a relação das classes geradas. Algumas das classes geradas são necessárias para o consumo do serviço, como a classe *CalcularTaxaContrato*, que representa a mensagem SOAP de requisição ao serviço, e a *CalcularTaxaContratoResponse*, que representa a mensagem SOAP de resposta do serviço além da classe *CalcularTaxaContratoService* que representa a interface do serviço *CalcularTaxaContrato*.

As classes geradas são armazenadas em uma pasta chamada Referências de serviços web, conforme destacado na Figura 21. Uma vez criadas as referências na aplicação cliente, os serviços foram consumidos através de classes Servlets com anotações *@WebServiceRef*. A Figura 19 apresenta a classe Servlet do caso de uso *Consultar proposta de crédito* consumindo o serviço *TratarPropostaCredito*.

[16] https://jax-ws.dev.java.net/jax-ws-ea3/docs/wsimport.html

```java
public class ConsultarPropostaServlet extends HttpServlet {

    @WebServiceRef(wsdlLocation = ""
    + "http://localhost:8080/TratarPropostaCreditoDataServiceService/TratarPropostaCreditoDataService?WSDL")
    TratarPropostaCreditoDataServiceService tratarPropostaCreditoService;

    protected void consultarPropostaCredito(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        TratarPropostaCreditoDataService port = tratarPropostaCreditoService.getTratarPropostaCreditoDataServicePort();

        String resultadoVerificacao = request.getParameter("resultadoVerificacao");
        List listaPropostaCredito = null;

        if ("".equals(resultadoVerificacao)) {
            listaPropostaCredito = port.listarPropostaCredito();
        } else {
            listaPropostaCredito = port.listarPropostaCreditoPorResultadoVerificacao(resultadoVerificacao);
        }

        request.getSession().setAttribute("listarPropostaCredito", listaPropostaCredito);

        if (listaPropostaCredito == null || listaPropostaCredito.isEmpty()) {
            request.setAttribute("mensagem", "Não existem propostas cadastradas!");
        }

        RequestDispatcher dispatcher = request.getRequestDispatcher("consultarProposta.jsp");
        dispatcher.forward(request, response);

    }
```

Figura 19 - Classe Servlet

```
command line: wsimport -d C:\Users\np2tec-07\Documents\NetBeansProjects
              \VerificarContratoRisco\build\generated\jax-wsCache\CalcularTaxaContratoServiceService
parsing WSDL...


generating code...

br\np2tec\soa\servicos\CalcularTaxaContrato.java
br\np2tec\soa\servicos\CalcularTaxaContratoResponse.java
br\np2tec\soa\servicos\CalcularTaxaContratoService.java
br\np2tec\soa\servicos\CalcularTaxaContratoServiceService.java
br\np2tec\soa\servicos\ObjectFactory.java
br\np2tec\soa\servicos\package-info.java
```

Figura 20 - Geração das classes clientes com *wsimport*

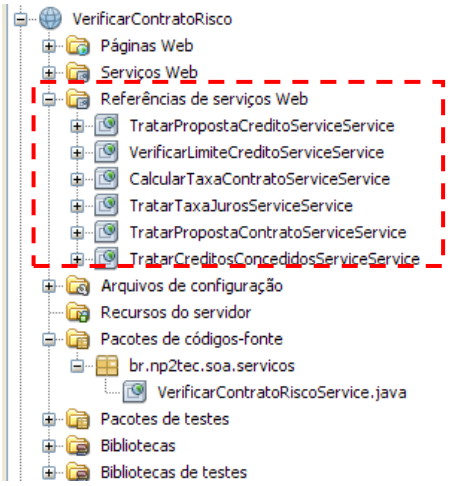


Figura 21 - Referências de serviços web no *Netbeans*

O uso de anotações disponíveis como parte nativa da linguagem Java simplifica o desenvolvimento de serviços e a geração das classes necessárias para seu consumo. O

uso de JPA permite abstrair o modelo de dados do modelo de objetos, sem que haja necessidade dos desenvolvedores dos serviços conhecerem a estrutura e a semântica dos dados que o serviço manipula, permitindo inclusive que os serviços de lógica e dados sejam mantidos por equipes diferentes.

4 Conclusões

Este relatório descreveu um conjunto de passos seguidos para especificação e implementação de serviços os quais foram identificados e analisados no estudo de caso realizado por Souza *et al.* [2010]. Foram apresentadas atividades realizadas para modelagem de serviços com UML padrão, modelos para detalhamento de operações de serviços, especificação de aplicação cliente de serviços, implementação das entidades de dados do negócio, implementação dos serviços de dados e de lógica (tecnologia utilizada, estrutura interna dos projetos, decisões de projeto etc.) e implementação da aplicação cliente.

Com a execução destas atividades foi possível verificar que os serviços obtidos com o método eram alinhados ao negócio da organização, tendo em vista que tratam as principais necessidades descritas nos processos de negócio da organização. Além disso, como as heurísticas do método de identificação consideram o fluxo do processo e o detalhamento de suas atividades, o processo é analisado sintaticamente e semanticamente como um meio de tentar identificar todas as funcionalidades expressas no modelo que poderiam tornar-se serviços (independente de como serão combinadas posteriormente). Porém, como uma premissa para aplicação das heurísticas do método, é necessário que os processos apresentem detalhamento dos fluxos e das atividades. No caso de processos elaborados com a notação EPC [Scheer, 2000], os fluxos são especificados em diagramas eEPC e as atividades são detalhadas em diagramas FADs. Os principais elementos da notação EPC são apresentados no anexo IV. Os FADs precisam conter os seguintes elementos e suas respectivas descrições: regras de negócio, clusters (ligados a portadores de informação, quando a informação for lida ou escrita a partir destes), requisitos de negócio, sistemas executores ou de apoio às atividades (associados a seus requisitos de negócio) e papéis executores das atividades. Apesar desta abrangência possibilitar, primeiramente, a identificação de serviços candidatos diferentes com funcionalidades semelhantes, a etapa de consolidação dos serviços permite a eliminação destas duplicações.

Além disso, as heurísticas possibilitam a identificação de serviços com potencial de reutilização. Todos os elementos dos modelos de processos de negócio que são reutilizados em mais de um processo poderão levar, consequentemente, ao reuso dos serviços identificados a partir destes elementos. Além disso, são consideradas as informações de ocorrência dos serviços em atividades de múltiplas instâncias e a sua automatização em outros sistemas da organização, que auxiliam na verificação da reutilização do serviço. É realizado, também, o agrupamento dos serviços de dados de acordo com a manipulação das entidades definidas no modelo canônico, resultando em uma menor quantidade de serviços implementados fisicamente e com maior reuso.

As informações de priorização geradas na fase de análise são importantes para a governança SOA (na definição de quais serviços devem ser implementados para gerar maior ganho para a organização). No estudo de caso realizado no presente trabalho, todos os serviços foram implementados. No entanto, se fossem seguidas as prioridades identificadas, os serviços a serem implementados inicialmente seriam os serviços de dados *Tratar créditos concedidos* e *Tratar proposta de contrato* que possuem

maior prioridade (ambos com 9). As informações de granularidade e reuso podem ser utilizadas na fase posterior de projeto, quando decisões sobre a publicação e provisão dos serviços são tomadas. Para serviços de alta granularidade, é importante atentar ao tráfego das informações e para serviços de alto reuso são necessárias preocupações maiores com sua disponibilidade aos consumidores.

Algumas questões já discutidas na literatura também se confirmaram, como: versionamento de serviços após alteração (a fim de manter versões novas e antigas publicadas, evitando impacto nos consumidores); utilização de barramento de serviços ESB (afim de que os clientes não possuam referências diretas à localização do serviço, evitando dependências) [Heiwitt, 2009]; uso de registro de serviços UDDI (de forma a evitar a replicação de serviços e operações) [Apte e Mehta, 2002]; dentre outros casos.

Como trabalho futuro, sugerimos a implantação dos serviços identificados, realização de testes sistemáticos dos serviços e aplicação cliente implementada, além do monitoramento do uso dos serviços com posterior análise de uso.

5 Referências bibliográficas

ARIS, 2006. **Help Documentation**. ARIS Business Architect 7.0 v. 7.0.2.234414, IDS Scheer AG.

APTE, N; MEHTA, T. **UDDI: building registry-based Web services solutions**. Prentice Hall, 2002, 404 p.

AZEVEDO, L.; PEREIRA, V.; REVOREDO, K; SOUZA, J.; SANTORO, F.; BAIAO, F.; SOUSA, H. P. **Metodologia de identificação de serviços a partir da modelagem de processos de negócio**. Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0021/2009, 2009a. Disponível (também) em: http://seer.unirio.br/index.php/monografiasppgi.

AZEVEDO, L.G., BAIAO, F., SANTORO, F., SOUZA, J., REVOREDO, K., PEREIRA, V., HERLAIN, I. 2009b. **Identificação de serviços a partir da modelagem de processos de negócio**. In: Simpósio Brasileiro de Sistemas de Informação (SBSI), Brasília.

AZEVEDO, L.G., SANTORO, F., BAIAO, F., SOUZA, J., REVOREDO, K., PEREIRA, V., HERLAIN, I., 2009c. **A Method for Service Identification from Business Process Models in a SOA Approach**. In: 10th International Workshop on Business Process Modeling, Development, and Support (BPMDS), 2009, Amsterdam. Enterprise, Business-Process, and Information Systems Modelling. v. 29.

AZEVEDO, L.G., SOUSA, H.P., SOUZA, J.F., SANTORO, F., BAIÃO, F. 2009d. **Identificação Automática de Serviços Candidatos a partir de Modelos de Processos de Negócio.** In: Conferencia IADIS Ibero Americana WWW/INTERNET 2009, Alcalá de Henares, Madri,Espanha.

AZEVEDO, L.; SOUZA, J.; SANTORO, F.; BAIAO, F. **Metodologia para análise e projeto de serviços em uma abordagem SOA**. Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0023/2009, 2009e. Disponível (também) em: http://seer.unirio.br/index.php/monografiasppgi.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. Campus, 2006, 308 p.

BPM-ADVISOR, 2009 - **Padrões e Notações**, disponível em http://www.bpm-advisor.com.br/padnotac.htm. Acessado em 6 de Jan. de 2010.

BURKE, B. e MONSON-HAEFEL, R. **Enterprise JavaBeans, 3.0.** O'Reilly, 2006, 760 p. Bibliografia: ISBN-10: 0-596-00978-X.

CHEUNG, S., MATENA, V., 2002**. Java Transaction API (JTA) 1.1**. Disponível em <https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewFilteredProducts-SimpleBundleDownload>. Acessado em 10 de dezembro de 2010.

DeMICHIEL, L., 2009**. Java Persistence API (JPA) 2.0**. Disponível em <http://www.jcp.org/en/jsr/detail?id=317>. Acessado em 10 de dezembro de 2010.

DIIRR, T.; SOUZA, A.; AZEVEDO, L. G.; SANTORO, F. **Modelo de processos de negócio Analisar Pedido de Crédito**. (2010). Relatório técnico do Departamento de Informática Aplicada – Universidade Federal do Estado do Rio de Janeiro (UNIRO).

ERL, T., **Service-Oriented Architecture: concepts, technology, and Design.** Prentice Hall, Crawfordsville: Indiana, 2005, 792 p. Bibliografia: ISBN, 0-13-18-5858-0.

ERL, T., **SOA: Principles of Service Design**, Prentice Hall, Crawfordsville: Indiana, 2007, 608 p. Bibliografia: ISBN, 0-13-23-4482-3.

FOWLER, M., 2004. **Inversion of Control Containers and the Dependency Injection pattern**. Disponível em <http://martinfowler.com/articles/injection.html#FormsOfDependencyInjection>. Acessado em 10 Dez. de 2010.

HEWITT, E. **Java SOA Cookbook**, O'Reilly, 2009.

KAWAGUCHI, K., FIALLI, J., VAJJHALA, S., 2010. **The Java Architecture for XML Binding (JAXB) 2.2**. Disponível em <http://jcp.org/aboutJava/communityprocess/mrel/jsr222/index2.html>. Acessado em 10 Dez. de 2010.

KEITH, M., SCHINCARIOL, M., 2006. **Pro EJB 3: Java Persistence API.** Apress, 2006. Bibliografia: ISBN-978-1-59059-645-6.

RAHMANI, A. T.; RAFE, V.; SEDIGHIAN, S.; ABBASPOUR, A.. "**An MDA-Based Modeling and Design of Service Oriented Architecture**". Computational Science – ICCS 2006 - 6th International Conference, 2006.

KOTAMRAJU, J., 2009a. **The Java API for XML-Based Web Services (JAX-WS) 2.2.** Disponível em <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index3.html>. Acessado em 10 Dez. 2010.

KOTAMRAJU, J., 2009b. **Implementing Enterprise Web Services for Java EE.** Disponível em <http://cds-esd.sun.com/ESD5/JSCDL/websvcs/1.3-final/websvcs-1_3-final-spec.pdf?AuthParam=1286802282_b9df1ad9bd46475f99eb64ad7226b3ad&TicketId=B%2Fw6nRSHRFNPSBNGOV5blAbh&GroupName=CDS&FilePath=/ESD5/JSCDL/websvcs/1.3-final/websvcs-1_3-final-spec.pdf&File=websvcs-1_3-final-spec.pdf>. Acessado em 10 Jan. 2010.

MORDANI, R., 2009. **Common Annotation for the Java Platform.** Disponível em <http://cds-esd.sun.com/ESD5/JSCDL/common_annotations/1.1-mrel/common_annotations-1_1-mrel-spec.pdf?AuthParam=1286804089_cf4ad5fcc770a898f95244fc943cce13&TicketId=nod2BlAYRH5%2Fl%2BEsnESfVJ%2BbdA%3D%3D&GroupName=CDS&FilePath=/ESD5/J

SCDL/common_annotations/1.1-mrel/common_annotations-1_1-mrel-spec.pdf&File=common_annotations-1_1-mrel-spec.pdf>. Acessado em 10 Jan. 2010.

OMG, 2010. **Unified Modeling Language**. Disponível em <http://www.uml.org/>. Acessado em 10 Dez. 2010.

PAPAZOGLOU, M. P., TRAVERSO, P., DUTSDAR, S., LEYMANN, F., 2007. **Service Oriented Computing: State of the Art and Research Challenges.** In: IEEE Computer Society, vol. 40, issue 11, pp. 38-45.

RICHARDSON, C. **POJOs in action - Developing Enterprise Applications with Lightweight Frameworks.** Manning, 2006, 592 p.

SCHEER, A.-W., 2000. **ARIS - Business Process Modelling**. Springer, Berlin, Alemanha.

SOUZA, A., DIIRR, T., AZEVEDO, L. G., SANTORO, F. **Identificação e Análise de Serviços a partir de Modelos de Processos de Negócio: Um estudo de caso**. Relatórios Técnico NP2Tec/UNIRIO, 2010. Disponível em <http://np2tec.uniriotec.br:9093/np2tec/publicacoes/relatorios-tecnicos>. Acessado em Ago. 2010.

WOOLF, B. **Introduction to SOA governance**, 2006. Disponível em: <http://www.ibm.com/developerworks/library/ar-servgov>. Acessado em 10 Dez. 2010.

Anexo I – Modelos de classes dos serviços

Este anexo detalha a modelagem dos serviços em UML padrão. As figuras Figura 22, Figura 23, Figura 24, Figura 25, Figura 26, Figura 27, Figura 28, Figura 29, Figura 30 e Figura 31 apresentam os modelos de classes dos serviços projetados a partir do estudo de caso apresentado por Souza *et al.* [2010]. Cada diagrama apresenta as seguintes informações:

- Classe que representa o serviço, a qual aparece com o estereótipo <<*interface*>>.
- Classes que representam as entidades manipuladas pelo serviço. Para estas classes é ajustada uma dependência entre o serviço e a entidade.
- Classes utilizadas pelo serviço. Esta informação é representada por uma dependência criada entre o serviço e outro serviço.

Como exemplos, na Figura 22, o serviço TratarCliente manipula a entidade Cliente. Na Figura 23, o serviço TratarPropostaCredito manipula as entidades PropostaDeCredito, Funcionario e Peca. Neste caso, o serviço TratarPropostaCredito também invoca TratarCliente. É importante ressaltar que o relacionamento de dependência representa a dependência entre o serviço e entidades e entre o serviço e outros serviços. Este relacionamento não explicita qual método que manipula a entidade ou qual método o outro serviço é invocado.

Observe ainda que o diagrama representa, inclusive, uma cadeia de dependências entre serviços. Um exemplo aparece na Figura 24, onde é representado que o serviço TratarCreditosETaxas invoca o serviço TratarPropostaCredito que invoca o serviço TratarCliente.

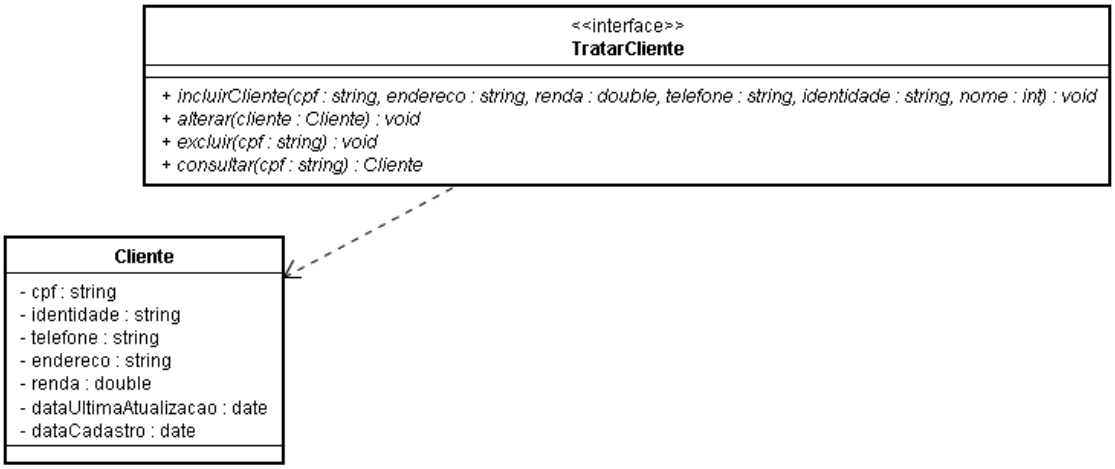


Figura 22 - Serviço TratarCliente

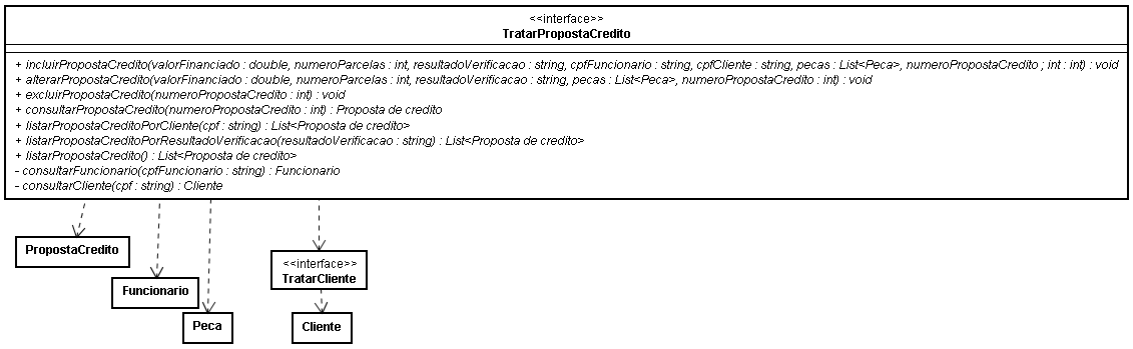


Figura 23 - Serviço TratarPropostaCrédito

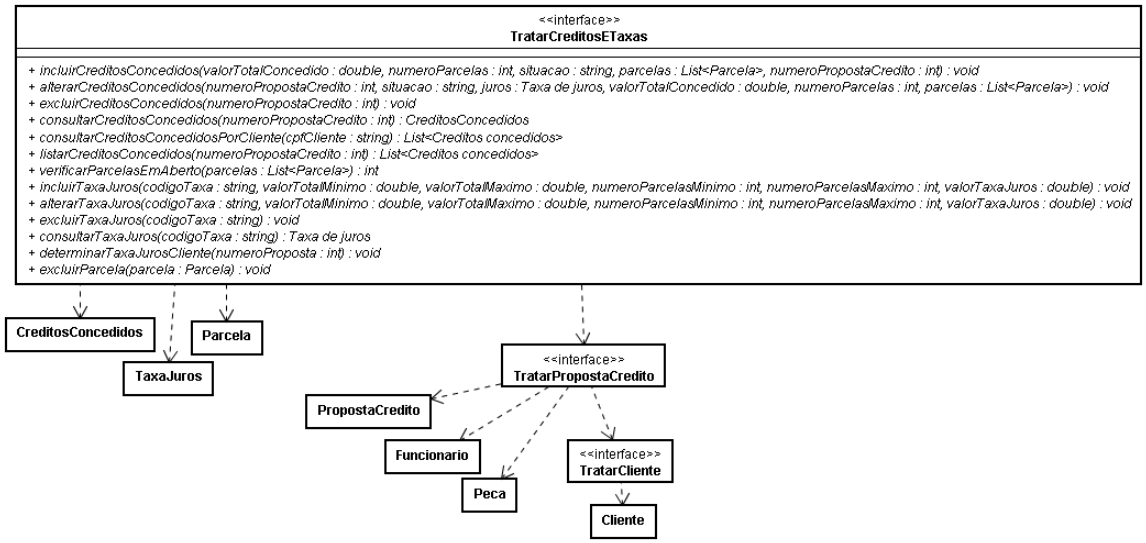


Figura 24 - Serviço TratarCreditosETaxas

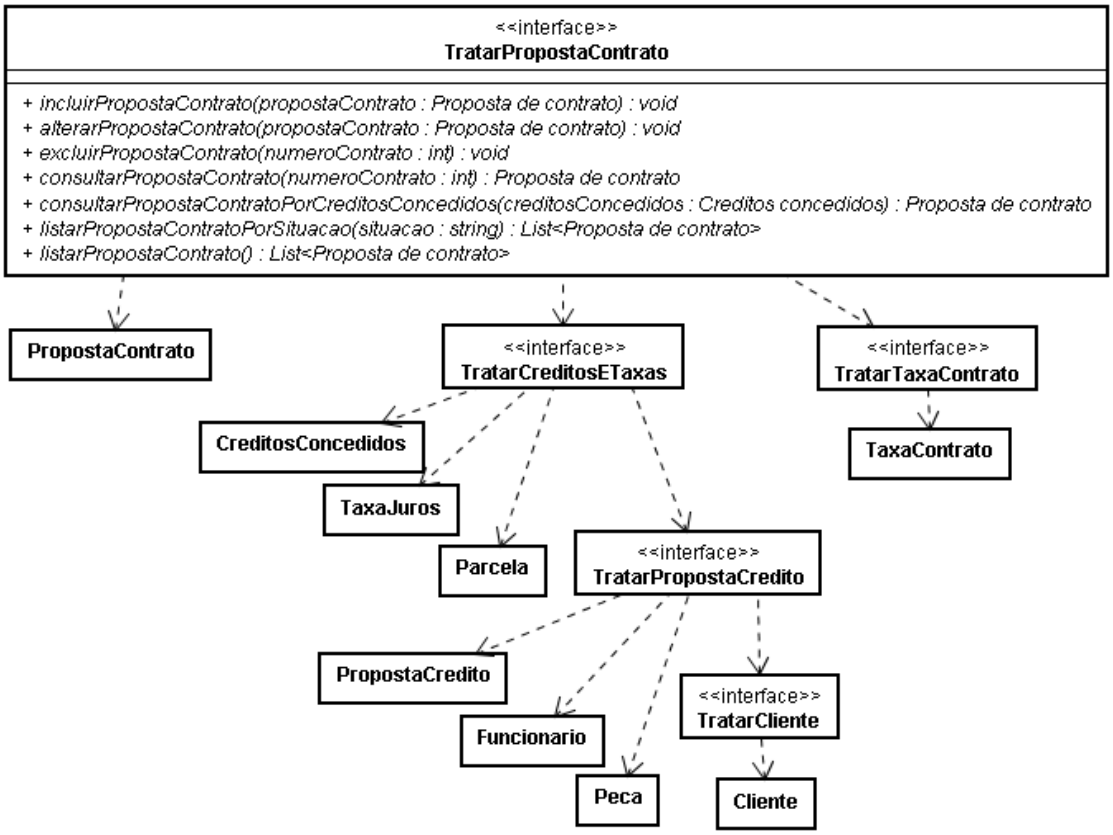


Figura 25 – Serviço TratarPropostaContrato

Figura 26 – Serviço TratarTaxaContrato

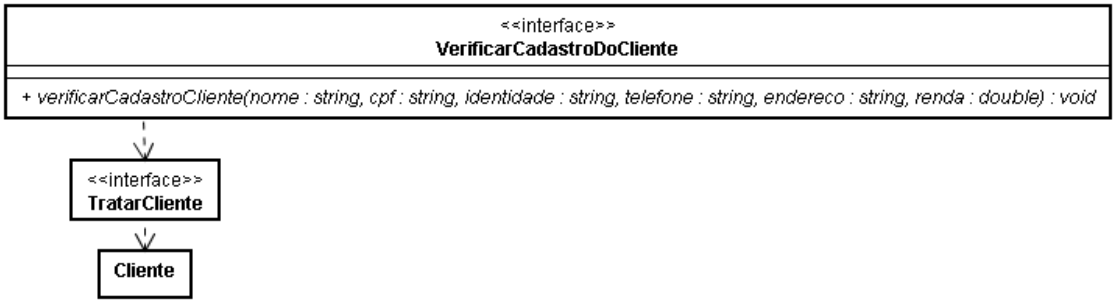


Figura 27 - Serviço VerificarCadastroDoCliente

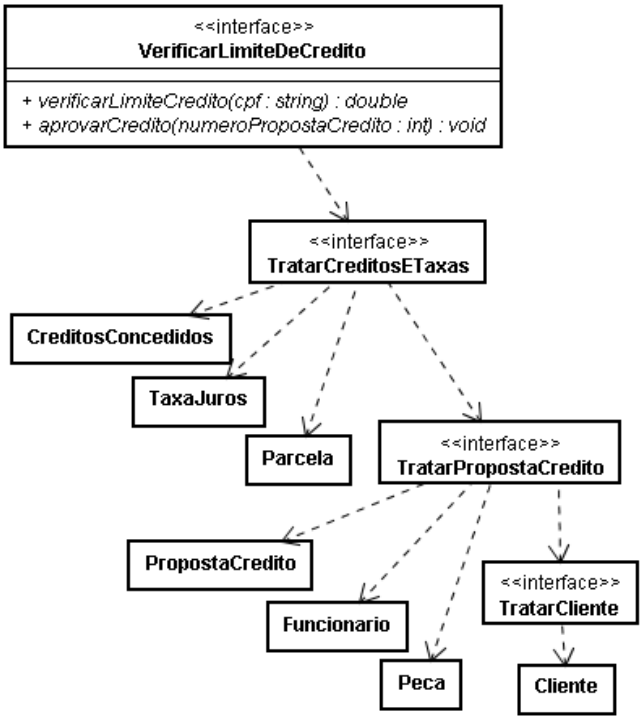


Figura 28 - Serviço VerificarLimiteDeCrédito

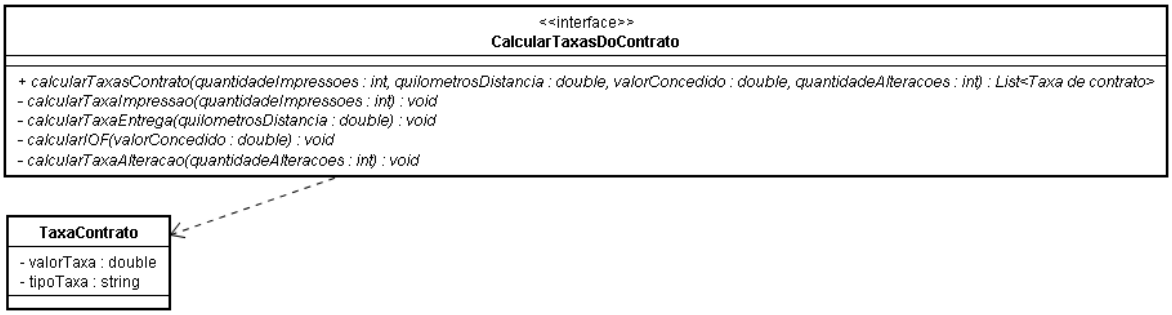


Figura 29 - Serviço CalcularTaxasDoContrato

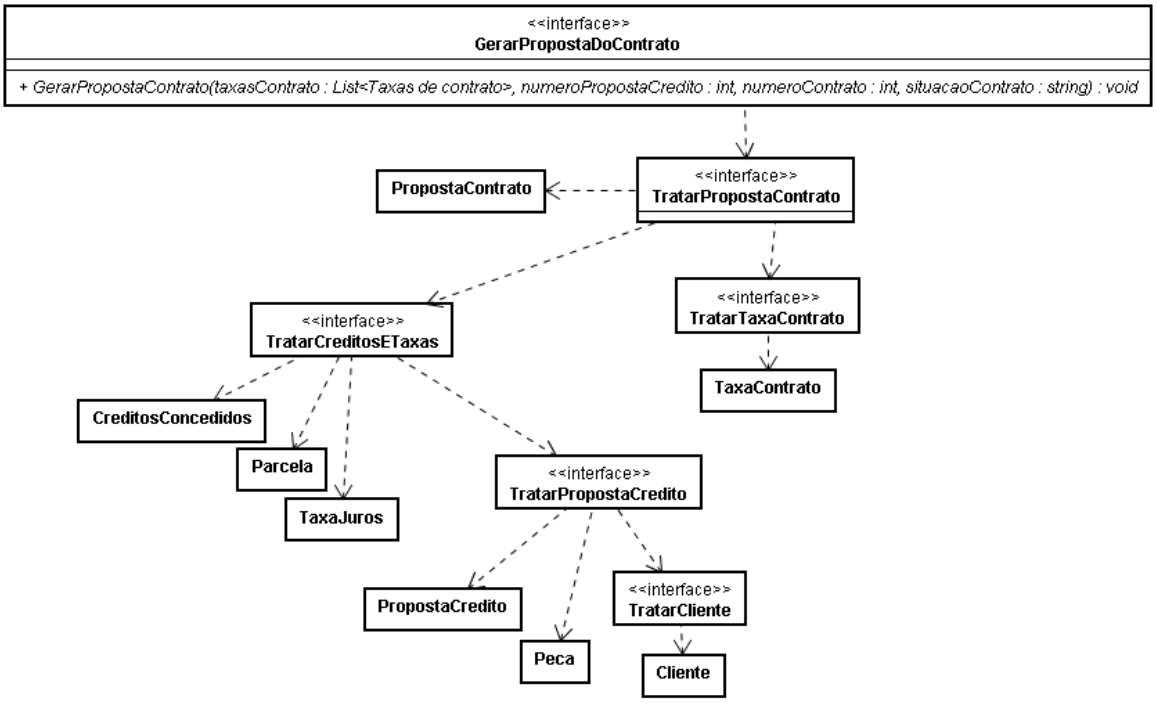


Figura 30 - Serviço GerarPropostaDoContrato

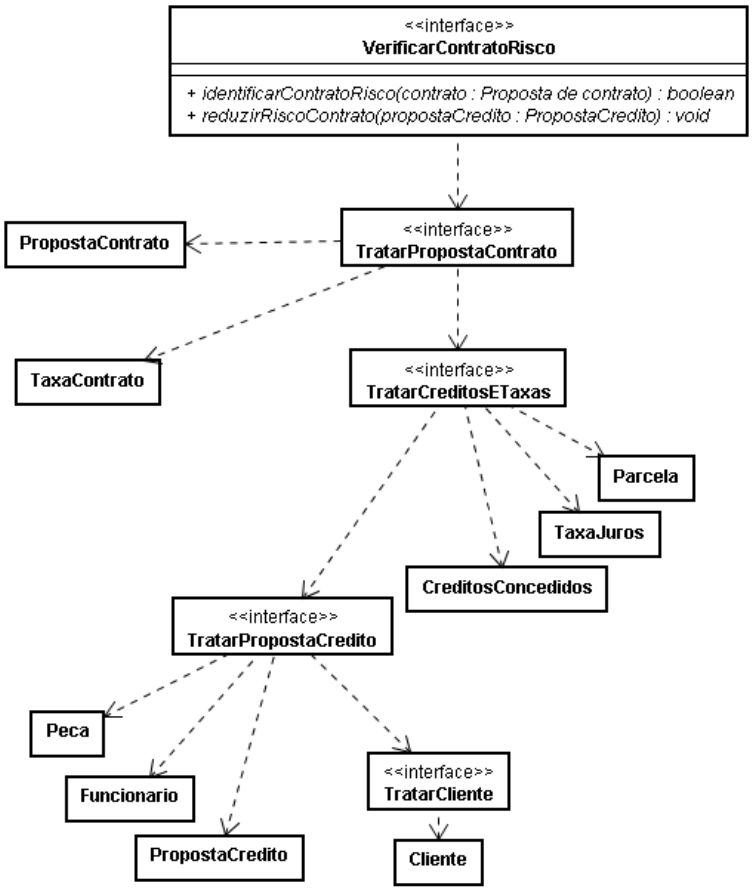


Figura 31 - Serviço VerificarContratoRisco

Anexo II – Diagramas de atividades de operações dos serviços

Este anexo detalha os diagramas de atividades criados para as operações dos serviços mais complexas. Estes diagramas facilitam o desenvolvedor no entendimento do que deve ser implementado para operações que possuem fluxo mais complexos. Na nossa prática, observamos que não há necessidade de se elaborar estes diagramas para operações mais simples. As Figura 32, Figura 33, Figura 34, Figura 35, Figura 36, Figura 37, Figura 38 e Figura 39 apresentam estes diagramas de atividades modelados. Em alguns casos, foram incluídas anotações com explicações mais detalhadas do que deve ser feito. Estas anotações facilitam o melhor entendimento do que deve ser feito em uma determinada chamada de operação de serviço. Um exemplo de anotações é apresentado na Figura 33.

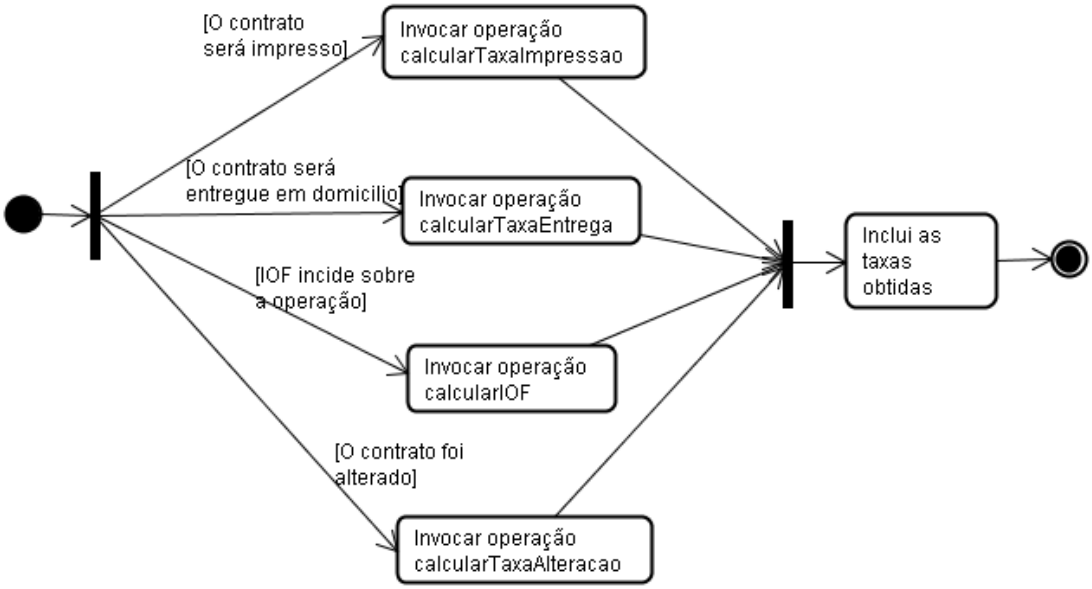


Figura 32 - Fluxo da operação "Calcular taxas contrato”

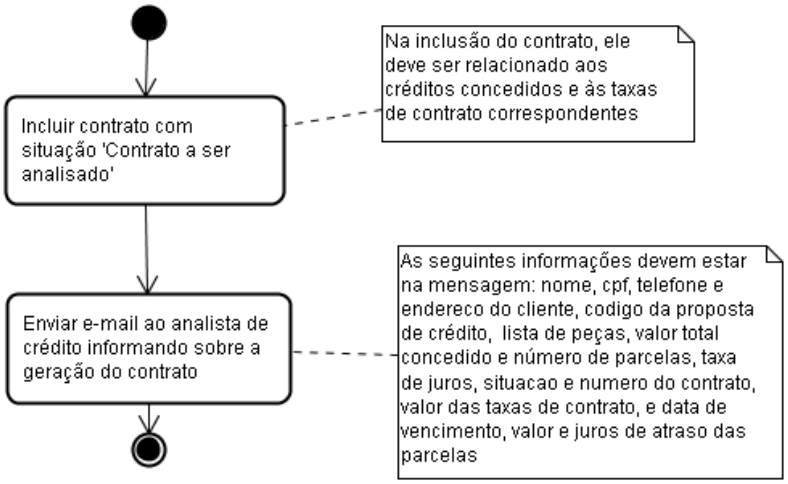


Figura 33 - Fluxo da operação "Gerar proposta contrato"

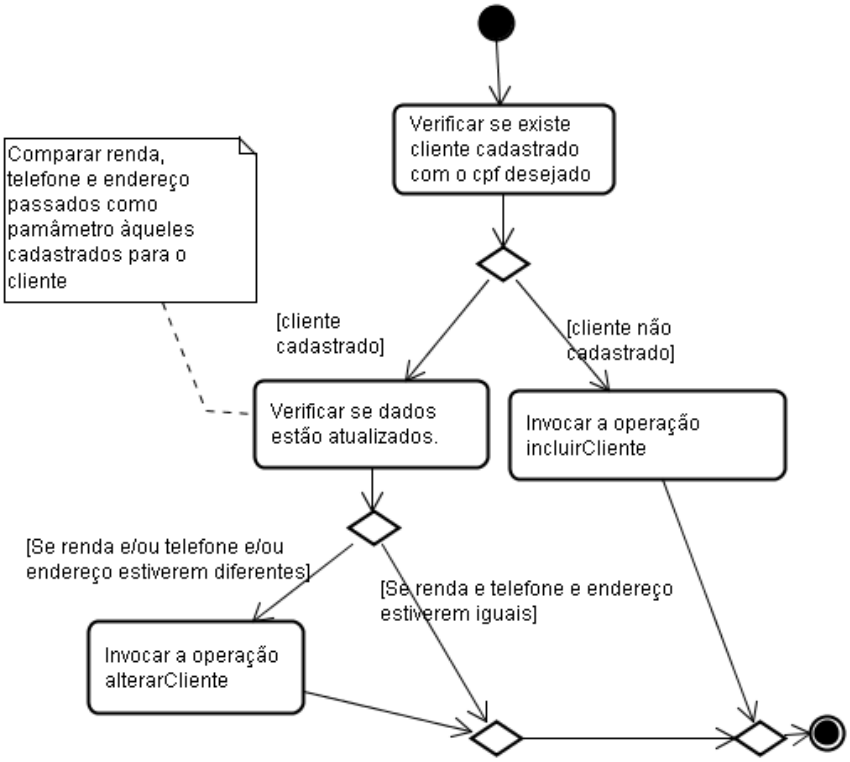


Figura 34 - Fluxo da operação "Verificar cadastro cliente"

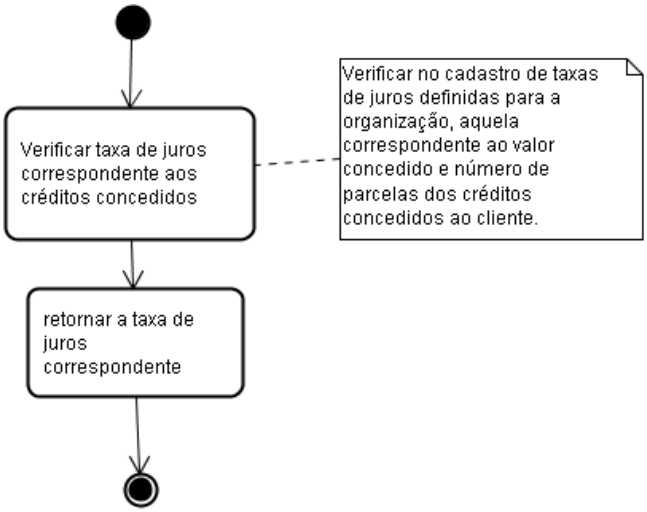


Figura 35 - Fluxo da operação "Determinar taxa juros do cliente"

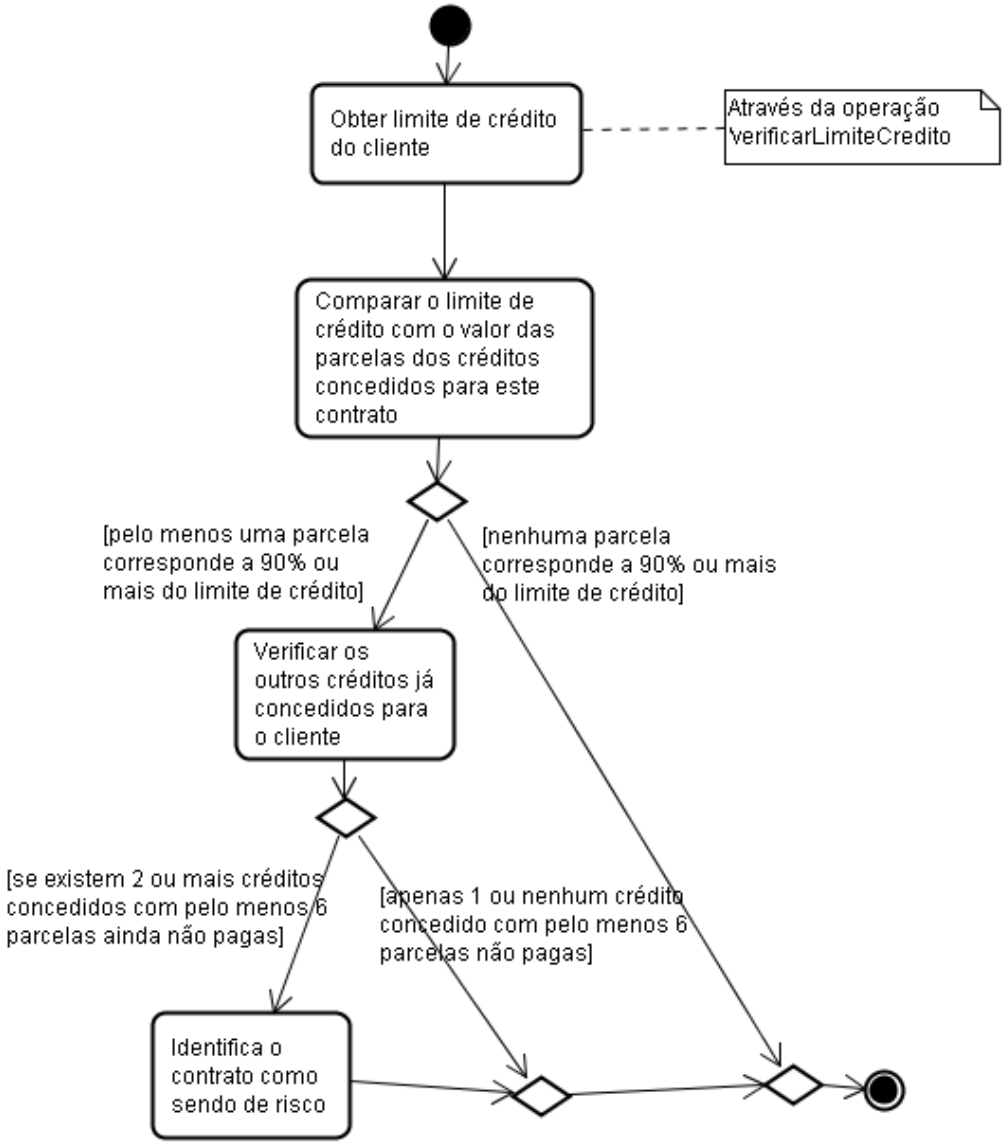


Figura 36 - Fluxo da operação "Identificar contrato risco"

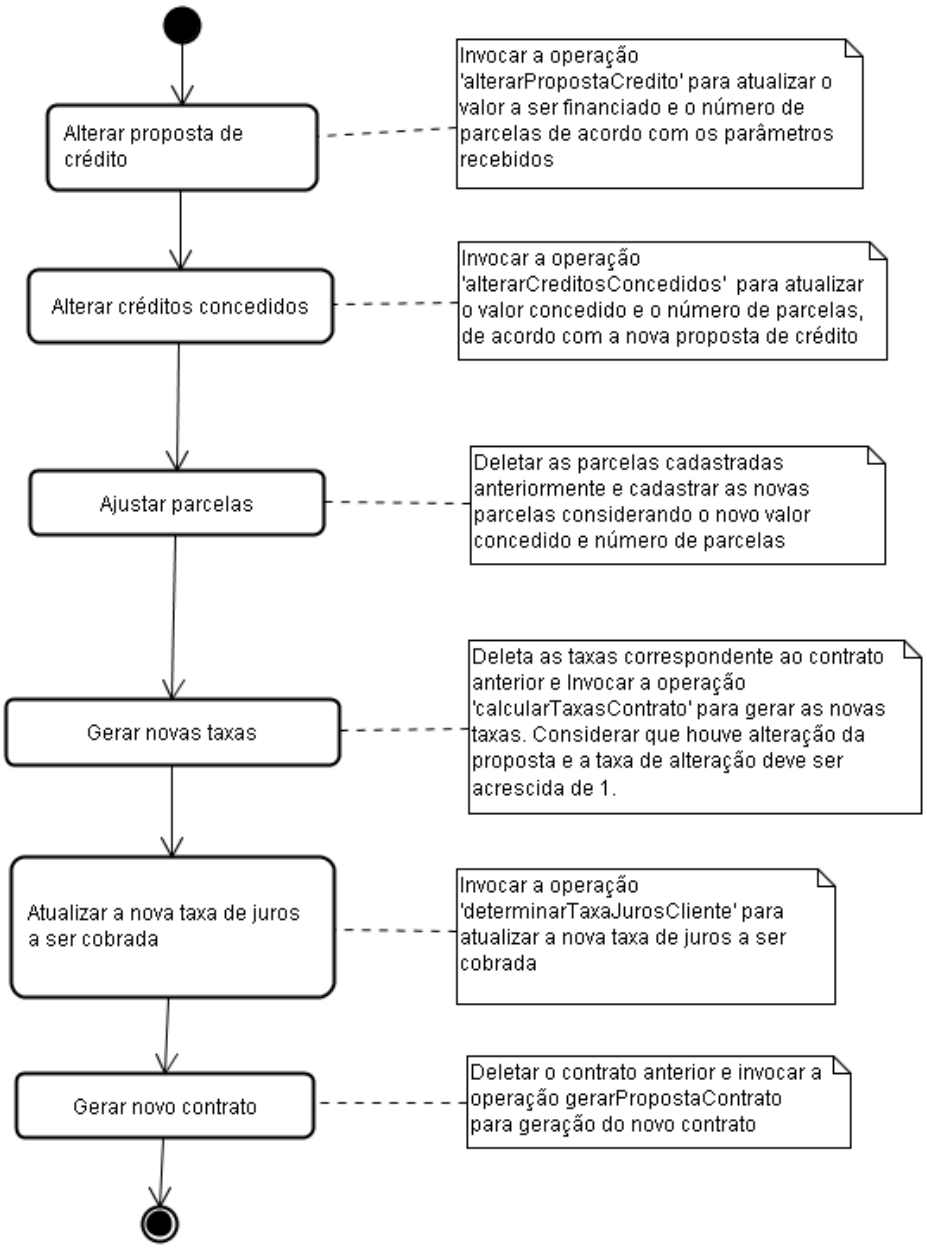


Figura 37 - Fluxo da operação "Reduzir risco contrato"

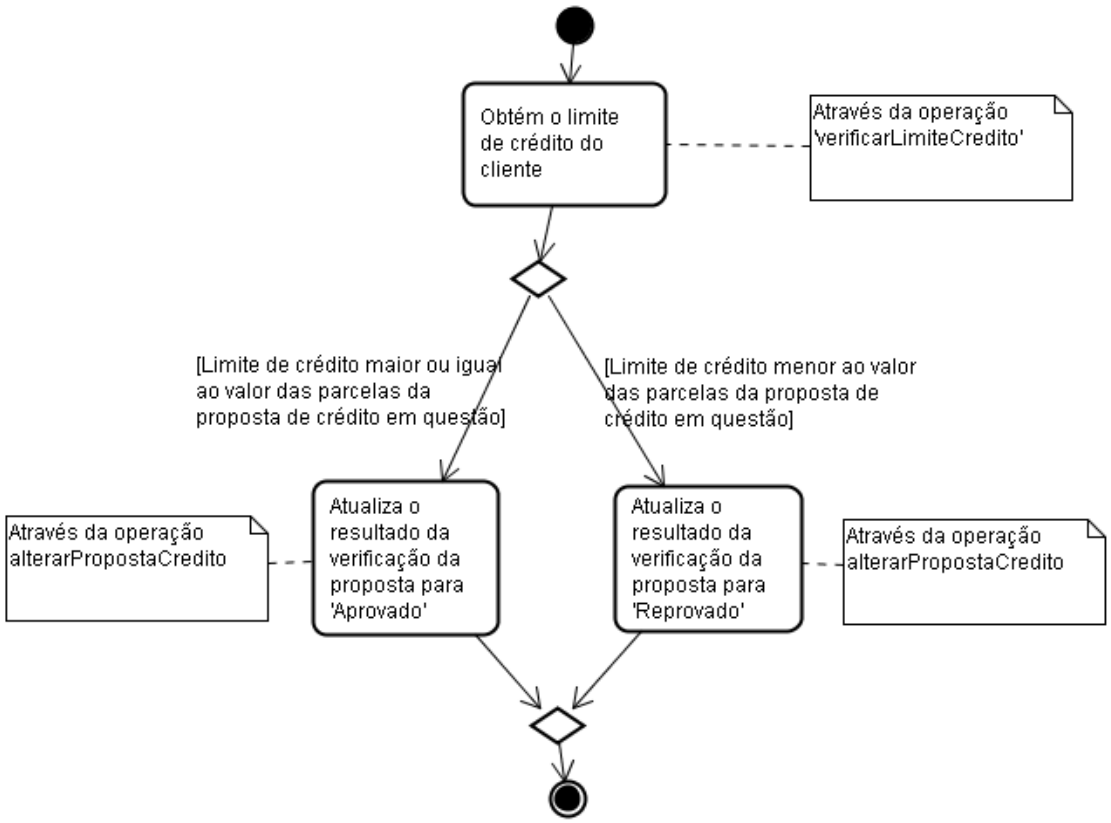


Figura 38 - Fluxo da operação "Aprovar crédito"

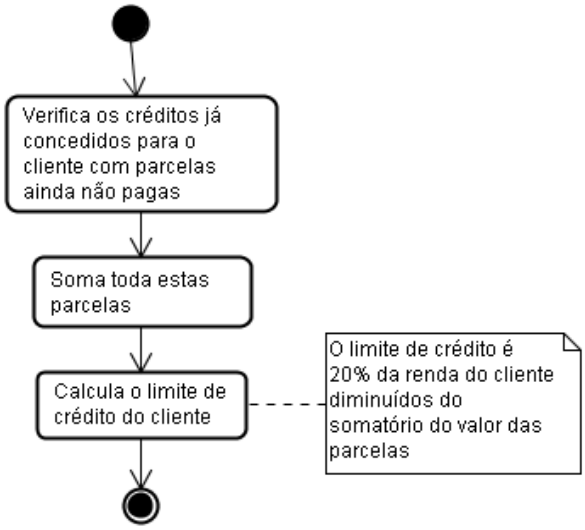


Figura 39 - Fluxo da operação "Verificar limite crédito"

Anexo III – Casos de uso da aplicação cliente "Analisar pedido de crédito"

Este anexo apresenta a especificação dos casos de uso da aplicação "Analisar Pedido de Crédito" a qual invoca os serviços identificados, analisados, projetados e implementados. Os casos de uso foram identificados a partir do modelo de processos des-

crito em Souza *et al.* [2010]. Eles foram descritos de acordo com o formato de descrição de casos de uso apresentado por Bezerra[2006], considerando as informações principais para descrição dos casos de uso e incluindo extensão para explicitar quais atividades do modelo de processos são representadas no caso de uso. Por exemplo, o caso "Consultar proposta de crédito" representa as ações executadas na atividade "Comunicar proposta não aprovada", enquanto que o caso de uso "Analisar contrato" representa as ações executadas nas atividades "Analisar contrato", "Cancelar contrato de risco", "Alterar proposta de crédito" e "Comunicar não aprovação de contrato de risco".

1. Consultar proposta de crédito

Relação com o modelo de processos: Atividade Comunicar proposta não aprovada.

Observações quanto decisões tomadas em relação à modelagem de processos: Apesar do caso de uso ter sido identificado a partir de uma única atividade, observamos que sua generalização para permitir consulta mais geral sobre proposta é simples e aumenta o reuso do caso de uso.

Ator: Atendente

Tipo: Primário e Essencial

Descrição: O atendente consulta as propostas de crédito realizadas e visualiza seus detalhes.

Pré-condições: Propostas de crédito cadastradas

Pós-condições: Propostas de crédito consultadas

Início: O caso de uso inicia quando o Atendente deseja consultar propostas de crédito realizadas pelos clientes.

Fluxo principal:

1. O Atendente escolhe a opção para consulta de propostas de crédito
2. O Sistema lista propostas de créditos contendo as seguintes informações:
    - Código da proposta de crédito
    - Resultado de verificação
    - CPF do cliente
    - Nome do cliente

Além de opção para filtragem através do resultado de verificação das propostas (aprovada ou reprovada).

3. O Atendente escolhe qual proposta deseja visualizar.
4. O Sistema exibe os atributos da proposta de crédito, contendo as seguintes informações:
    - Proposta de crédito
        - Código da proposta de crédito
        - Resultado de verificação
        - Valor financiado

        - Número de parcelas
        - Data de cadastro
    - Cliente
        - CPF
        - Nome
        - Identidade
        - Renda
        - Endereço
    - Lista de peças
        - Nome de cada peça
    - Funcionário responsável pelo cadastro
        - CPF
        - Nome

Fluxo alternativo 2.1: Não existem propostas de crédito cadastradas

1. O Sistema exibe mensagem informando que não existem propostas cadastradas
2. Encerrar caso de uso

Fluxo alternativo 2.2: Ator escolhe opção para filtragem de propostas por resultado de verificação.

1. O Sistema exibe opção para escolha do resultado de verificação (aprovada ou reprovada)
2. O ator escolhe o resultado de verificação desejado
3. O sistema exibe lista propostas de créditos com o resultado de verificação escolhido, contendo as seguintes informações:
    - Código da proposta de crédito
    - Resultado de verificação
    - CPF do cliente
    - Nome do cliente
4. O caso de uso retorna ao passo 3 do fluxo principal.

Fluxo alternativo 2.2/3.1: Não existem propostas de crédito com o resultado de verificação escolhido.

1. O Sistema exibe mensagem informando que não existem propostas com resultado de verificação escolhido pelo ator.
2. Encerrar caso de uso.

Fluxo alternativo 4.1: A proposta de crédito detalhada possui resultado de verificação igual a “Reprovada”.

1. O ator comunica ao cliente, via e-mail ou telefone, a não aprovação da proposta de crédito.
2. Encerrar caso de uso

2. Analisar contrato

Relação com o modelo de processos: Atividade Analisar contrato, Cancelar contrato de risco, Alterar proposta de crédito e Comunicar não aprovação de contrato de risco.

Ator: Analista de crédito

Tipo: Primário e Essencial

Descrição: O analista de crédito analisa as propostas de contrato, podendo aprová-las, cancelá-las quando houver riscos ou ajustar a proposta de crédito quando for possível a redução de riscos.

Pré-condições: Propostas de contrato cadastradas

Pós-condições: Contrato cancelado ou aprovado

Início: O caso de uso inicia quando o Analista de crédito deseja analisar propostas de contrato cadastradas.

Fluxo principal:

1. O Analista de crédito escolhe a opção para consulta de propostas de contrato.
2. O sistema lista propostas de contrato, contendo:
    - Número do contrato
    - Situação
    - Código da proposta de crédito
    - CPF do cliente
    - Nome do cliente
3. O Analista de crédito escolhe qual proposta deseja visualizar.
4. O Sistema exibe as seguintes informações:
    - Proposta de contrato
        - Número
    - Proposta de crédito correspondente à proposta de contrato
        - Código da proposta
    - Créditos concedidos correspondentes à proposta de crédito
        - Valor total concedido
        - Número de parcelas
        - Valor da taxa de juros
        - Parcelas
            - Data de vencimento
            - Valor a ser pago

            - Juros de atraso
    - Cliente
        - CPF
        - Nome
        - Limite de crédito do cliente
    - Créditos já concedidos ao cliente
        - Valor total concedido
        - Número de parcelas
        - Valor da taxa de juros
        - Parcelas
            - Data a ser realizado o pagamento
            - Valor a ser pago
            - Juros de atraso
            - Valor pago
            - Data de pagamento
    - Opções para:
        - Cancelar contrato de risco
        - Aprovar o contrato
        - Ajustar proposta de crédito para redução de risco
5. O Analista de crédito aciona opção para aprovar contrato.
6. O Sistema altera a situação da proposta de contrato para "aprovado" e encerra o caso de uso

Fluxo alternativo 5.1: Cancelamento de contrato de risco.
Descrição: No passo 5 do fluxo principal, o ator aciona a opção de cancelamento de contrato.

3. O Sistema altera a situação da proposta de contrato para "Cancelado por corresponder a contrato de risco" e a situação dos créditos concedidos é ajustada para "cancelado".
4. Encerrar caso de uso.

Fluxo alternativo 5.2: Alteração de proposta de crédito para redução de risco.
Descrição: No passo 5 do fluxo principal, o ator aciona a opção de ajuste da proposta de crédito.

1. O Sistema solicita o novo valor a ser financiado e o novo número de parcelas.
2. O Analista de crédito informa os dados solicitados.
3. O sistema altera a proposta de crédito com os dados informados e gera novo contrato para reduzir o risco.

4. Ir para o passo 4 do caso de uso.

Fluxo alternativo 2.1: Propostas de contrato inexistentes.

Descrição: No passo 2 do fluxo principal, não existem propostas de contrato cadastradas.

1. O sistema exibe mensagem informando que não existem propostas de contrato cadastradas.
2. Encerrar caso de uso

3. Validar proposta de contrato

Relação com o modelo de processos: Atividades Verificar condições de contrato com o cliente, Aprovar contrato e Cancelar contrato.

Ator: Atendente (iniciante), Cliente.

Tipo: Primário e Essencial

Descrição: O atendente valida com o cliente a proposta de contrato, que pode aprová-la ou cancelá-la.

Pré-condições: Proposta de contrato aprovada pelo analista de crédito

Pós-condições: Proposta de contrato aprovada ou cancelada pelo cliente

Início: O caso de uso inicia quando o ator deseja verificar condições de contrato com o cliente, de acordo com o cliente, propostas de contrato já aprovadas pelo analista de crédito.

Fluxo principal:

1. O Atendente escolhe a opção para validação de propostas de contrato com o cliente.
2. O Sistema lista propostas de contrato com situação "Aprovada", contendo:
    - Número do contrato
    - Situação
    - Código da proposta de crédito
    - CPF do cliente
    - Nome do cliente
3. O Atendente escolhe qual proposta deseja visualizar.
4. O Sistema exibe as seguintes informações:
    - Proposta de contrato
        - Número
    - Proposta de crédito correspondente
        - Código da proposta
    - Créditos concedidos correspondentes
        - Valor total concedido

        - Número de parcelas
        - Valor da taxa de juros
        - Parcelas
            - Data de vencimento
            - Valor a ser pago
            - Juros de atraso
    - Cliente
        - CPF
        - Nome
        - Telefone
    - Opções para Aprovar ou Cancelar o contrato
5. O Atendente entra em contato com o Cliente para apresentar-lhes as características do contrato.
6. O Atendente apresenta as características do contrato para o Cliente.
7. O Cliente aceita o contrato.
8. O Atendente aciona a opção de aprovação de contrato.
9. O sistema altera a situação da proposta de contrato para "Aprovado pelo cliente".

Fluxo alternativo 2.1: Não existem proposta de contrato na situação "Aprovada".

1. O Sistema exibe mensagem informando que não existem propostas de contrato na situação "Aprovada".
2. Encerrar caso de uso.

Fluxo alternativo 7.1: Cliente rejeita características do contrato.

Descrição: No passo 7 do fluxo principal, Cliente não aceita as características do contrato, rejeitando-o.

1. O Atendente aciona a opção de reprovação do contrato.
2. O Sistema altera a situação da proposta de contrato para "Cancelado pelo cliente" e a situação dos créditos concedidos para "cancelado".
3. Encerrar o caso de uso.

Anexo IV – Conceitos de modelagem com notação EPC

Os processos de negócio são modelados em níveis de visão bem definidas. Cada nível é representado por um tipo de modelo que possui objetos representativos adequados para sua visão. O nível superior é o mais abstrato, sendo seus inferiores mais específicos. No nível mais abstrato encontram-se os processos de negócio da organização. Cada processo é detalhado por um conjunto de atividades coordenadas que, por sua vez, são detalhadas ao nível de execução, onde é possível encontrar informações operacionais da atividade.

Este anexo apresenta os diagramas que são utilizados na modelagem de processos de negócio, de acordo com a notação EPC [Scheer, 2000].

1. VAC – *Valued Added* Chain

O diagrama VAC especifica as funções em uma organização as quais influenciam diretamente o real valor agregado da organização. Estas funções podem ser ligadas a outras, de forma a sequenciar as funções e então formar a cadeia de valor agregado [ARIS, 2006].

O diagrama VAC descreve os processos de negócio do ponto de vista mais abstrato. Cada processo contido no modelo possui um ou mais objetivos que agregam valor que garante a existência do negócio. Um modelo VAC pode ser detalhado em outros macro-processos. A cadeia de valor do nível mais alto representa o processo de negócio da organização.

Em um diagrama VAC, as funções podem ser dispostas em uma hierarquia ou similar a uma árvore de função. A orientação do processo subordinado ou superior é sempre ilustrada. Assim como a representação da função subordinada ou superior, um diagrama de cadeia de valor representa as ligações entre as funções e as unidades organizacionais e as funções e objetos informativos [ARIS, 2006].

A Figura 40 exemplifica um modelo VAC. Este modelo possui o macroprocesso "Realizar empréstimos para pessoa física" composto por outros três macroprocessos que estão ligados aos seus respectivos objetivos e indicadores que são extraídos durante a realização do processo. A execução coordenada destes três macroprocessos permitirá ao negócio realizar empréstimos para pessoa física.

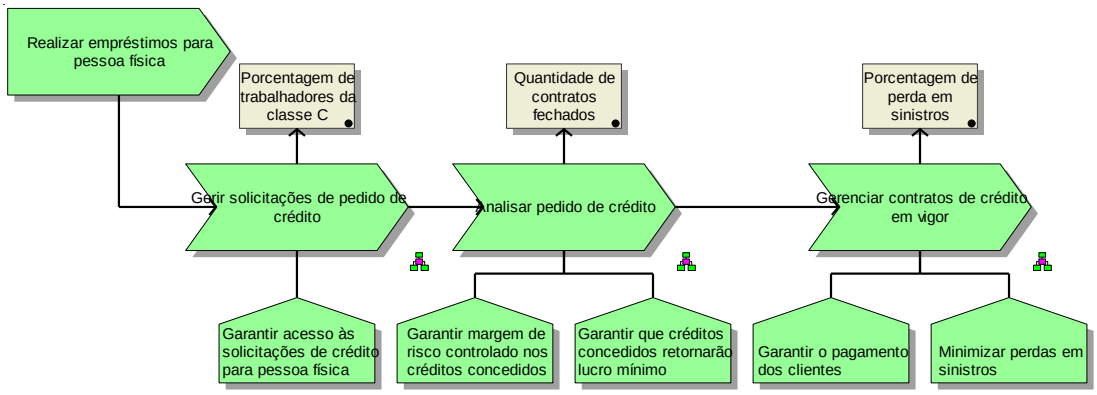


Figura 40 – Exemplo de modelo VAC

A visão de macroprocesso é conhecida como uma visão gerencial devido a sua abstração. Essa visão é moldada durante a primeira fase do ciclo de vida da modelagem de processos de negócio com o propósito de adquirir conhecimento necessário para calcular tempo e custo de um projeto de modelagem de processos do negócio.

2. EPC – Event-driven Process Chain

O diagrama EPC é o modelo central para toda a modelagem de negócio. É um modelo dinâmico que traz consigo os recursos estáticos do negócio (sistemas, organizações, dados, etc.) e os organiza para conceber uma sequência de tarefas ou atividades ('o processo') que agrega valor ao negócio [Davis, 2002]. O modelo EPC pode detalhar tanto um processo previsto na cadeia de valor como uma atividade crítica ou complexa que caracteriza um subprocesso.

O fluxo de trabalho detalhado em um modelo EPC engloba informações como papéis executores e suas unidades organizacionais, raias que organizam as atividades de acordo com seus papéis executores, elementos de interface com outros processos, eventos que demarcam o início e o fim do processo, eventos intermediários que sinalizam circunstâncias importantes para a continuidade do processo, principalmente nos fluxos de decisão, operadores lógicos para inserir regras no fluxo e as próprias atividades que serão executadas ao longo do processo.

O nível de detalhe em um modelo EPC o difere de um simples modelo de *workflow*, permitindo uma leitura clara das atividades do negócio. Isto auxilia no estudo de melhorias e planejamento de automação das atividades.

A Figura 41 ilustra um exemplo de EPC. Este modelo possui atividades executadas pela área de "Crédito e taxas contratuais" e pelo sistema "Crédito direto". As atividades de responsabilidade de cada papel estão respectivamente em suas raias. As linhas que cruzam a linha das raias representam *handoffs* entre os papéis executores do processo.

A leitura deste processo pode ser interpretada da seguinte forma: O processo inicia quando a área de "Crédito e taxas contratuais" recebe a proposta de crédito. Então o sistema "Crédito direto" verifica a situação cadastral do cliente e o limite de crédito do cliente. Se o limite de crédito do cliente não for aprovado, o sistema cancela a proposta e a área de "Crédito e taxas contratuais" comunica ao cliente que o limite não foi aprovado, finalizando o processo. Caso o limite de crédito seja aprovado, o sistema "Crédito direto" calcula a alíquota de imposto para cada tipo de imposto e determina a taxa de juros a ser cobrada do cliente. Então a área de "Crédito e taxas contratuais" analisa o contrato. Caso seja identificado que o contrato é de risco, o sistema "Crédito direto" cancela o contrato de risco, finalizando o processo. Caso não haja necessidade de ajustar o contrato, a área de "Crédito e taxas contratuais" aprova o contrato, encerrando o processo. Caso seja identificada a necessidade de ajuste do contrato, a área de "Crédito e taxas contratuais" altera a proposta de crédito, retornando a proposta para uma nova análise. Ao final do processo, o contrato será cancelado ou efetivado.

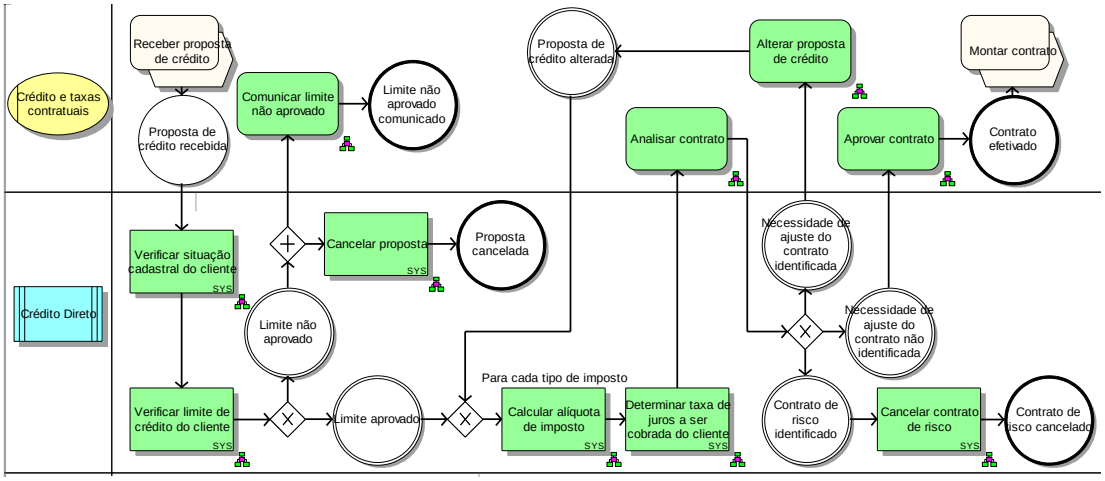


Figura 41 – Exemplo de modelo EPC

O modelo EPC é utilizado para detalhar todos os processos de negócio na cadeia de valor, desde que a modelagem deste nível de detalhamento faça parte do escopo do projeto de modelagem de processos de negócio.

3. FAD – Function Allocation Diagram

O diagrama FAD é um modelo que possui informações sobre uma dada atividade do ponto de vista operacional. É utilizado para apresentar uma visão mais detalhada dos recursos disponíveis e necessários, que são relevantes para as atividades. O FAD também é utilizado para reduzir a complexidade dos processos de negócio, representando elementos como: cargos, áreas, transações e sistemas que suportam a atividade, as entradas e saídas de dados, os documentos, os riscos envolvidos nas atividades, entre outras possibilidades que podem ser criadas pelo modelador [BPM-Advisor, 2009].

A Figura 42 exemplifica um modelo FAD. Esta atividade é executada pelo sistema "Crédito Direto", caracterizando uma atividade automatizada. As informações que a atividade consome são "Proposta de crédito", "Créditos concedidos" e "Cadastro de cliente", sendo esta última extraída da base de dados "BDCliente". O resultado da atividade é a informação "Proposta de crédito" atualizada com um status de aprovação que depende da regra de negócio "Verificação de limite de crédito" e que é implementado de acordo com o requisito de negócio "Verificar limite de crédito do cliente".

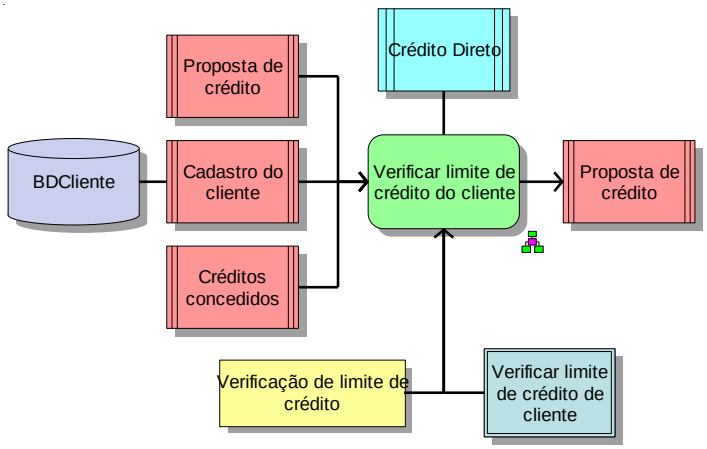


Figura 42– Exemplo de modelo FAD

O modelo FAD deve ser utilizado para representar todas as atividades de um diagrama EPC, desde que a modelagem deste nível de detalhamento faça parte do escopo do projeto de modelagem de processos de negócio.