**UNIVERSIDADE DE SÃO PAULO**

**LEONARDO GASPARINI ROMÃO**

# PROCESSO DE DESCRIÇÃO ARQUITETURAL DE SOFTWARE UTILIZANDO TÉCNICAS DE ENGENHARIA REVERSA

São Paulo

2016

**LEONARDO GASPARINI ROMÃO**

# PROCESSO DE DESCRIÇÃO ARQUITETURAL DE SOFTWARE UTILIZANDO TÉCNICAS DE ENGENHARIA REVERSA

Monografia apresentada ao PECE – Programa de Educação Continuada da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Especialista em Tecnologia da Informação.

Área de Concentração: Tecnologia da Informação

Orientador:
Prof. Dr. Jorge Luis Risco Becerra

**São Paulo**

**2016**

## FICHA CATALOGRAFICA

Romão, Leonardo Gasparini
Técnicas de engenharia reversa em um processo de descrição arquitetural – São Paulo, 2015;
Nº de páginas: 94

Monografia (MBA em Tecnologia da Informação) - Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

Orientador: Prof. Dr. Jorge Luis Risco Becerra.

1.Arquitetura de Software; 2. Engenharia Reversa; 3. Descrição Arquitetural

## AGRADECIMENTOS

Agradeço primeiramente ao Grupo de Fábrica de Software do Laboratório de Tecnologia de Software por me dar a chance de realizar o curso do MBA e ao professor Jorge Luis Risco Becerra por todo o esforço na orientação deste trabalho, e agradeço também aos meus pais, aos professores Ana Claudia Rossi, Juan Felipe Restrepo Naranjo e Leonardo Dominguez Dias, aos meus amigos Laryssa Machado e Íris Xavier, à Ana A. Wertzner e todas as pessoas que me ajudaram a fazer este trabalho.

## RESUMO

A arquitetura de software tem se tornado um fator cada vez mais importante para o desenvolvimento e evolução de novos softwares, tanto pelo fato de alinhar o software ao modelo de negócio ao qual ele está inserido, como também para criar sistemas que sejam adaptáveis as rápidas e constantes mudanças do negócio. Apesar de sua importância, a documentação arquitetural não é um item priorizado pelas equipes de desenvolvimento de software, como equipes de desenvolvimento de softwares *open-source*.

Analisando os softwares de código aberto, tanto as organizações que trabalham para evoluir o software, como as equipes que adaptam ou realizam manutenções nestes softwares não criam ou não disponibilizam informações arquiteturais do sistema, nem utilizam um processo especifico de adaptação baseado em arquitetura, e técnicas que permitam esta evolução de forma que não corrompa sua arquitetura.

Para realizar modificações no software, é comum que as equipes gastem tempo entendendo o software analisando seu código fonte, fazendo com que os desenvolvedores tenham um conhecimento especializado. Entretanto, outros envolvidos no projeto e novos integrantes da equipe terão dificuldade de compreensão do sistema, pois não possuem este conhecimento, para isso, uma forma de compreender o sistema, de forma que outras partes possam discutir sobre, é utilizar a engenharia reversa para criar modelos que representem o código fonte, para que seja possível que as discussões sobre o sistema sejam mais efetivas.

Este trabalho visa propor um processo para construir uma descrição arquitetural contendo modelos que representem a arquitetura do software. A metodologia para construção deste trabalho foi primeiro identificar as informações necessárias e os modelos necessários para construir uma descrição arquitetural na visão computação. Em seguida, foi necessário construir um processo baseando nos requisitos da primeira fase do modelo Horseshoe, aplicando 3 técnicas de engenharia reversa no software: Uma técnica para obter um diagrama de classes, uma técnica para obter um diagrama de caso de uso e uma técnica para obter um diagrama de sequência e por último, aplicar o processo para construir uma descrição arquitetural de um e-commerce de código aberto.

## ABSTRACT

Software architecture has become an increasingly important factor for the development of new software, both because of aligning the software to the business model to which it is inserted, as well as to create systems are adaptable to the rapid and constant business changes. Despite its importance, the architectural documentation is not an item prioritized by software development teams, as open-source development teams.

Analyzing the context of open-source software, both organizations working to evolve the software, as well as the teams that adapt or perform maintenance on these software do not create or provide architectural information system, nor utilize a specific adaptation process based on architecture, and techniques that allow this development in a way that doesn't corrupt its architecture.

To make changes to the software, it is common that teams spend time studying and understanding the software's source code, so that developers have a specialized knowledge. However, other people involved in the project and new team members will find it difficult to comprehend on the system, because they do not have this knowledge, for it is a way to understand the system, so that other parties can discuss, is to use reverse engineering to create models representing the source code, so it's possible that the discussions about the system be more effective.

This paper aims to propose a process to create these models representing the system architecture. The methodology for the construction of this paper was first identify the necessary information and models to create an architectural description in computational viewpoint. Second, it was necessary to create a process thought the first phase of horseshoe model applying three software reverse engineering techniques: A technique to recover a class diagram, a technique to recover an use case diagram, a technique to recover a sequence diagram, and for last , apply the process to create an architectural description of an open-source ecommerce

**SUMÁRIO**

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| **RM-ODP** | *Reference Model – Open Distribuíted Systems* |
| **MVC** | *Model-View-Controller* |
| **BPMN** | *Business Process Model Notation* |
| **UML** | *Unified Modeling Language* |
| **SQL** | *Structured Query Language* |
| **OFG** | *Object Flow Graph* |
| **UML4ODP** | *Use of UML for ODP system* |

# 1 INTRODUÇÃO

## 1.1 Contexto

Os modelos de negócio das organizações mudam rapidamente e constantemente, para se manter no mercado com novas oportunidades de negócio. Com isso, a capacidade de evoluir um software para adequá-lo as mudanças dos modelos de negócio organizacionais em um curto tempo é uma necessidade crítica das organizações, tendo em vista que, os softwares tornaram-se uma das principais ferramentas de apoio aos modelos de negócios das organizações. De acordo com Breivold, Crnkovic e Larsson (2012), há muito tempo, as organizações concentram os custos do ciclo de vida do software na evolução, para atender as mudanças de requisitos e novas oportunidades de negócio**.**

Softwares precisam ser adequados rapidamente para atender o seu papel na organização e ser relevante aos *stakeholders*. Um exemplo de sistemas que possuem a necessidade de se adaptar rapidamente são sistemas de comércio eletrônico, que segundo Laguna e Hernandez (2010), o domínio de aplicações como um e-commerce necessitam de funções gerais com variações específicas para cada organização, sendo assim, estes softwares precisam de uma estrutura que seja modificável.

Para que um software tenha a capacidade de apoiar modelos de negócio que mudam frequentemente, sua arquitetura precisa ser adaptável, pois segundo a afirmação de Breivold, Crnkovic e Larson (2012), a arquitetura é a base de qualquer software, sendo assim, um sistema rígido que não leva as mudanças em consideração, não acompanha as mudanças do modelo de negócio de sua organização e tende a morrer pois o software não atenderá mais ao seu propósito e perderá sua utilidade.

Para implementar mudanças no software, é necessário entender como o software está estruturado, sendo assim, é preciso entender a arquitetura do software. Segundo Breivold, Crnkovic e Larsson (2012), a análise arquitetura possibilita evoluir um software adequadamente ao ambiente organizacional que ele está inserido. Entretanto, entender a arquitetura de um software geralmente é uma tarefa difícil, pois segundo Chadha (2014), apesar de existirem princípios que auxiliam a entender como

o software foi construído como abstração, modularização e a engenharia reversa, as técnicas que são criadas a partir destes princípios são limitadas a entender o código fonte de um software, e não sua arquitetura.

Para analisar a arquitetura de um software, é necessário que existam modelos que representem várias visões do software, permitindo uma análise mais ampla e menos detalhada que o código fonte. Entretanto, a maioria das equipes de desenvolvimento não criam estes modelos, ou não utilizam um padrão para a construção de modelos arquiteturais. Dentro deste cenário estão as equipes de desenvolvimento de projetos de código aberto. Ding et al (2014) apresentou uma pesquisa em que, entre 2000 projetos de código aberto, apenas 108 possuíam alguma documentação referente a arquitetura de software e desses projetos, muitos não tinham uma estrutura ou não estavam detalhados de forma adequada, pois 88.9% destes projetos utilizavam uma linguagem informal.

## 1.2 Problema

Como consequência da falta de documentação arquitetural adequada em projetos *open-source*, saber se um software atende as necessidades do modelo de negócio se torna uma tarefa complexa, porque de acordo com Ding et al (2014), modelos arquiteturais permitem guardar informações de decisão do projeto que tenham relação com o modelo de negócio e fornecer informações entre os *stakeholders* para contribuir com a arquitetura.

A falta de uma descrição arquitetural torna-se um obstáculo para modificar um software, porque é incerto saber os pontos de impacto necessários para adicionar uma nova funcionalidade sem danificar a arquitetura do software. Para passar este obstáculo, Laguna e Hernandez (2010) afirmam que são necessários modelos que expressem a arquitetura de um software possibilitando rastrear as características e o impacto das modificações em um software ao adicionar uma nova função.

Portanto, é um desafio para as equipes de desenvolvimento evoluir um software de código aberto sem danificar a arquitetura, porque o código fonte é o único artefato que expressa fielmente a atual arquitetura de um software de código aberto, portanto, o problema abordado nesta monografia é a necessidade de compreender estes

softwares através de modelos que representem sua arquitetura para auxiliar sua evolução para adequa-lo a diferentes ambientes no qual será inserido.

## 1.3 Objetivo

O objetivo deste trabalho é propor um processo de construção de uma descrição arquitetural de softwares na visão computação do modelo RM-ODP. Este processo utiliza diferentes técnicas de engenharia reversa em diferentes fases do processo para abstrair modelos do código fonte de um sistema de código aberto que forneçam informações que serão utilizadas para descrever a arquitetura do software.

O Segundo passo, é realizar a descrição de um processo para descrever uma arquitetura do software na visão computação, utilizando o framework UML4ODP como modelo para construir a descrição arquitetural que usa as informações recuperadas do software no primeiro passo mais as informações do ambiente no qual ele será inserido. A escolha do UML4ODP como linguagem de especificação é que por utilizar diagramas UML para modelar a arquitetura, serão modelos que provavelmente serão mais precisos ao serem analisados por um stakeholder e construídos por um desenvolvedor.

## 1.4 Justificativa

Para Breivold, Crnkovic e Larson. (2012), é um requisito de qualidade muito forte um software em que sua arquitetura seja capaz de apoiar um modelo de negócio que muda rapidamente. De acordo com os autores, uma organização que possuí softwares que são ineficientes em evoluir, consequentemente irá perder oportunidades de negócio.

Segundo Chadha (2014), os modelos que representam a arquitetura de software são considerados artefatos importantes para sua evolução, porque para evoluir um software, mesmo que possua uma arquitetura fácil de evoluir, é necessário entender sua arquitetura para saber os pontos de impacto quando as alterações forem implementadas.

Entretanto, as conclusões feitas pelo trabalho de Ding et al (2014) indicam que na maioria das comunidades de software de código aberto, a documentação

arquitetural não é bem vista, porque muitas comunidades são compostas por pessoas que em alguns casos são freelancers que possivelmente não tiveram o treinamento adequado e não precisam seguir padrões corporativos, além disso, os resultados da pesquisa mostram que em muitos casos não foi utilizado em nenhum momento linguagens de descrição arquitetural. Portanto, estas conclusões contradizem a adoção da engenharia de software tradicional e a documentação destes projetos não são suficientes para entender o software e sua arquitetura.

Segundo a ISO/IEC 14764 (2006), quando a documentação é insuficiente e o código fonte é o único artefato para manutenção do sistema, é recomendado realizar a engenharia reversa. Tonella (2005), apresenta uma técnica de engenharia reversa em softwares orientados a objetos para construção de modelos UML, pois são modelos mais convencionais utilizados pelas equipes de desenvolvimento de software.

Porém, a UML por si só é limitada para construir modelos arquiteturais. Mesmo assim, a UML era adaptada por diversas equipes que utilizavam a linguagem para representar a arquitetura dos sistemas. Com a grande demanda em utilizar a UML para representar arquitetura de softwares, foi criado o UML4ODP ou Uso da UML para Especificação de Sistemas ODP e a ISO/IEC 19703, permitindo criar modelos arquiteturais utilizando a UML (Vallecilo, 2011).

A parte da visão computação deste padrão será utilizada neste trabalho, porque, utilizar linguagens de descrição arquitetural, padroniza os modelos arquiteturais, facilitando o entendimento entre diferentes stakeholders.

## 1.5 Metodologia

Este trabalho terão as seguintes fases:

1. Primeira Fase: Levantamento bibliográfico dos assuntos necessários para desenvolvimento do trabalho.
2. Segunda Fase: Montar o processo de construção arquitetural utilizando técnicas de engenharia reversa e especificação de arquitetura de software analisadas durante a fase um, e utilizar as normas ISO/IEC 19793, ISO/IEC

14794 e a ISO/IEC 42010, como base para criar os artefatos necessários para obter uma descrição arquitetural.

3. Terceira Fase: Aplicação do processo em um software de comércio eletrônico que será inserido dentro de uma organização, apresentando seus desafios, necessidades e objetivos e como o trabalho pretende contribuir sobre este cenário.
4. Quarta fase: Contextualização e aplicação do processo construído e apresentar os resultados.
5. Quinta fase: Considerações finais e trabalhos futuros.

## 1.6 Estrutura do Trabalho

Esta monografia está organizada nos seguintes capítulos:

No capítulo 1, é apresentado a introdução deste trabalho, contextualizando o escopo em que este trabalho utiliza, a definição do problema deste cenário, o objetivo deste trabalho que é propor uma solução do problema definido, e a justificativa de trabalho, apresentando sua contribuição para a literatura.

O capitulo 2 apresenta os conceitos dos assuntos abordados neste trabalho, para fundamentação teórica para a proposta de solução que será apresentada, entre os assuntos abordados estão:

- Engenharia Reversa;
- Arquitetura de Software;
- Descrição Arquitetural;
- Framework UML4ODP;

A partir do capitulo 3, é descrito o processo com as atividades e tarefas especificado na linguagem BPMN, para resolução do problema apresentado no primeiro capítulo.

O capitulo 4 apresenta o roteiro operacional do processo apresentado no capitulo 3. Os resultados que são obtidos a partir da aplicação deste roteiro é a descrição arquitetural apresentada no apêndice deste trabalho.

Por último, o capítulo 5, apresenta a conclusão deste trabalho, mostrando considerações dos resultados que foram obtidos e trabalhos futuros sobre este

estudo, seguido pelo apêndice contento a descrição arquitetural resultante deste trabalho.

# 2 A ENGENHARIA REVERSA E ARQUITETURA DE SOFTWARE

## 2.1 Engenharia Reversa

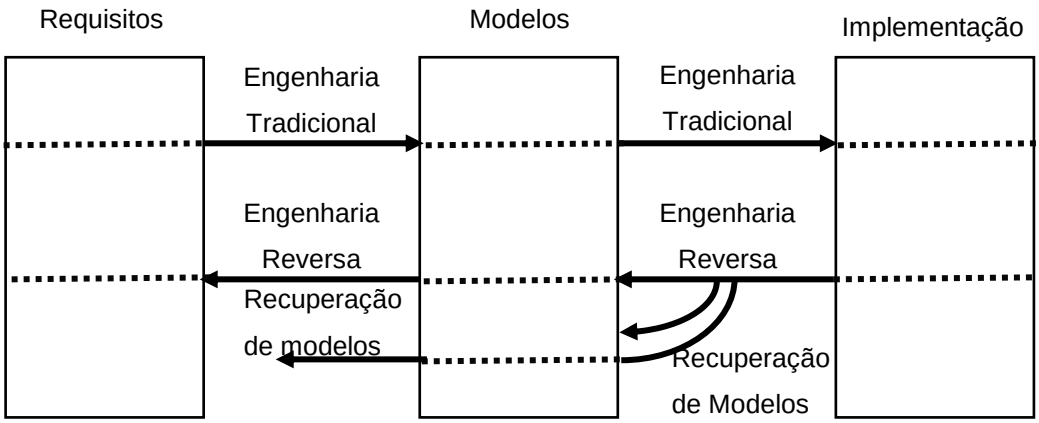
### 2.1.1 Definição e Objetivos

Segundo a ISO/IEC 14764 (2016), a engenharia reversa é a recomendação para documentar softwares quando o código fonte é a representação mais precisa que se possuí. A engenharia reversa é necessária, pois softwares que possuem um longo tempo de vida sofrem mudanças para atender novos requisitos de negócio, o que faz com que estes softwares sejam diferentes do projeto inicial que foi pensado, mas a documentação destes softwares não é criada ou atualizada quando há mudanças. Tripathy e Naik (2014) afirmam que existem fatores que implicam na necessidade de realizar a engenharia reversa, estes fatores são:

- Os programadores originais deixaram a organização;
- A linguagem de implementação se tornou obsoleta, sendo necessário migrar para uma nova linguagem;
- Não há documentação suficiente sobre o sistema;
- O modelo de negócio da organização depende do software, e muitas pessoas não sabem como o software funciona;
- A empresa adquiriu o sistema como parte de uma aquisição maior e carece de acesso ao código fonte inteiro;
- O sistema requer modificações ou melhorias;
- O sistema não opera como o esperado;

Já para Pereira, Martinez e Favre (2011), a engenharia reversa é o processo de analisar artefatos de software disponíveis como requisitos, modelos ou códigos para extrair informações e construir modelos amplos que estejam coerentes com o código fonte. A figura 1 apresenta a diferença entre a engenharia reversa e as metodologias tradicionais de desenvolvimento de software.

**Figura 1 - Diferença entre engenharia tradicional e engenharia reversa.**




**Fonte: Adaptado de Tripathy e Naik (2014).**

Conforme a figura mostra, os modelos tradicionais de desenvolvimento possuem um processo de implementar o código fonte a partir de uma documentação composta por modelos construídos de acordo com os requisitos identificados. Já a engenharia reversa possuí um processo ao contrário, onde a partir do código fonte, são recuperadas informações para documentar um software através da construção de modelos arquiteturais ou requisitos de software.

De acordo com Tripathy e Naik (2014), tanto o processo de engenharia tradicional de software e o processo de engenharia reversa são separadas em três etapas principais:

- Requisitos, onde é definido o que o software deve fazer;
- Modelos, onde é definida a estrutura do software;
- Implementação, onde é construído o código fonte, e a realização dos testes;

### 2.1.2 Técnicas

Segundo Tripathy e Naik (2014), as técnicas que são utilizadas para a engenharia reversa são:

- **Análise Léxica:** É o processo de analisar uma ou mais instruções de um código fonte, para identificar operações, números, símbolos e palavras reservadas. A

análise léxica é realizada por um compilador, transformando os caracteres inseridos em tokens que sejam referentes à gramatica que o compilador entenda e possa realizar a análise sintática. Esta técnica auxilia a primeira etapa do processo de engenharia reversa, identificando a estrutura de uma instrução;

- **Analise Sintática:** É a forma mais complexa de automatizar a análise de um programa é analisar sua sintaxe. Compiladores analisam se os tokens criados a partir da análise léxica estão de acordo com a gramatica pré-definida no compilador. Esta técnica permite identificar declarações e expressões de uma instrução, Esta técnica auxilia a primeira etapa do processo de engenharia reversa, identificando a gramatica de uma instrução;
- **Analise de Fluxo de Controle:** É a técnica utilizada para analisar estaticamente qual a sequência de instruções de um código fonte. Esta técnica auxilia a segunda etapa do processo de engenharia reversa, identificando o fluxo de instruções que formam uma função;
- **Analise de Fluxo de Dados:** É utilizado para analisar como os dados são transformados entre as entradas e saídas de instruções e funções. Segundo Pereira, Martinez e Favre (2011), Autores adaptaram a técnica de fluxo de dados para gerar diagramas UML como diagramas de classe, objeto, iteração, estado e pacotes. Esta técnica auxilia a primeira segunda e terceira etapa do processo de engenharia reversa, identificando as entradas e saídas entre instruções e funções;
- **Divisão do programa:** É a técnica para dividir linhas de instrução de um código fonte para melhorar visualmente sua estrutura e compreensão, essa analise separa uma função em partes contendo uma série de instruções, caso seja necessário modificar apenas parte da função. Esta técnica auxilia a primeira e quinta etapa do processo de engenharia reversa, separando as instruções em funções e funções em aplicações;
- **Visualização:** Essa técnica é utilizada para representar graficamente o software, sendo utilizada principalmente para apresentar componentes e como estes componentes estão ligados. Esta técnica auxilia a quinta e sexta etapa do processo de engenharia reversa, identificando as aplicações e os componentes do software;

- **Métricas de software:** Essa técnica é utilizada controlar o processo de engenharia de software de um código fonte, medindo elementos como complexidade ciclomática e pontos por função. Em linguagens orientadas a objetos, outras métricas foram criadas, sendo elas:
    - Números de métodos por classe**.**
    - Responsabilidade da classe
    - Falta coesão nos métodos de uma classe
    - Acoplamento entre objetos de uma classe
    - Profundidade da arvore de herança
    - Número de classes filho de uma classe;

  Esta técnica auxilia a quinta e sexta etapa do processo de engenharia de software, servindo como medidas para avaliar as aplicações e a arquitetura de um software.

## 2.2 Arquitetura de Software

### 2.2.1 Definição e Objetivos

Segundo a ISO/IEC 42010 (2011), a arquitetura de um software consiste nos conceitos fundamentais ou propriedades de um sistema em seu ambiente incorporado em seus elementos, relações e nos princípios da sua concepção e evolução. Segundo a norma, apesar de não existir uma simples definição sobre quais são os conceitos fundamentais de um software, a norma afirma que eles podem ser:

- Elementos ou componentes de um software;
- Como os elementos do sistema são organizados e relacionados;
- Princípios da organização ou construção do software;
- Princípios que governam a evolução do software ao longo do seu ciclo de vida.

Para Rozansky e Woods (2005), todo software possuí uma arquitetura, entretanto isso não significa que toda arquitetura de um software está documentada, de fácil compreensão ou de acordo com as necessidades dos stakeholders.

### 2.2.2 Elementos Arquiteturais

Segundo Rozansky e Woods (2005), Elementos arquiteturais são itens fundamentais que devem ser considerados na construção de um sistema, tais como bibliotecas, subsistemas ou até mesmo outros sistemas, sendo que, eles precisam ter uma série de responsabilidades, interfaces, serviços e limites claramente definidas. Um elemento arquitetural deve possuir os seguintes atributos chaves:

- Uma série de responsabilidades claramente definidas
- Um limite claramente definido
- Uma série de interfaces claramente definidas que caracterizem os serviços que um elemento arquitetural fornece a outros elementos arquiteturais

## 2.3 Descrição Arquitetural

### 2.3.1 Definição e aplicações

Uma descrição arquitetural é o produto de trabalho que expressa a arquitetura de um software (ISO/IEC 42010: 2011). De acordo com a norma, a descrição arquitetural é um artefato, que contribui no entendimento do propósito de um sistema e as principais propriedades referentes ao seu comportamento, composição e evolução, pois são fatores que afetam questões como a viabilidade, utilidade e manutenção do sistema.

Para Rozanky e Woods (2005), a arquitetura de um software pode ser complexa, e a descrição arquitetural é usada para descrever a arquitetura de uma forma que seja possível entender esta complexidade. Sendo assim, qualquer artefato que é utilizado para auxiliar a apresentação da arquitetura aos stakeholders, faz parte de uma descrição arquitetural.

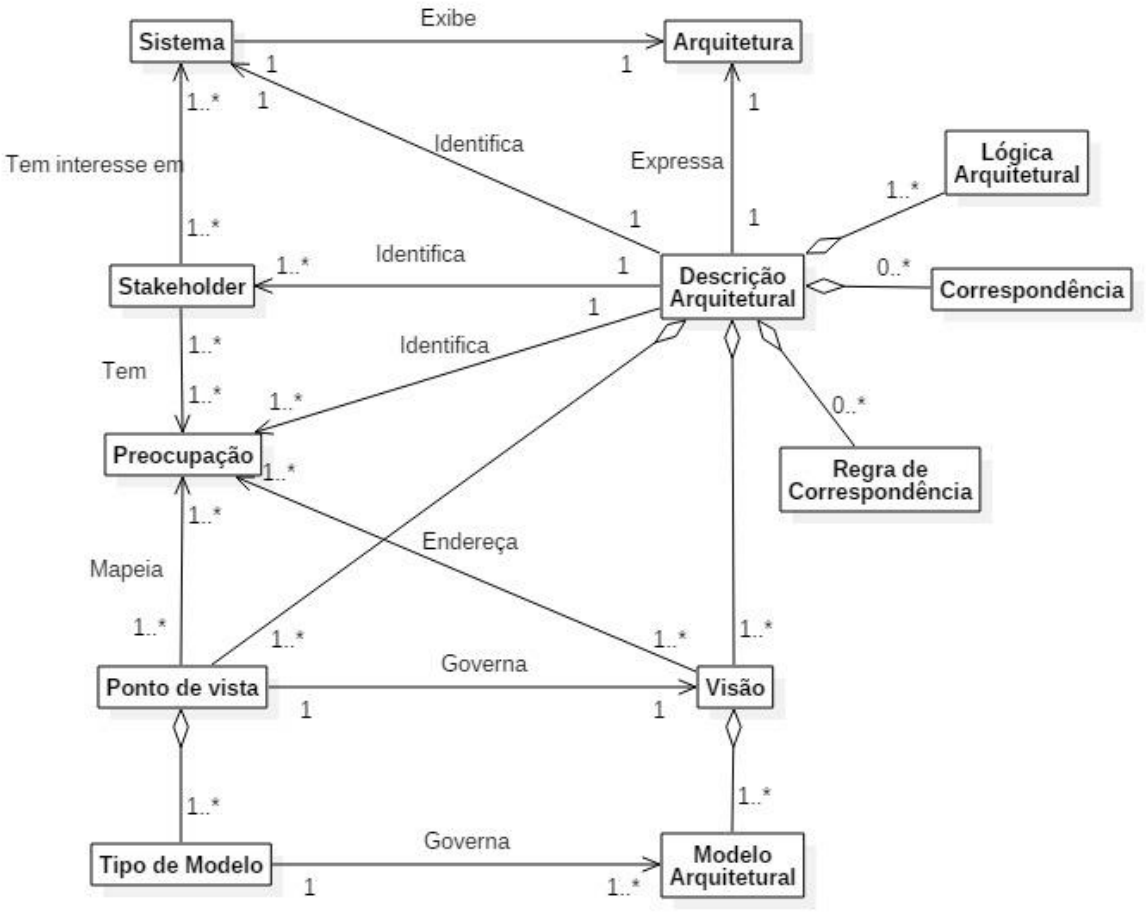
Segundo a ISO/IEC 42010 (2011) as descrições arquiteturais são usadas pelos s*takeholders* para criar, utilizar e gerenciar a construção, funcionamento e evolução de softwares para melhorar a comunicação e cooperação, habilitando os envolvidos

no projeto a trabalhar de forma integrada e visualmente util. As descrições não estão limitadas entre os exemplos de utilização informados pela norma. Alguns dos exemplos fornecidos de utilização de descrições arquiteturais são:

- Base do modelo do Sistema e atividades de desenvolvimento;
- Base para analisar e validar alternativas de implementação da arquitetura;
- Documentação para desenvolvimento e manutenção;
- Documentar aspectos essenciais de um sistema como uso e ambiente envolvido;
- Princípios, premissas e restrições para guiar futuras mudanças;
- Pontos de flexibilidade e limitações do sistema com relação a futuras mudanças;
- Decisões arquiteturais, suas justificativas e implicações;
- Comunicação entre partes envolvidas sobre o desenvolvimento, produção, implantação, operação e manutenção de um sistema;
- Comunicação entre clientes, adquirentes, apoiadores e desenvolvedores como parte do contrato de negociações;
- Base para revisão, analise e validação de um sistema através de seu ciclo de vida;
- Planejar para transição de uma arquitetura legada para uma nova arquitetura;
- Apoio para atividades de planejamento, agendamento e cobrança;

A figura 2 apresenta o modelo conceitual da descrição arquitetural.

**Figura 2 - Modelo Conceitual da descrição arquitetural**




**Fonte: ISO/IEC 42010**

De acordo com a figura, uma arquitetura de software é diferente de uma descrição arquitetural, sendo uma descrição arquitetural um produto de trabalho, composto por conceitos e propriedades, que expressam a arquitetura de um software exibida pelo sistema. Cada um dos elementos do modelo conceitual são elementos que quando analisados para entender a arquitetura de um sistema, são úteis para a construção de uma descrição arquitetural.

### 2.3.2 Visões

Segundo a norma ISO/IEC 42010 (2011), uma visão é uma maneira de se representar um sistema através de modelos que atendam determinadas preocupações dos stakeholders. Para Rozansky e Woods (2005), não é possível

capturar as características funcionais e propriedades de qualidade de um sistema complexo com apenas um único modelo que seja entendido e útil por todos os stakeholders. Para isso, uma arquitetura é descrita em visões separadas, porém relacionadas entre si, sendo que cada visão analisa aspectos e problemas diferentes de uma arquitetura.

Pontos de Vista

De acordo com a ISO/IEC 42010 (2011), um ponto de vista são conceitos e padrões que um modelo arquitetural deve seguir para representar adequadamente uma visão. Uma visão endereça uma ou mais preocupações dos stakeholders de forma que estejam de acordo com um ponto de vista. Um ponto de vista deve especificar os seguintes itens:

- Um ou mais interesses estruturados pelo ponto de vista;
- *stakeholders* genéricos para os interesses estruturados pelo ponto de vista;
- Um ou mais tipos de modelos usados neste ponto de vista;
- Para cada modelo identificado, as linguagens, notações, convenções, técnicas de modelagem, métodos analíticos e/ou outras operações para ser usadas em um modelo deste tipo;
- Referencias para suas fontes;

Para Vallecilo et al. (2011), os pontos de vista têm o propósito de quebrar a complexidade de especificar um sistema em peças separadas, utilizando técnicas diferentes para especificar modelos que sejam familiares para os stakeholders específicos de cada visão e paralelizar as atividades entre equipes diferentes.

### 2.3.3 Modelos Arquiteturais

Um modelo arquitetural usa convenções de modelagem apropriadas para as preocupações endereçadas à uma visão. Essas convenções são especificadas por

um tipo de modelo que governa aquele modelo. Dentro de uma descrição arquitetural, um modelo arquitetural pode ser parte de uma ou mais visões. (ISO/IEC 42010, 2011).

Para Rozansky e Woods (2005), Modelos arquiteturais são representações de um ou mais aspectos de um software, que sejam endereçados a uma visão. Os autores citam algumas razões para se construir modelos, das quais são:

- Trazer precisão e foco nos elementos importantes para uma situação
- Agir como mediação para comunicação, ajudando a explicar a arquitetura a outras pessoas envolvidas.
- Ajudam a analisar situações permitindo isolar elementos chaves e entender seus inter-relacionamentos.
- Auxiliam na organização de processos, equipes e entregáveis.

### 2.3.4 Correspondência

De acordo com a norma 42010 (2011), uma correspondência expressa um relacionamento entre elementos de uma descrição arquitetural, e são utilizadas para representar relações de uma descrição arquitetural ou entre descrições arquiteturais. Modelos, visões, pontos de vista possuem correspondências entre si. Geralmente uma correspondência é governada por uma regra de correspondência.

### 2.3.5 Regra de Correspondência

Segundo a norma ISO/IEC 42010 (2011), uma regra de correspondência força um relacionamento dentro de uma descrição arquitetural ou entre descrições arquiteturais. As correspondências entre modelos e entre visões de uma descrição arquitetural devem estar de acordo com as regras especificadas por uma linguagem ou framework arquitetural.

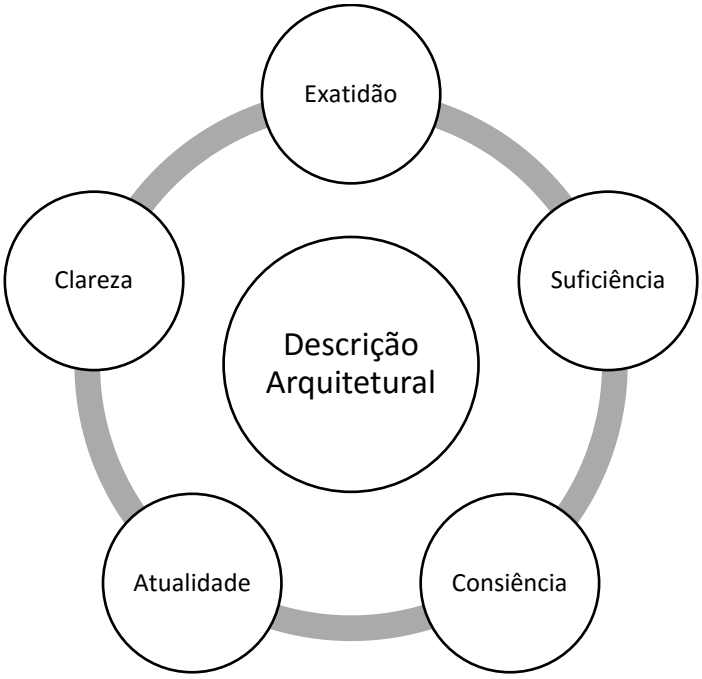
### 2.3.6 Lógica arquitetural

Segundo a norma ISO/IEC 42010 (2011), a lógica arquitetural expressa as justificativas, explicações ou motivos para uma decisão arquitetural ter sido escolhida ao invés das alternativas propostas.

### 2.3.7 Atributos de qualidade

De acordo com Rozansky e Woods (2005), uma descrição arquitetural eficiente deve balancear seis propriedades: Exatidão, Suficiência, Consciência, Clareza, atualidade e precisão. A figura 4 apresenta os atributos de qualidade de uma descrição arquitetural.

**Figura 3 - Qualidade de uma descrição arquitetural eficiente**




**Fonte: Adaptado de Rozansky e Woods, 2005.**

Conforme a figura mostra, para uma descrição arquitetural ser eficiente, ela precisa balancear as seguintes propriedades: (Rozansky e Woods, 2005)

- **Exatidão:** É considerado o mais importante atributo de qualidade da descrição arquitetural, onde as informações precisam estar exatas em representar como a arquitetura irá atender as necessidades e interesses dos s*takeholders*;
- **Suficiência:** A descrição arquitetural precisa estar detalhada o suficiente para responder questões importantes sobre a arquitetura do software. Se a descrição arquitetural não tiver informações suficientes, será um obstáculo realizar decisões arquiteturais antes que o sistema esteja no ciclo de vida do desenvolvimento;

- **Consciência:** A descrição arquitetural precisa ser mais objetiva e simples em expressar os elementos importantes da arquitetura. Entretanto, decidir quais elementos são importantes e o detalhe deles dependem de diversos fatores como:
    - Capacidade e experiência dos stakeholders;
    - Extensão caso a equipe não esteja familiarizado com a tecnologia
    - Dificuldade do problema que está sendo analisado
    - Quanto tempo e recurso você tem disponível para criar a descrição arquitetural.
- **Clareza:** A descrição arquitetural precisa ser entendida por todas as classes de stakeholders. O conceito de ponto de vista é útil para auxiliar neste ponto.
- **Atualidade:** A descrição arquitetural precisar estar de acordo com as mudanças feitas na arquitetura, para isso, a descrição arquitetural tem que ter um tamanho aceitável para que as mudanças não sejam complexas.
- **Precisão:** uma descrição arquitetural precisa descrever a arquitetura precisamente para que o sistema seja modelado e implementado, e se não for feito direito, a precisão pode se tornar o contrário de consciência

## 2.4 UML4ODP

### 2.4.1 Linguagens de descrição arquitetural

Linguagem de descrição arquitetural, segundo a ISO/IEC 42010 (2011), é um mecanismo criado através da construção de conceitos utilizados em uma descrição arquitetural e tem por objetivo expressar as convenções e práticas comuns para construir descrições arquiteturais para diferentes comunidades e domínios de aplicação. Segundo a norma, uma linguagem de descrição arquitetural precisa especificar:

- A identificação de um ou mais interesses expressados pela linguagem de descrição arquitetural;
- A identificação de um ou mais stakeholders com seus interesses expressados;

- Os tipos de modelos implementados que mapeie os interesses;
- Os pontos de vista que serão utilizados;
- Regras de correspondências;

### 2.4.2 Contexto

UML4ODP ou Uso da UML para sistemas especificados em ODP é, segundo a ISO/IEC 19703 (2008), uma linguagem de descrição arquitetural que foi criada para atender o crescimento da adoção do Modelo de referência de processamento aberto distribuído (RM-ODP), utilizando a Linguagem de Modelagem Unificada (UML) para expressar a especificação de sistemas que utilizam o modelo ODP.

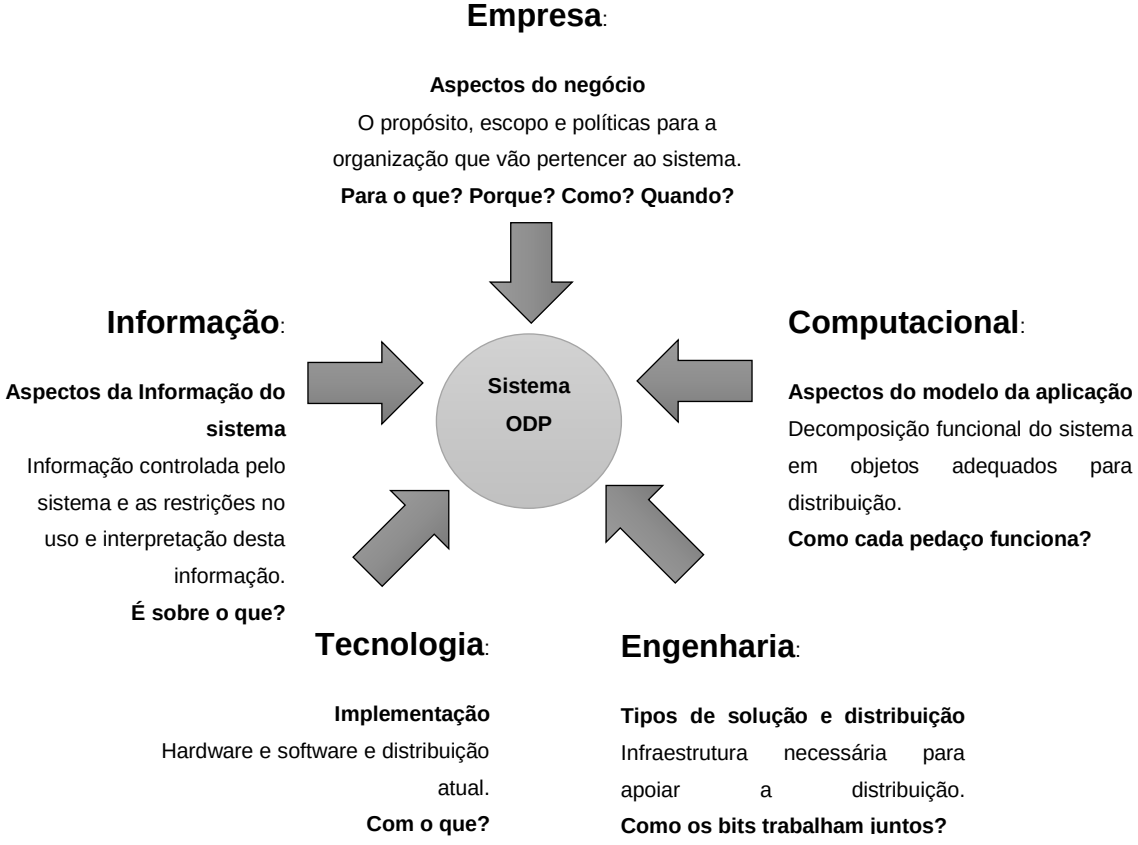
Para Rozansky e Woods (2005), provavelmente a UML é a forma mais prevalente para criar uma descrição arquitetural. A UML tem algumas vantagens, incluindo a sofisticação de algumas de suas notações e sua flexibilidade e extensibilidade, e como a UML é muito utilizada, muitos stakeholders não terão problemas em entende-la, já que notações mais complexas e menos difundidas, serão de difícil entendimento e acompanhamento pelos stakeholders de negócio.

De acordo com Vallecillo et al (2011), o problema da UML por si só é que ela não suporta a separação de preocupações e interesses que existem nos pontos de vista de uma descrição arquitetural, entretanto, equipes de desenvolvimento adaptavam a UML para torna-lo mais compatível com o ODP. Vendo isso, foi criado este padrão, que providencia um perfil que mapeia os conceitos do ODP para a notação UML.

### 2.4.3 Pontos de Vista

Os modelos criados a partir da utilização do UML4ODP são baseados nos conceitos do RM-ODP. O RM-ODP utiliza cinco pontos de vista para especificar a arquitetura de um sistema, como mostra a Figura 4:

**Figura 4 - Pontos de Vista do RM-ODP**




**Fonte: ISO/IEC 19793:2008**

Conforme mostra a Figura 4, cada ponto de vista aborda um grupo de interesses do sistema e seu objetivo é auxiliar a responder as perguntas correspondentes à sua visão do sistema, segundo o modelo RM-ODP, um sistema é especificado seguindo 5 pontos de vista, cada um com sua respectiva linguagem de ponto de vista que é usado para especificar o sistema. (ISO/IEC 19793, 2008).

- **Ponto de vista empresa**: Este ponto de vista foca no escopo do sistema, propósito do sistema que é definido pelo comportamento especificado do e políticas capturam futuras restrições do comportamento entre o sistema e seu ambiente, ou entre as decisões de negócio dos donos do sistema. Sua linguagem providencia os conceitos necessários para modelar um sistema ODP no contexto do negócio da organização em que ele opera. Sua

especificação, é modelar objetos empresa, as comunidades do ambiente e papéis envolvidos.

Para Vallecilo et al (2011), os stakeholders que precisam estar satisfeitos com a especificação da visão empresa são os donos do processo de negócio que será apoiado e os gerentes responsáveis pelas políticas operacionais;

- **Ponto de vista informação:** Este ponto de vista foca nos tipos de informação controlados e usados pelo sistema. Sua linguagem apresenta os componentes individuais e a comunicação entre eles através de um entendimento comum das informações que são trafegadas. Sua especificação abrange uma série de esquemas:
    - O esquema invariante, que expressa a estrutura, tipos e os relacionamentos entre os objetos informação,
    - O esquema estático, que expressa o estado dos objetos informação em um determinado tempo;
    - O esquema dinâmico que especifica como a informação pode evoluir durante a operação do sistema e quais são os estados que um objeto informação pode ter.

  Segundo Vallecillo et al (2011), a visão informação foca em quais informação que o software irá manipular e não com as interfaces realizarão essa manipulação, nem as tecnologias que definem como os dados serão guardados. Portanto, o objetivo deste ponto de vista é ter um dicionário de dados para todas as partes. Os stakeholder que tem mais interesses por esta visão são os que trabalham com banco de dados;

- **Ponto de vista computação:** Este ponto de vista foca em expressar a composição funcional do sistema através de uma série de serviços que se interagem por interfaces, e qual o comportamento destes serviços. Sua especificação modela objetos computacionais que são funções individuais realizadas pelo sistema e a interação entre estes objetos através de interfaces. Para Vallecillo et al. (2011), o objetivo da visão computação é expressar um modelo com as funcionalidades básicas do software, os serviços oferecidos e como estes serviços são construídos e estão conectados. A visão computação permite a reutilização da arquitetura, separando as funcionalidades de plataformas ou tecnologias;

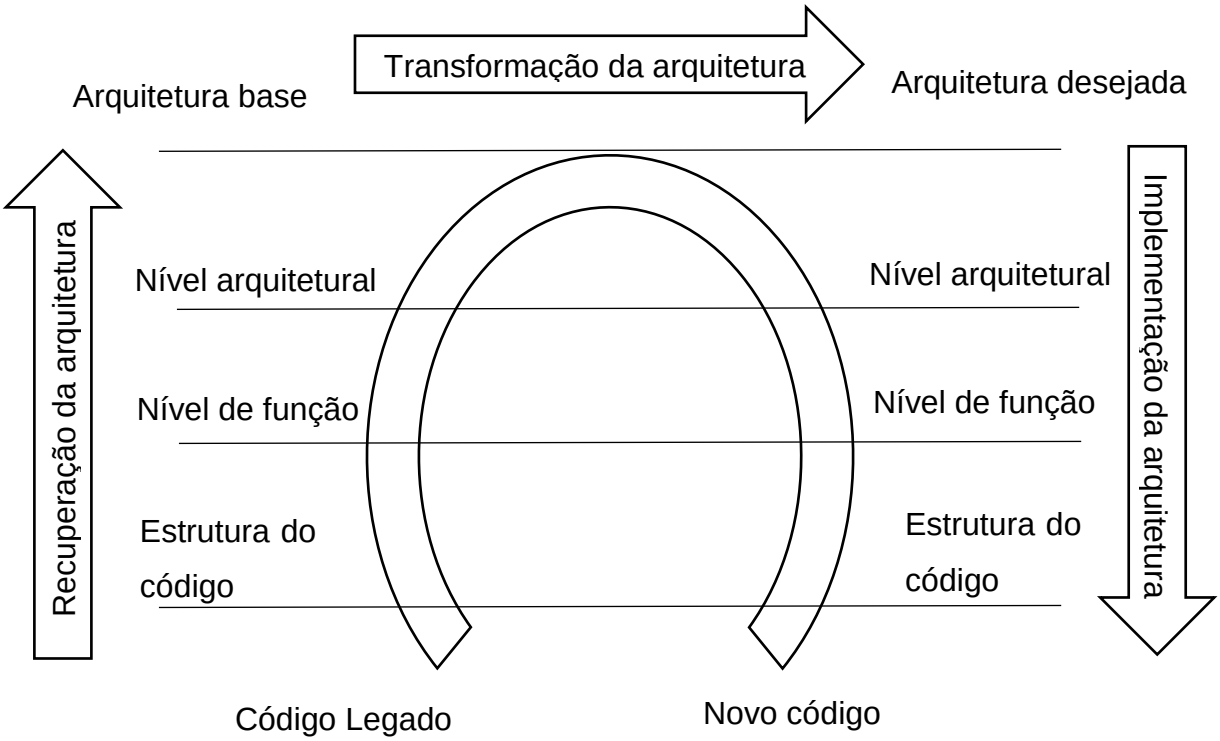
- **Ponto de vista da engenharia:** Este ponto de vista foca na infraestrutura necessária para apoiar o sistema, concentrado em como os objetos se interagem. Sua especificação é definir os mecanismos necessários para suportar as funções do sistema fazendo uso das tecnologias especificadas na visão tecnologia.

  Segundo Vallecillo et al. (2011), o ponto de vista engenharia é endereçado aos projetistas interessados na infraestrutura do sistema, pois este ponto de vista foca nos mecanismos para distribuir os objetos do sistema;
- **Ponto de vista tecnologia:** Este ponto de vista foca na escolha dos fornecedores e tecnologias para apoiar a infraestrutura do software expressada na visão engenharia. Sua especificação modela a configuração de componentes de hardware e software para implementação, restringindo custos e disponibilidade dos objetos de tecnologia;

## 2.5 Relação entre a engenharia reversa e a arquitetura de software.

A literatura há muitos anos, contribui com modelos para construção de arquiteturas de software, a partir da transformação e migração de arquitetura de software, começando por softwares legados. Um dos modelos criado com este propósito modelos é o modelo *horseshoe* apresentado na figura 5.

**Figura 5 - Modelo Horseshoe**




**Fonte: Adaptado de Tripathy e Naik, 2014**

Segundo (Tripathy e Naik, 2014), O modelo *Horseshoe* descreve um processo de três etapas para realizar a reengenharia de uma arquitetura de um software, sendo as três etapas:

- Recuperação de arquitetura: Representada pelo lado esquerdo da figura, esta etapa consiste em utilizar técnicas de engenharia reversa e o princípio de abstração para representar a arquitetura do software a partir do código fonte e avaliar se a arquitetura está de acordo com atributos de qualidade do projeto. Para recuperar a arquitetura do software, é necessário expressar o código fonte nos seguintes modelos:
    - Estrutura de código: Representa a estrutura das instruções, estilos de codificação e fluxo da aplicação.
    - Representação de função: Representa o código fonte, em nível de funções e os dados manipulados por estas funções, a UML pode ser utilizada para esta representação

    - Representação Arquitetural: Tem o objetivo de representar padrões de projetos, estilos arquiteturais e conceitos envolvidos no código fonte, sua representação pode ser feita através de modelos arquiteturais.
- Transformação de arquitetura: Representada pela seta do topo da figura, consiste em transformar a arquitetura atual na arquitetura desejada usando o princípio de alteração.
- Implementação da nova arquitetura: Representada pelo lado direito da figura, esta etapa consiste em realizar um desenvolvimento baseado em arquitetura, construindo o novo código.

A primeira etapa do modelo *horseshoe* apresenta uma série de representações que são necessárias para recuperar a arquitetura de um sistema, estas representações são necessárias pois não é possível compreender a arquitetura de um sistema diretamente pelo código fonte. Para construir modelos que atendam cada representação, são necessárias diferentes técnicas para construir estes modelos, sendo estas técnicas de engenharia reversa.

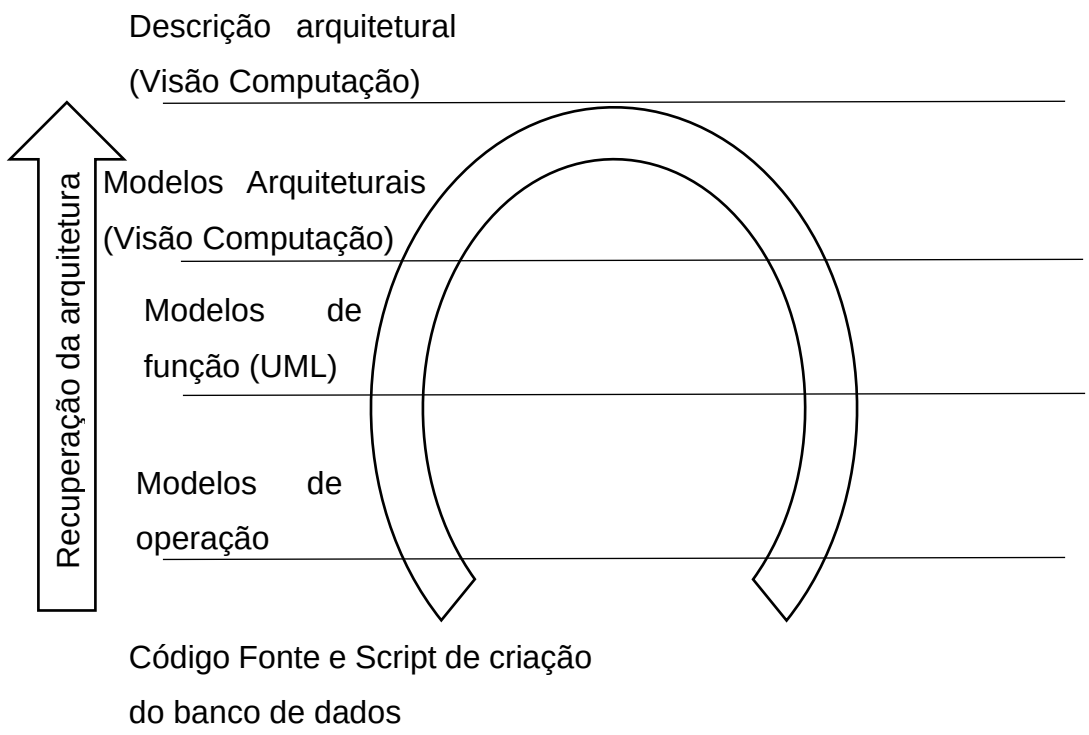
# 3 PROCESSO DE DESCRIÇÂO ARQUITETURAL

Esta seção descreve o processo propostas neste trabalho, com suas atividades, tarefas e como foi construído. O processo é especificado e modelado utilizando a notação BPMN e baseado na primeira etapa do modelo *Horseshoe*, que consiste em expressar a arquitetura de um software, construindo modelos arquiteturais a partir do código fonte. Neste processo, os modelos arquiteturais construídos são do ponto de vista da computação do RM-ODP.

A construção deste processo, foi realizada da seguinte forma:

1. Construir um processo genérico: Primeiro, foi realizada a construção de um processo genérico para construir uma descrição arquitetural a partir do código fonte. Esse processo genérico define os objetivos das atividades e artefatos do processo. Esse processo genérico definiu-se com base no modelo *Horseshoe*, o produto de trabalho desta etapa é explicado na figura 6
    - Definir os requisitos: Os requisitos deste trabalho foram identificados a partir de quais modelos eram necessários para realizar as abstrações do modelo *Horseshoe* adaptado na Figura 6;
    - Definir as técnicas e ferramentas: Identificar técnicas que utilizem e resultem nos modelos identificados na primeira etapa: Nesta etapa, foram pesquisadas quais técnicas poderiam ser utilizadas para abstrair um modelo para outro modelo, seguindo a ordem do modelo *Horseshoe* adaptado. Nesta etapa foram definidas a ordem das técnicas que seriam utilizadas para aplicar o processo;
    - Definir os artefatos: Foram definidas as características dos artefatos resultantes deste processo: Nesta etapa foram definidos quais as propriedades do artefato gerado por este processo, no caso a descrição arquitetural;

**Figura 6 - Adaptação do modelo Horseshoe para o modelo proposto**



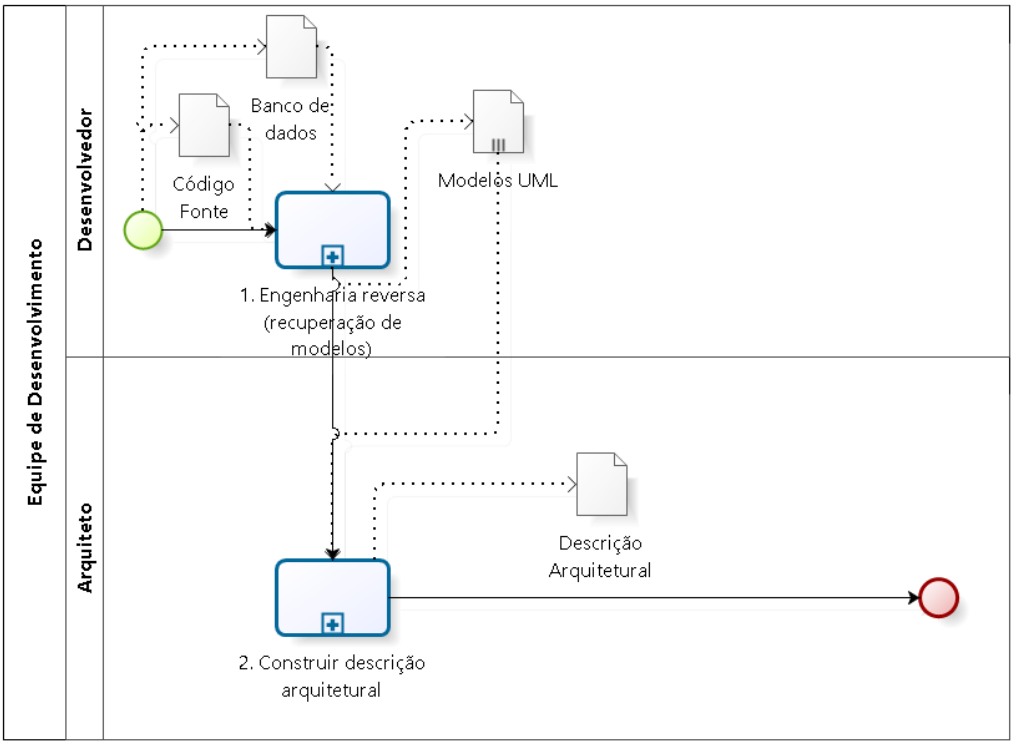

**Fonte: Elaborado pelo autor**

Como a Figura 6 mostra, o processo tem o objetivo de gerar modelos dos seguintes tipos:

- Modelos de operação: São modelos que não possuem um nome especifico, pois, cada técnica utiliza diferentes modelos que são necessários porque todas as técnicas de engenharia reversa utilizadas neste trabalho não constroem modelos UML direto do código fonte, sendo modelos intermediários de cada técnica. Estes modelos tem o objetivo de abstrair código fonte para a análise ser feita independente de tecnologia, linguagem ou framework;
- Modelos do nível de função (UML): O resultado final da primeira etapa, geralmente as técnicas de engenharia reversa tem como resultado final diagramas UML, ele já fornece informações suficientes para construir modelos arquiteturais na segunda etapa;
- Modelos arquiteturais (Visão computação): Estes modelos serão construídos na segunda etapa do processo seguindo a norma ISO/IEC 19793 UML4ODP (2008), para expressar o sistema adequadamente com os conceitos de arquitetura de software.

2. Detalhar as atividades do processo genérico: O resultado é uma descrição explícita das tarefas de cada atividade e de cada artefato. Esse detalhamento é um tipo de instanciação de processo Borsoi (2008), ou seja, o processo é especializado para um modelo aplicável dentro dos requisitos de um contexto específico. Para detalhar as atividades utilizou-se o método de instanciação de processo Dias (2010) e foram utilizadas as técnicas de engenharia reversa de Ramanathan e Hodges (1996), Pereira, Martinez e Favre (2011) e Tonella (2005) como contexto sendo apresentado na Figura 7 e detalhado no item 3.1 e no item 3.2

**Figura 7 - Processo de construção de uma descrição arquitetural de sistemas através da engenharia reversa**




**Fonte: Elaborado pelo autor**

De acordo com a Figura 6, o processo deste trabalho é dividido em duas etapas principais: Engenharia Reversa para recuperação de Modelos, que mapeia o modelo *Horseshoe* partindo de analisar o código fonte até a representação do nível de função. Nesta etapa, um desenvolvedor que possuí um conhecimento detalhado do código fonte tem o papel de utilizar técnicas de engenharia reversa para construir os modelos UML de diagrama de classes, diagramas de caso de uso e diagramas de sequência.

A segunda etapa é a construção de uma descrição arquitetural da visão computação extraindo informações geradas nos modelos UML para a construção de modelos arquiteturais. Nesta etapa, um arquiteto tem o papel de usar as informações dos modelos construídos para construir a descrição arquitetural criando modelos da visão computação. Esta etapa tem como objetivo selecionar quais informações de cada modelo serão necessárias para construir cada modelo arquitetural.

## 3.1 Etapa 1: Engenharia reversa para recuperação de Modelos

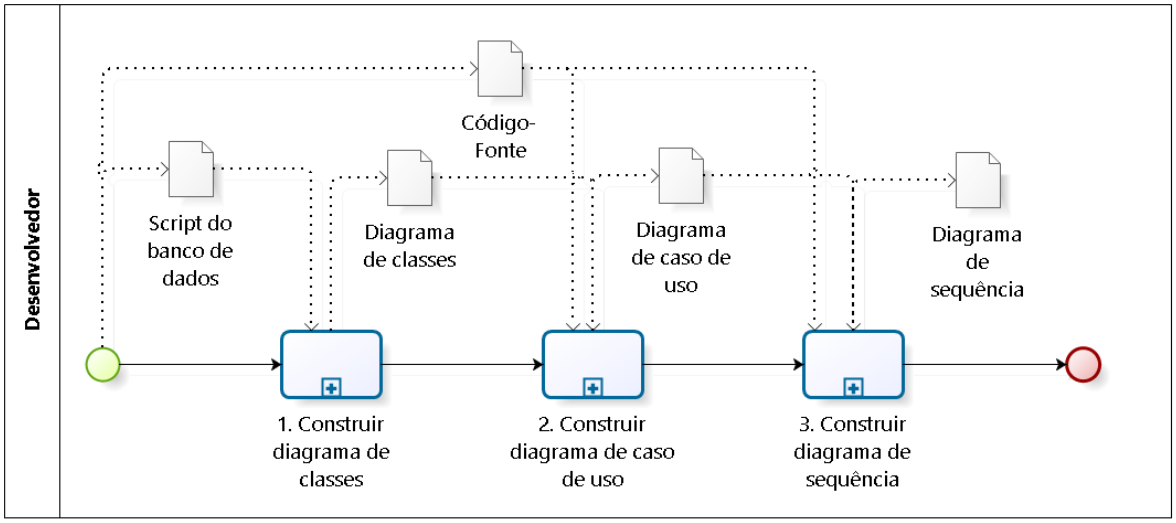
Esta seção apresenta o objetivo, a base para criação e quais os artefatos necessários para aplicação da primeira etapa do processo. O objetivo desta etapa é possibilitar a identificação das entidades de domínio e compreensão das funcionalidades do software e pela equipe de desenvolvimento, e possibilita o arquiteto analisar as informações contidas nos resultados desta etapa para expressar a arquitetura. A base para esta etapa é utilizar as seguintes técnicas de engenharia reversa:

1. Engenharia reversa de esquemas relacionais para esquemas orientados à objetos (Ramanathan e Hodges,1996);
2. Engenharia reversa para recuperação de diagramas de caso de uso (Pereira, Martinez e Favre, 2011);
3. Engenharia reversa para recuperação de diagramas de sequência (Tonella, 2005);

Para a aplicação de todas as técnicas, são necessários o código fonte e o script de criação do banco de dados do software. O objetivo das técnicas utilizadas nesta etapa é construir modelos no nível de função do modelo *horseshoe*. Apesar desta etapa utilizar estas técnicas, esta etapa não é limitada em utilizar apenas estás

técnicas, desde que sejam usadas técnicas que resultem em artefatos no nível de função do modelo horseshoe que expressem as funcionalidades do sistema e as entidades do domínio.

**Figura 8 - Etapas do processo de engenharia reversa**




**Fonte: Elaborado pelo autor**

Como a Figura 8 mostra, todas estas técnicas são atividades do processo, e têm como resultado final a construção de modelos UML, dentre estes modelos estão: Um diagrama de classe, um diagrama de caso de uso e um diagrama de sequência. Estes modelos foram escolhidos porque o uso dos diagramas de classe, caso de uso e sequencia são os diagramas mais utilizados para compreender o comportamento de um software. Esta etapa mostra quais informações são necessárias para construir uma descrição arquitetural e como elas devem ser organizadas.

### 3.1.1 Engenharia reversa de esquemas relacionais para esquemas orientado à objetos

3.1.1.1. Definição da técnica

A técnica apresentada por Ramanathan e Hodges (1996), utiliza a engenharia reversa em esquemas de banco de dados relacionais, para construir um diagrama de classes. Como premissa para utilizar a técnica, o banco de dados a ser analisado deve estar na terceira forma normal. A técnica consiste em três etapas principais:

- Identificação de classes de objetos: Esta etapa cria classes de objetos que correspondam às tabelas do modelo relacional. Uma tabela do modelo relacional que é categorizada como classe de objetos precisa estar em uma das seguintes condições:
    - Uma tabela que possua apenas um atributo como chave primária
    - Uma tabela que possua mais de um atributo como chave primária mas pelo menos um destes atributos não seja uma chave estrangeira
- Identificação de Relacionamentos: Para cada tabela do modelo relacional que possua chaves estrangeiras, existe um relacionamento entre duas tabelas. É necessário que estes relacionamentos sejam expressados no diagrama de classes. O objetivo desta etapa é identificar os relacionamentos divididos entre:
    - Associação: todas as tabelas do modelo relacional cuja chave primária consiste inteiramente por chaves estrangeiras, são tabelas associativas. Portanto, O diagrama de classes deve expressar um relacionamento entre todas as classes de objetos que correspondam as tabelas que estão associadas com uma tabela associativa.
    - Herança: As tabelas do modelo relacional que possuam a mesma chave primária são representadas no diagrama de classes através de um relacionamento de herança. A tabela no qual a chave primária pertence é representada no modelo conceitual como classe pai, e as tabelas que utilizam a chave primária são representadas no modelo conceitual como classes filho.
    - Agregação: Toda a tabela que a chave primária seja composta, possuindo chaves estrangeiras e que não sejam associativas, os relacionamentos entre estas tabelas são representados no diagrama de classes por uma agregação entre a classe que utiliza a chave estrangeira, e a classe que possuí a chave estrangeira como atributo.
- Expressar Cardinalidades: No modelo relacional, existem as cardinalidades entre relacionamentos de uma tabela, estas cardinalidades devem ser

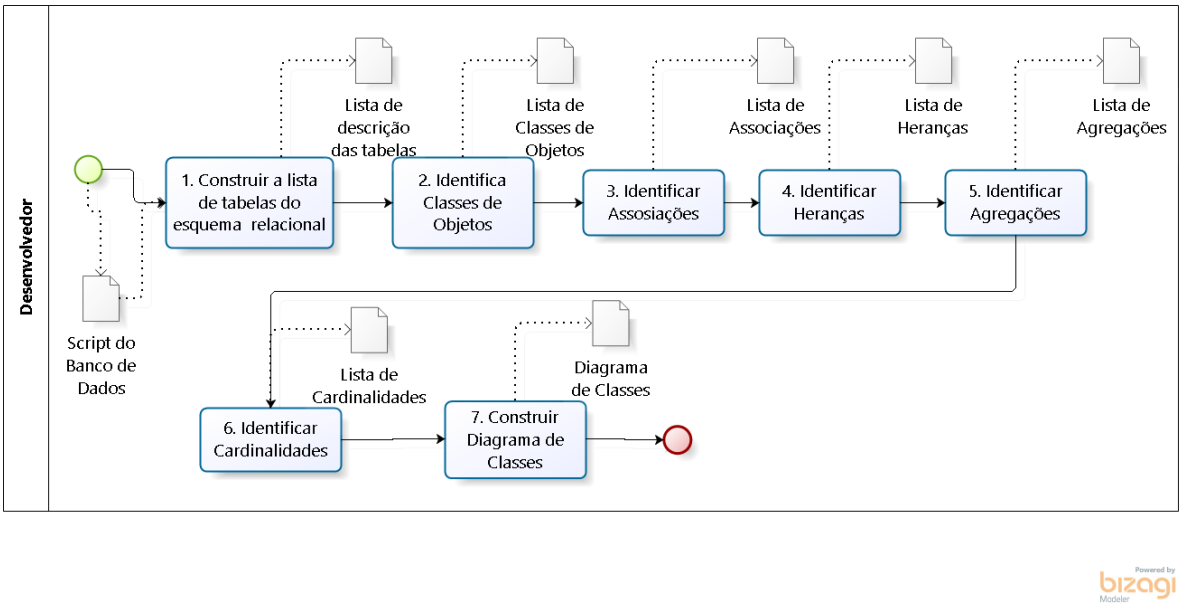
expressadas igualmente como multiplicidades entre as classes de objetos no diagrama de classes.

As informações desta técnica permitem construir o diagrama de tipos de dados na etapa de construir os modelos da visão computação no item 3.2.1 pois o diagrama de classes resultante desta técnica representa as entidades de domínio do software.

### 3.1.1.2. Aplicação da técnica

O objetivo desta técnica dentro do processo deste trabalho, é identificar as classes de entidade que o software possuí e os relacionamentos destas classes.

**Figura 9 - Processo de recuperação do esquema orientado a objetos**




**Fonte: Elaborado pelo autor**

Como a figura mostra, o desenvolvedor utiliza a técnica em uma série de atividades e tarefas que correspondem às etapas da técnica de engenharia reversa. Cada atividade é detalhada nas tabelas a seguir:

**Tabela 1 - Descrição da atividade "Construir a lista de tabelas do esquema relacional”**

| **Atividade** | A1. Construir a Lista de Tabelas do Esquema Relacional |
|---|---|
| Tarefas | T1. Identificar Tabelas<br>T2. Identificar atributos de cada tabela identificada<br>T3. Identificar relações de chaves primárias e chave estrangeira de cada tabela. |
| **Artefato de entrada** | Script do Banco de dados (Arquivo DDL) |
| **Artefato de saída** | Lista de descrição das tabelas |

**Fonte: Elaborado pelo autor**

**Tabela 2 - Descrição da atividade "Identificar Classes de Objetos”**

| **Atividade** | A2. Identificar Classes de Objetos |
|---|---|
| Tarefas | T1. Identificar as tabelas do esquema relacional que esteja nas condições de ser uma classe de objetos<br>T2. Para cada tabela identificada na tarefa anterior, modelar uma classe com o mesmo nome da tabela e os mesmos atributos. |
| **Artefato de entrada** | Lista de descrição das tabelas |
| **Artefato de saída** | Lista de tabelas de Classes de Objetos |

**Fonte: Elaborado pelo autor**

**Tabela 3 - Descrição da atividade "Identificar Associações"**

| **Atividade** | A3. Identificar Associações |
|---|---|
| Tarefas | T1. Criar uma Lista com o nome de “Lista de Associações”<br>T2. Identifique todas as tabelas associativas<br>T3. Para cada tabela identificada, insira um registro na lista de heranças na seguinte estrutura. |

| | |
|---|---|
| **Artefato de entrada** | • Lista de descrição das tabelas<br>• Tabela de Classes de Objetos |
| **Artefato de saída** | • Lista de Associações |

**Fonte: Elaborado pelo autor**

**Tabela 4 - Descrição da atividade "Identificar Heranças"**

| | |
|---|---|
| **Atividade** | A4. Identificar Heranças |
| Tarefas | T1. Criar uma Lista com o nome de “Lista de Heranças”<br>T2. Identificar todas as tabelas que estejam categorizadas por uma relação de herança (tabelas que possuem a mesma chave primária)<br>T3. Para cada tabela identificada, insira um registro na lista de heranças |
| **Artefato de entrada** | • Esquema relacional<br>• Tabela de Classes de Objetos |
| **Artefato de saída** | • Lista de Heranças |

**Fonte: Elaborado pelo autor**

**Tabela 5 - Descrição da atividade "Identificar Agregações"**

| | |
|---|---|
| **Atividade** | A5. Identificar Agregações |
| Tarefas | T1. Criar uma Lista com o nome de “Lista de Agregações”<br>T2. Identifique todas as tabelas que possuam relacionamento de agregação (tabelas que a chave primária tenha mais de um atributo e pelo menos um deles não é uma chave estrangeira)<br>T3. Para cada tabela identificada, insira um registro na lista de agregações |
| **Artefato de entrada** | • Esquema relacional<br>• Tabela de Classes de Objetos |
| **Artefato de saída** | • Lista de Agregações |

**Fonte: Elaborado pelo autor**

**Tabela 6 - Descrição da atividade “Identificar Cardinalidades”**

| **Atividade** | A6. Identificar Cardinalidades |
|---|---|
| Tarefas | T1. Criar uma Lista com o nome de “Lista de Cardinalidades”<br>T2. Para cada relacionamento entre 2 classes, identificar o tipo de cardinalidade<br>T3. Para cada cardinalidade identificadas, inserir um registro de multiplicidade correspondente a cardinalidade |
| **Artefato de entrada** | • Esquema relacional<br>• Tabela de Classes de Objetos<br>• Tabela de Associações<br>• Tabela de Heranças<br>• Tabela de Agregações |
| **Artefato de saída** | • Lista de Cardinalidades |

**Fonte: Elaborado pelo autor**

**Tabela 7 - Descrição da atividade "Construir o diagrama de classes"**

| **Atividade** | A7. Construir o diagrama de classes |
|---|---|
| Tarefas | T1. Crie um diagrama de classes com as seguintes considerações:<br>T2. Para cada classe de objetos, construa uma classe correspondente com o mesmo nome e os mesmo atributos.<br>T3. Para cada relacionamento na tabela de associações, herança e agregação, construa um no diagrama um relacionamento entre as classes<br>T4. Para cada cardinalidade entre as relações de tabelas, defina a multiplicidade entre as classes de forma idêntica. |

| | |
|---|---|
| **Artefato de entrada** | • Esquema relacional<br>• Tabela de Classes de Objetos<br>• Tabela de Associações<br>• Tabela de Heranças<br>• Tabela de Agregações<br>• Tabela de Cardinalidade |
| **Artefato de saída** | • Esquema Orientado a Objetos |

**Fonte: Elaborado pelo autor**

### 3.1.2 Engenharia reversa para recuperação de diagramas de caso de uso

3.1.2.1. Definição da técnica

A técnica apresentada por Pereira, Martinez e Favre (2011), constrói diagramas de caso de uso a partir da análise estática de um código orientado a objetos. Os diagramas de caso de uso resultantes desta técnica precisam ter os seguintes elementos:

- Casos de uso: Cada método público de uma classe é construído um caso de uso correspondente.
- Generalizações: Se existe um método em duas classes diferentes com o mesmo nome, é criada uma generalização.
- Dependências: Se um método público chama outro método público, é criado uma relação de dependência entre estes 2 métodos.

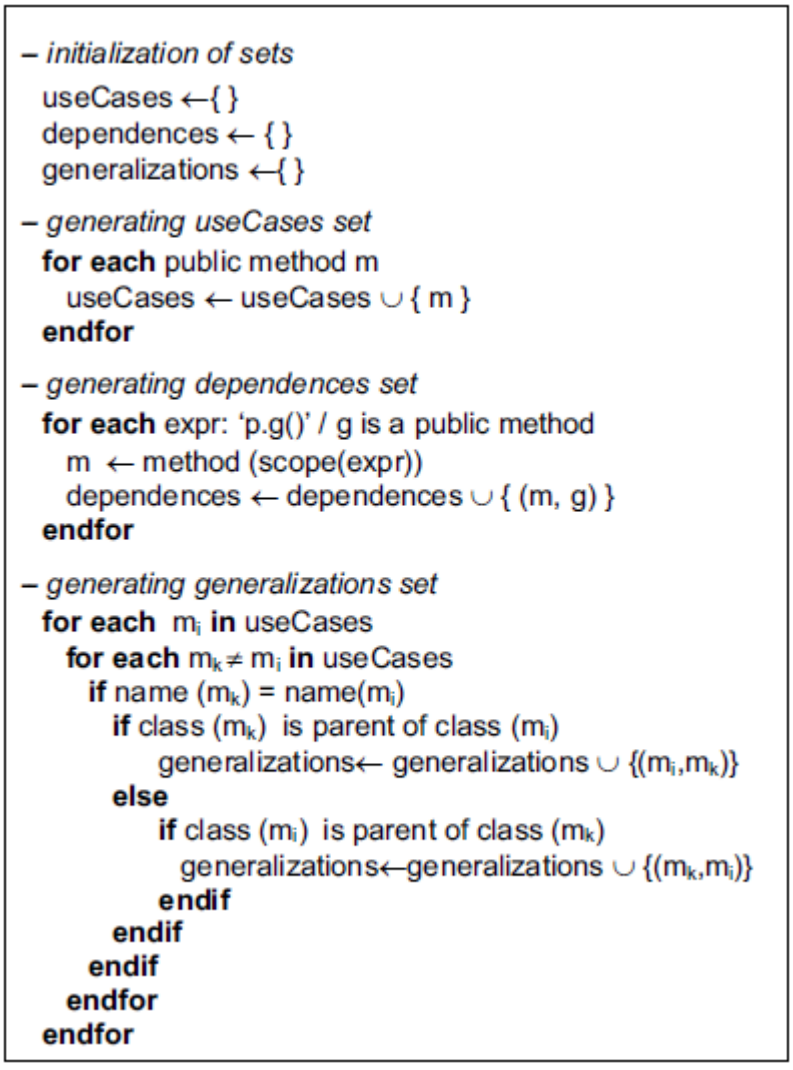
A técnica realiza as seguintes etapas:

- Criar o código abstrato da classe, realizando as seguintes tarefas:
  - Identificar declarações: No código fonte são linhas que declaram atributos, métodos e construtores.
  - A partir de cada declaração identificada, construa o código abstrato, sendo sua sintaxe: [nome da classe]. [nome do método].[declaração]

- Identificar instruções que são alocações, assinatura e invocações de métodos, para cada instrução, gere sua sintaxe que deve ser [nome da classe]. [nome do método].[instrução].
- Desconsidere condições, loops, retornos e importações.

- Aplicar algoritmo de recuperação de diagramas de caso de uso como mostra a Figura 10.

**Figura 10 - Algoritmo de Recuperação de Diagramas de Caso de Uso**



```
– initialization of sets
  useCases ←{ }
  dependences ← { }
  generalizations ←{ }
– generating useCases set
  for each public method m
    useCases ← useCases ∪ { m }
  endfor
– generating dependences set
  for each expr: 'p.g()' / g is a public method
    m ← method (scope(expr))
    dependences ← dependences ∪ { (m, g) }
  endfor
– generating generalizations set
  for each mi in useCases
    for each mk ≠ mi in useCases
      if name (mk) = name(mi)
        if class (mk) is parent of class (mi)
            generalizations← generalizations ∪ {(mi,mk)}
        else
            if class (mi) is parent of class (mk)
              generalizations←generalizations ∪ {(mk,mi)}
            endif
        endif
      endif
    endfor
  endfor
```

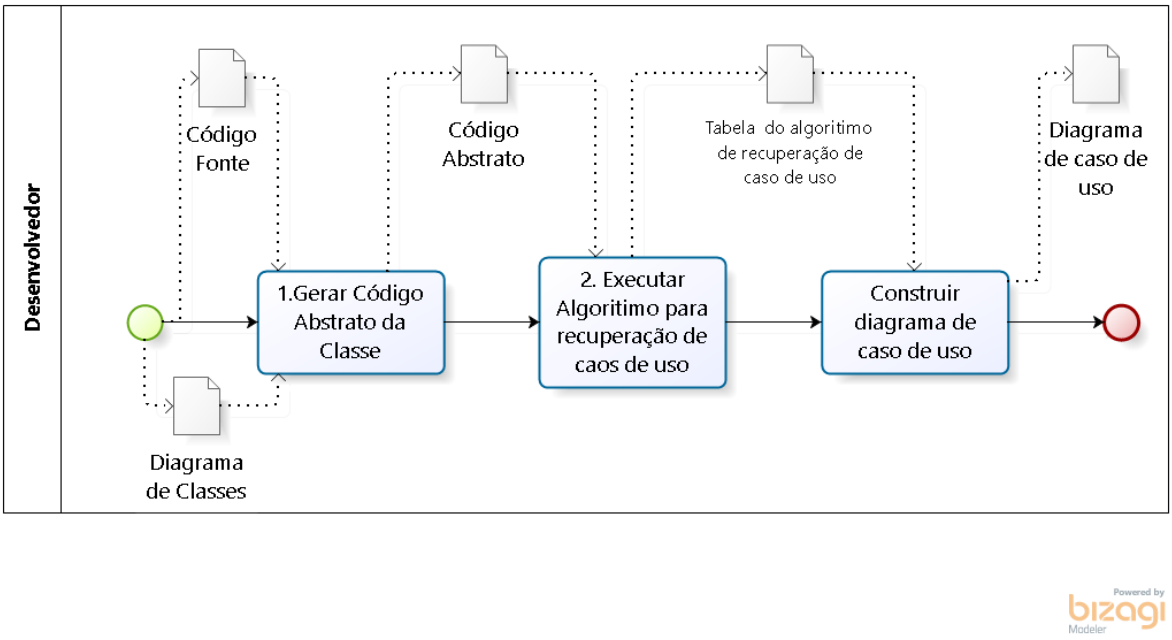
**Fonte: Pereira, Martinez e Favre (2011)**

Esta técnica se limita a identificação dos casos de uso pela análise estática, pois a analise não é realizada enquanto o sistema está em funcionamento, não sendo possível identificar os atores do sistema. De acordo com os autores, para identificar os fluxos do caso de uso seria necessária uma análise dinâmica no código fonte, extraindo os fluxos do sistema em funcionamento ou através da utilização de casos de teste. Os autores citam algumas etapas para recuperar casos de uso, que tem como base o modelo utilizado por Tonella (2005), que será explicado no item 3.1.3. As informações obtidas neste modelo permitem construir os diagramas de assinaturas de interface, template de interface e template de objetos no item 3.2.1, pois o resultado desta técnica expressam os serviços relacionadas com cada entidade identificada no item 3.1.1.

#### 3.1.2.2. Aplicação da Técnica

O objetivo da técnica no processo desse trabalho é analisar os serviços que são responsáveis pelas classes identificadas no diagrama de classes da técnica anterior. Para utilizar essa técnica no processo descrito no presente trabalho, esta foi dividida no seguinte processo:

**Figura 11 - Processo de recuperação de diagramas de caso de uso**




**Fonte: Elaborado pelo autor**

Como a Figura 11 mostra, o desenvolvedor utiliza a técnica em uma série de atividades que correspondem às etapas da técnica de engenharia reversa. Cada atividade é detalhada nas tabelas a seguir:

**Tabela 8 - Descrição da atividade "Gerar código abstrato da classe"**

| **Atividade** | A1. Gerar código abstrato da Classe |
|---|---|
| Tarefas | T1. Identificar classes que manipulam a classe de objeto do modelo orientado a objetos.<br>T2. Para cada classe identificada, gerar um código abstrato de cada classe. |
| **Artefato de entrada** | • Esquema orientado à objetos<br>• Código Fonte |
| **Artefato de saída** | • Código Abstrato |

**Elaborado pelo autor**

**Tabela 9 - Descrição da atividade "Aplicar algoritmo de recuperação de caso de uso"**

| **Atividade** | A2. Aplicar algoritmo de recuperação de caso de uso |
|---|---|
| Tarefas | T1. Criar uma tabela com a seguinte estrutura de colunas (Caso de Uso, Dependência, Generalização)<br>T2. A partir do código abstrato, aplicar o algoritmo de análise estática:<br>T3. Identificar casos de uso.<br>T4. Identificar generalizações.<br>T5. Identificar dependências. |
| **Artefato de entrada** | • Esquema orientado à objetos<br>• Código Fonte |
| **Artefato de saída** | • Tabela de saída do algoritmo de análise estática |

**Fonte: Elaborado pelo autor**

**Tabela 10 - Descrição da atividade "Construir diagrama de caso de uso"**

| **Atividade** | A3. Construir diagrama de caso de uso |
|---|---|
| **Tarefas** | T1. Modelar casos de uso<br>T2. Modelar generalizações<br>T3. Modelar dependências |
| **Artefato de entrada** | • Tabela de saída do algoritmo de análise estática<br>• Código Fonte |
| **Artefato de saída** | • Tabela de saída do algoritmo de análise estática |

**Fonte: Elaborado pelo autor**

### 3.1.3 Engenharia reversa para recuperação de diagramas de sequência

3.1.3.1. Definição da técnica

A técnica apresentada por Tonella (2005) consiste em utilizar a engenharia reversa para modelar diagramas de sequência, com o objetivo de entender o comportamento interno das funcionalidades do software e as mudanças que um objeto tem através do fluxo de mensagens entre métodos. A técnica do autor consiste nas seguintes etapas:

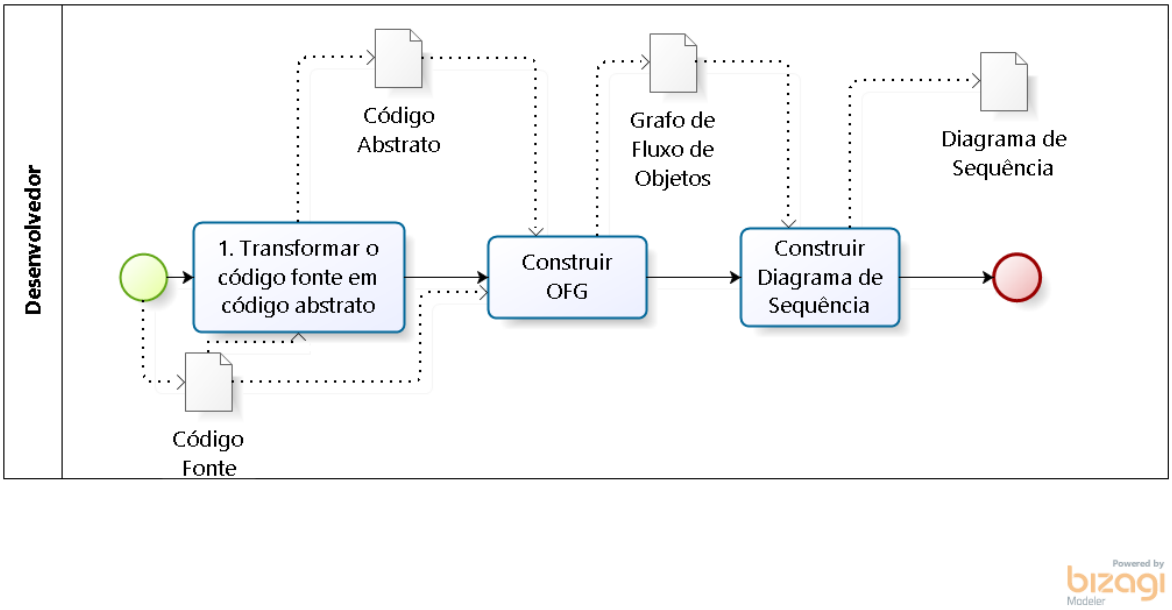
- Extrair o código abstrato do método, utilizando a mesma técnica do item 3.1.2;
- Construir um OFG (grafo de fluxo de objetos) através do código abstrato, para identificar as mudanças que o objeto sobre ao decorrer do método;
- Identificar as chamadas de métodos, o objeto que realiza a chamada e os objetos que são impactados pela chamada;
- Identificar condições e loops codificados.

As informações obtidas neste modelo permitem construir comportamento dos serviços do diagrama de assinaturas de interfaces no item 3.2.1, pois o resultado desta técnica expressa como funciona os serviços por cada caso de uso identificado no item 3.1.2.

#### 3.1.3.2. Aplicação da Técnica

O objetivo desta técnica neste trabalho é modelar o comportamento interno dos casos de uso identificados na técnica anterior. Para utilizar está técnica no processo descrito por este trabalho, ela foi dividida no seguinte processo:

**Figura 12 - Processo de recuperação de diagrama de sequência**




**Fonte: Elaborado pelo autor**

Como a Figura 12 mostra, o desenvolvedor utiliza a técnica em uma série de atividades que correspondem às etapas da técnica de engenharia reversa. Está técnica, assim como a técnica para recuperação do caso de uso, não identifica um ator externo do sistema, e por não ser uma análise dinâmica, não identifica quais fluxos ocorrem durante o funcionamento do sistema. Cada atividade é detalhada nas tabelas a seguir:

**Tabela 11 - Descrição da atividade "Transformar método em código abstrato"**

| **Atividade** | A1. Transformar método em código abstrato |
|---|---|
| Tarefas | T1. Identificar declarações<br>T2. Identificar Instruções |
| **Artefato de entrada** | • Diagrama de caso de uso<br>• Código Fonte |
| **Artefato de saída** | • Código Abstrato |

**Fonte: Elaborado pelo autor**

**Tabela 12 - Descrição da atividade "Construir código OFG do código abstrato"**

| **Atividade** | A2. Construir OFG do código abstrato |
|---|---|
| Tarefas | T1. Identifique as declarações de objetos e instruções do código abstrato<br>T2. Crie um nó para cada objeto declarado e para cada instrução que modifique o fluxo deste objeto<br>T3. Para cada relacionamento entre as declarações e as instruções que modificam este objeto, crie uma ponta entre os nós. |
| **Artefato de entrada** | • Código abstrato |
| **Artefato de saída** | • Grafo de fluxo de objetos |

**Fonte: Elaborado pelo autor**

**Tabela 13 - Descrição da atividade "Construir diagrama de sequência"**

| **Atividade** | A3. Construir diagrama de sequência |
|---|---|
| Tarefas | T1. Crie uma tabela com as seguintes colunas (Linha, Chamada, Fontes, Alvos)<br>T2. Analisar chamadas de métodos no OFG e identifique os métodos.<br>T3. Analisar o método, mapeando as ocorrências de loops e condições.<br>T4. Inserir marcas do diagrama de sequência. |
| **Artefato de entrada** | • Código fonte<br>• Grafo de fluxo de objetos |
| **Artefato de saída** | • Diagrama de sequência |

**Fonte: Elaborado pelo autor**

## 3.2 Etapa 2: Construção de uma descrição arquitetural

Esta seção apresenta a segunda etapa do processo descrito neste trabalho, a justificativa da escolha da visão computação, a relação dos artefatos gerados na primeira etapa com os artefatos que serão gerados nesta etapa e os conceitos utilizados na primeira etapa com relação aos conceitos desta etapa.

O objetivo desta etapa é utilizar as informações de modelos do nível de função para descrever a arquitetura do software. Os modelos resultantes da primeira etapa conceitualmente expressam as funcionalidades e comportamento de um software, através de diagramas estruturais como o diagrama de classe e diagramas de comportamento como os diagramas de caso de uso e de sequência, além disso, estes diagramas são utilizados pela UML4ODP para representar a visão computação de uma arquitetura. Portanto, a visão computação é a visão escolhida para realizar a descrição arquitetural, pois está visão tem o papel de expressar os serviços e o comportamento destes serviços dentro de uma descrição arquitetural.

De acordo com Romero (2011), os conceitos da UML mapeiam em grande parte os conceitos do ODP, portanto, é viável construir um modelo da visão computação, baseando-se em um modelo UML. O autor apresenta uma tabela correspondendo os conceitos da visão computação do ODP com elementos UML pela tabela a seguir:

**Tabela 14 - Tabela de perfil de conceitos ODP e UML**

| 1. Conceito ODP | 2. Elemento UML | 3. Estereótipo |
|---|---|---|
| **Computacional Object Template** | Component | <<CV_CompObjectTemplate>> |
| **Computacional interface template** | Port | <<CV_CompInterfaceTemplate>> |
| **Signal Interface Signature** | Interface(s) | <<CV_SignalInterfaceSignature>> |
| **Operation Interface Signature** | Interface(s) | <<CV_OperationInterfaceSignature>> |

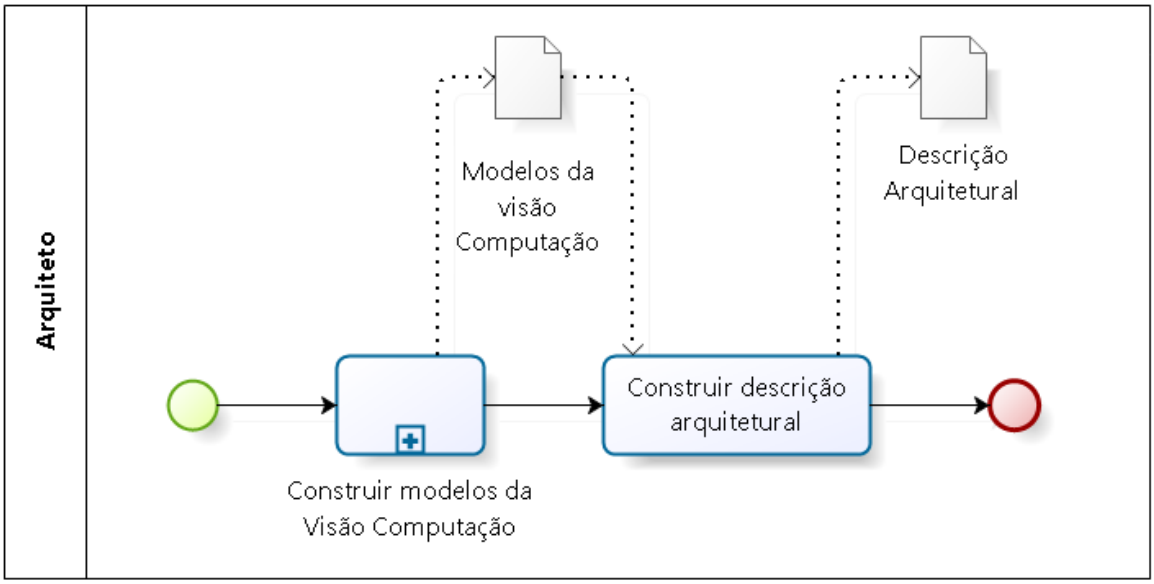
**Tabela 14 - Tabela de perfil de conceitos ODP e UML (Conclusão)**

| | | |
|---|---|---|
| **Stream Interface Signature** | Interface(s) | <<CV_StreamInterfaceSignature>> |
| **Announcement Signature** | Reception | <<CV_AnnouncementSignature>> |
| **Interrogation Signature** | Reception | <<CV_InterrogationSignature>> |
| **Termination Signature** | Reception | <<CV_TerminationSignature>> |
| **Signal signature** | Reception | <<CV_SignalSignature>> |
| **Flow Signature** | Reception | <<CV_FlowSignature>> |
| **Computacional Object** | InstanceSpecification | <<CV_Object>> |
| **Signal Interface** | Port(interaction Point) | <<CV_SignalInterface>> |
| **Operation Interface** | Port (Interaction Point) | <<CV_OperationInterface>> |
| **Stream Interface** | Port (Interaction Point) | <<CV_StreamInterface>> |
| **Signal** | Message | <<CV_Signal>> |
| **Flow** | Interaction/Message | <<CV_Flow>> |
| **Announcement** | Message | <<CV_Announcement>> |
| **Invocation** | Message | <<CV_Invocation>> |
| **Termination** | Message | <<CV_Termination>> |

**Fonte: Romero e Vallecillo, 2005**

Como a tabela mostra, para cada conceito da visão computação do ODP, sua representação pode ser feita por um elemento UML, desde que o elemento no modelo seja categorizado por um estereótipo do conceito, este estereótipo categoriza o conceito apresentado em cada elemento do modelo arquitetural. O processo de construir modelos arquiteturais baseado na tabela 13 é o objetivo da primeira atividade da segunda etapa. As atividades de construção da descrição arquitetural são apresentadas na Figura 13:

**Figura 13 - Processo de contrução da descrição arquitetural**




**Fonte: Elaborado pelo autor**

**Tabela 15 - Descrição da atividade "Construir descrição arquitetural"**

| **Atividade** | A1. Construir descrição arquitetural |
|---|---|
| Tarefas | T1. Definir o propósito do sistema<br>T2. Definir as características do sistema<br>T3. Definir os stakeholders<br>T4. Inserir a visão computação |

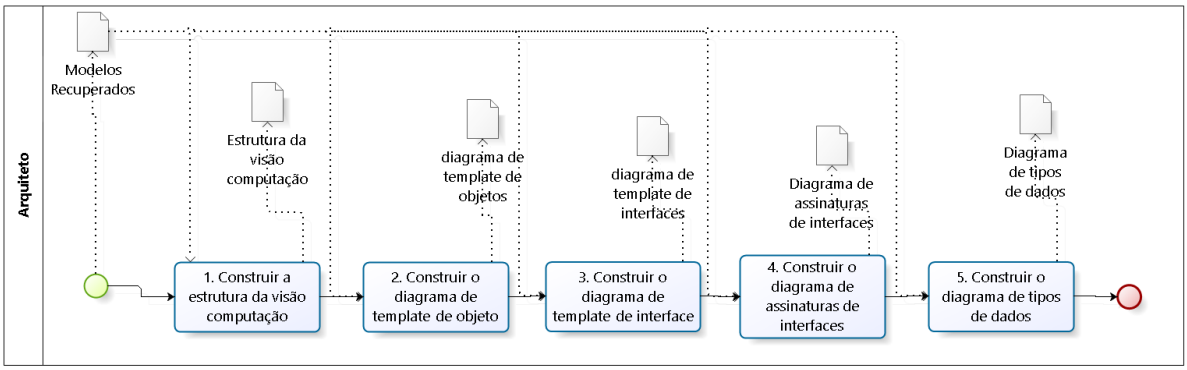
| | |
|---|---|
| | T5. Inserir os modelos da visão computação<br>T6. Inserir o ponto de vista computacional<br>T7. Inserir os modelos do ponto de vista computacional |
| **Artefato de entrada** | Modelos da visão computação |
| **Artefato de saída** | Descrição Arquitetural |

**Fonte: Elaborado pelo autor**

### 3.2.1 Construir Modelos da Visão Computação

Esta seção apresenta a base conceitual para especificação da visão computação, sua contribuição para a compreensão da arquitetura do sistema, limitações e as atividades e tarefas do processo para especificar este ponto de vista.

**Figura 14 - Processo de construção da visão computação**




**Fonte: elaboração do autor**

**Tabela 16 - Descrição da atividade "Construir uma estrutura da visão computação"**

| | |
|---|---|
| **Atividade** | A2. Construir estrutura da visão computação |
| Tarefas | T1. Construir um diagrama de pacotes com os seguintes pacotes: |

| | |
|---|---|
| | a. Template de objetos<br>b. Template de interfaces<br>c. Assinaturas de Interfaces<br>d. Tipos de dados |
| **Artefato de entrada** | |
| **Artefato de saída** | Diagrama da estrutura da visão computação |

**Fonte: Elaborado pelo autor**

**Tabela 17 - Descrição da atividade "Construir o diagrama de template de objetos"**

| **Atividade** | A3. Construir o diagrama de template de objetos |
|---|---|
| Tarefas | T1. Identificar objetos computacionais<br>T2. Para cada objeto identificado na tarefa 1, criar uma classe correspondente com o estereótipo “Objeto da visão computação” |
| **Artefato de entrada** | Diagrama de caso de uso |
| **Artefato de saída** | Diagrama de assinaturas |

**Fonte: Elaborado pelo autor**

**Tabela 18 - Descrição da atividade "Construir o diagrama de templates de interfaces"**

| **Atividade** | A4. Construir o diagrama de template de interfaces |
|---|---|
| Tarefas | T1. Identificar interfaces da visão computação<br>T2. Para cada interface identificado no passo 1, criar uma classe correspondente com o estereótipo “Interface da visão computação” |
| **Artefato de entrada** | Diagrama de caso de uso |
| **Artefato de saída** | Diagrama de assinaturas |

**Fonte: Elaborado pelo autor**

**Tabela 19 - Descrição da atividade "Construir o diagrama de assinaturas"**

| **Atividade** | A5. Construir o diagrama de assinaturas |
|---|---|
| Tarefas | T1. Identificar assinaturas de interface de operações<br>T2. Para cada assinatura identificada na tarefa 1, construir uma classe do estereótipo “Assinatura de interface da visão computação”<br>T3. Identificar sinais de interface<br>T4. Para cada sinal identificado, construir um relacionamento entre objetos de assinaturas de interface correspondentes |
| **Artefato de entrada** | Diagrama de caso de uso |
| **Artefato de saída** | Diagrama de assinaturas |

**Fonte: Elaborado pelo autor**

**Tabela 20 - Descrição da atividade "Construir diagrama de tipo de dados"**

| **Atividade** | A6. Construir o diagrama de tipos de dados |
|---|---|
| Tarefas | T1. Identificar classes no esquema orientado a objetos<br>T2. Identificar parâmetros dos métodos nas interações dos diagramas de sequencia<br>T3. Para cada item identificado no passo 1, modele uma classe com o estereótipo “Tipo de dado da visão computação”<br>T4. Para cada item modelado no passo 2, identificar seus atributos<br>T5. Para cada item identificado no passo 3, modelar os atributos correspondentes a cada classe |
| **Artefato de entrada** | Esquema orientado à objetos<br>Diagrama de sequencia |
| **Artefato de saída** | Diagrama de tipos de dados |

**Fonte: Elaborado pelo autor**

**Tabela 21 - Descrição da atividade "Construir o diagrama de comportamento da assinatura"**

| | |
|---|---|
| **Atividade** | A7. Construir o diagrama de comportamento da assinatura |
| Tarefas | T1. Construir um diagrama de sequência do estereótipo “Comportamento da visão computação”<br>T2. Identificar ciclos de vida<br>T3. Para cada ciclo de vida, modelar um ciclo de vida do estereótipo “Objeto da visão computação”<br>T4. Identificar fluxo<br>T5. Identificar interações<br>T6. Para cada mensagem do fluxo, modelar um sinal correspondente no diagrama construído no passo 1<br>T7. Para cada interação identificada no passo 5, modelar um sinal com o nome correspondente e os parâmetros iguais. |
| **Artefato de entrada** | Diagrama de sequência |
| **Artefato de saída** | Diagrama de comportamento |

**Fonte: Elaborado pelo autor**

## 3.3 Conclusão do capitulo

Este processo limita-se em analisar o software e descrever sua arquitetura, não em um processo de reengenharia de um sistema em si, pois, o objetivo deste processo é auxiliar as equipes de desenvolvimento a compreender a arquitetura do sistema e discutir as necessidades dos *stakeholders* adequadamente e o impacto das adaptações que são necessárias para atender estas necessidades, este processo não define linguagens de programação, tecnologias e ferramentas específicas para construir os modelos, estas definições são exemplificadas no capítulo 4.

# 4 ROTEIROS DO PROCESSO

Este capítulo apresenta o roteiro operacional que foi aplicado do processo especificado no capitulo 3. O Cenário de aplicação deste trabalho envolve uma organização do setor comercial que deseja obter um sistema de comércio eletrônico, entretanto, esta organização não possuí uma área de TI especializada para desenvolvimento de aplicações corporativas, portanto, a empresa necessitou terceirizar a implementação do ecommerce por uma empresa de soluções e consultoria em TI.

A empresa contratada fechou um acordo para criar o sistema, e como solução, apresentou um software de comércio eletrônico de código aberto, afirmando que o software possuía diversas funcionalidades que a organização desejava utilizar em seu modelo de negócio, além disso, afirmaram que após analisar as informações do software no site oficial e testar o sistema em um ambiente controlado, identificaram que o software tinha configurações que permitiam adequar o software com as necessidades da organização. O sistema apresenta as seguintes características técnicas:

- O sistema é de código aberto do paradigma Orientado à Objetos.
- O sistema é uma aplicação *web*, utilizando o framework ASP.NET MVC que utiliza a linguagem C#.
- O sistema utiliza um banco de dados relacional, criado na tecnologia SQL 2008 R2.
- O sistema é hospedado no servidor IIS.
- Como artefatos do sistema, o desenvolvedor possuí o código fonte e o script de criação do esquema SQL do banco de dados

Depois de testar em conjunto a aplicação com a empresa de consultoria, concluiu-se que apesar do sistema mapear seus interesses, o sistema precisava estar adequado às regras de negócio de frete, adicionar novos métodos de pagamento, modificar os campos para registro do cliente, modificar apresentação das tabelas no módulo administrativo. Sendo assim, a empresa de consultoria decidiu realizar a adaptação.

Durante a fase de especificação do ecommerce adaptado, a equipe de desenvolvimento considerou procurar alguma documentação do projeto de código aberto, para que fosse possível estimar o impacto, tempo e recursos que seriam necessários para realizar as adaptações. Entretanto, a documentação referente a arquitetura, não era detalhada, reforçando os resultados da pesquisa realizada por Ding et al (2014). A falta da documentação arquitetural do ecommerce resultou nos seguintes obstáculos durante o projeto de adaptação.

- A equipe de desenvolvimento precisou de um tempo acima do esperado para entender o código fonte do sistema;
- A comunicação entre o cliente da organização, o líder do projeto e à equipe de desenvolvimento entravam em conflito por modificações que o cliente considerava simples de implementar e a equipe de desenvolvimento afirmava que havia uma complexidade alta em realizar tal mudança;
- Os Prazos eram definidos com base apenas na inferência e especialidade do código fonte da equipe de desenvolvimento e geralmente acabavam sendo ultrapassados;

A empresa de consultoria deseja que as futuras alterações não tenham estes obstáculos, além disso, a empresa tem como objetivo fornecer o ecommerce para novos clientes que possam desejar à adoção e adaptação do sistema de comércio eletrônico, sendo assim, a equipe de desenvolvimento da empresa entende que é necessário criar as adaptações de forma que elas sejam reutilizadas para outros clientes. Estas adaptações dos serviços e funcionalidades do ecommerce precisam estar de acordo com uma arquitetura que permita o reuso, e para isso, os stakeholders do projeto precisam entender o ecommerce através de modelos que descrevam a arquitetura do sistema atual.

A partir destes obstáculos e como não foi possível realizar alguma análise eficiente sobre o sistema a partir de uma documentação, assim como a ISO/IEC 14764 (2006) sugere, será necessário realizar a engenharia reversa ao nível arquitetural em partes do código para uma análise sobre a adaptação do sistema.

### 4.1.1 Aplicação do Processo e Especificação do Roteiro

Esta seção apresenta a aplicação do processo apresentado no capitulo 3 e a criação do roteiro operacional a partir das atividades modeladas.

A aplicação deste roteiro tem como necessidade o seguinte requisito especificado pelo cliente.

- Modificar o cadastro do cliente para identificar se a pessoa é física ou jurídica
- Remover campos de endereço do registro do cliente para utilizar apenas o registro de endereço
- Adicionar campos de CNPJ e Inscrição Estadual
- Separar o campo de endereço no registro de endereço entre logradouro, número, complemento e bairro

### 4.1.2 Roteiro da recuperação do esquema orientado a objetos

**Tabela 22 - Descrição do roteiro da atividade "Construir a lista de tabelas do esquema relacional"**

<table>
<tr><td>Atividade</td><td>A1. Construir a Lista de Tabelas do Esquema relacional</td></tr>
<tr><td>Roteiro</td><td>1. Criar uma lista com a seguinte estrutura:
<table>
<tr><th colspan="4">Lista de tabelas de esquema relacional</th></tr>
<tr><td>C1. Nome da Tabela</td><td>C2. Atributos</td><td>C3. Chaves Primárias</td><td>C4. Chaves Estrangeiras</td></tr>
</table>
2. Abra o bloco de notas e abra o script do banco de dados para identificar instruções de criação de tabela ex:“Create Table [Nome da Tabela] {Lista de Atributos} ”.<br>3. Para cada instrução identificada no passo 2, inserir um registro na lista criada no passo 1, inserindo na coluna “C1” o nome da tabela que está no script, ex:
<table>
<tr><td>C1. Nome da Tabela</td><td>C2. Atributos</td><td>C3. Chaves Primárias</td><td>C4. Chaves Estrangeiras</td></tr>
<tr><td>Nome da Tabela</td><td></td><td></td><td></td></tr>
</table>
</td></tr>
</table>

4. Para cada registro inserido no passo 3, inserir os atributos correspondentes aos registros das tabelas identificadas no passo 2, ex:

| C1. Nome da Tabela | C2. Atributos |
|---|---|
| Nome da Tabela | Atributo 1 (tipo)<br>Atributo 2 (tipo) |

5. Para cada instrução identificada no passo 2, identificar atributos categorizados como chaves primária ex:“Atributo 1 (tipo) `PRIMARY KEY`”
6. Para cada item identificado no passo 5, inserir na lista criada no passo 1 no registro da tabela que a chave pertence, as chaves primárias identificadas no passo 5 na coluna “C3”, ex:

| C1. Nome da Tabela | C2. Atributos | C3. Chaves Primárias |
|---|---|---|
| Nome da Tabela | Atributo 1 (tipo)<br>Atributo 2 (tipo) | Atributo 1 (tipo) |

7. Identificar chaves estrangeiras, analisando instruções que tenham a seguinte estrutura: “CONSTRAINT [Nome da chave estrangeira] FOREIGN KEY (Nome do atributo da chave estrangeira) REFERENCES Tabela principal (Atributo Principal)”.
8. Para cada item identificado no passo 7, inserir na lista criada no passo 1 um registro correspondente a tabela pertencente a chave na seguinte estrutura:

| C1. Nome da Tabela | C2. Atributos | C4. Chaves Estrangeiras |
|---|---|---|
| Nome da Tabela | Atributo 1 (tipo)<br>Atributo 2 (tipo) | [Atributo 2 (Nome da Tabela Principal)] |

**Fonte: Elaborado pelo autor**

**Tabela 23 - Descrição do roteiro da atividade "Identificar classes de objetos"**

| Atividade | Identificar classes de objetos |
|---|---|

<table>
<tr><td>Roteiro</td><td>1. Criar uma lista a seguinte estrutura:<br><table><tr><th colspan="4">Tabela de Classes de Objetos</th></tr><tr><td>C1. Id</td><td>C2. Nome da Classe</td><td>C3. Atributos da Classe</td><td>C4. Tabela Correspondente</td></tr></table>2. Identificar as tabelas do banco de dados que sejam categorizadas como classes de objetos<br>3. Identificar atributos das tabelas<br>4. Para cada Tabela identificada no passo “2”, inserir um registro na lista criada no passo 1, com as seguintes considerações:<br>a. O campo Id é sequencial<br>b. O registro da coluna “C2” é idêntico ao nome da tabela<br>c. Os atributos devem ser expressados iguais aos atributos da tabela, na estrutura [Nome do atributo (Tipo do atributo)]<br>d. O campo “C4” é inserido o nome da tabela correspondente a classe registrada.</td></tr>
</table>

**Fonte: Elaborado pelo autor**

**Tabela 24 - Descrição do roteiro da atividade "Identificar associações"**

<table>
<tr><td>Atividade</td><td>Identificar associações</td></tr>
<tr><td>Roteiro</td><td>1. Criar uma Lista com a seguinte estrutura:<br><table><tr><th colspan="4">Lista de Associações</th></tr><tr><td>C1. Id</td><td>C2. Classe 1</td><td>C3. Classe 2</td><td>C4. Nome da Associação</td></tr></table>2. Identifique todas as tabelas associativas.<br>3. Para cada tabela identificada, insira um registro na lista de associações na seguinte estrutura:<br><table><tr><td>C1. Id</td><td>C2. Classe de objeto 1</td><td>C3. Classe de objeto 2</td><td>C4. Nome da Associação</td></tr><tr><td>Sequencial</td><td>Nome da Classe (correspondente ao relacionamento</td><td>Nome da Classe (correspondente ao relacionamento</td><td>R[Id] – [Id da classe1] [Id da classe 2]</td></tr></table></td></tr>
</table>

| | | | | |
|---|---|---|---|---|
| | | identificado na tabela associativa) | identificado na tabela associativa) | |

**Fonte: Elaborado pelo autor**

**Tabela 25 - Descrição do roteiro da atividade "Identificar Heranças"**

| Atividade | Identificar heranças |
|---|---|
| Roteiro | 1. Criar uma Lista com a seguinte estrutura:<br>**Lista de Heranças**<br>C1. Id \| C2. Classe Pai \| C3. Classe Filho \| C4. Nome da Herança<br>2. Analisar o esquema relacional e identificar todas as tabelas que possuam uma relação de herança (tabelas que possuem a mesma chave primária).<br>3. Para cada tabela identificada no passo 2, inserir um registro na lista criada no passo 1, na seguinte estrutura:<br>a. O Id segue a seguinte estrutura: “C [id da classe associada 1] R [Número sequencial do relacionamento]”<br>b. A classe pai corresponde a classe que possuí o atributo da chave primária<br>c. A classe filha corresponde a classe que utiliza a chave primária da classe pai |

**Fonte: Elaborado pelo autor**

**Tabela 26 - Descrição do roteiro da atividade "Identificar Agregações"**

| Atividade | Identificar Agregações |
|---|---|
| Roteiro | 1. Criar uma Lista com o a seguinte estrutura:<br>**Lista de Agregações**<br>C1. Id \| C2. Classe Agregadora \| C3. Classe Agregada \| C4. Nome da agregação |

| | |
|---|---|
| | 2. Identifique todas as tabelas que possuam relacionamento de agregação (tabelas que a chave primária tenha mais de um atributo e pelo menos um deles não é uma chave estrangeira (tabelas que possua uma chave primaria composta, mas que pelo menos um dos atributos não seja uma chave estrangeira).<br>3. Para cada Tabela Identificada, registre uma agregação com as seguintes condições:<br>a. Classe agregada = Classe que utiliza o atributo de outra classe.<br>b. Classe agregadora = classe que fornece o atributo usado pelas classes agregadas |

**Fonte: Elaborado pelo autor**

**Tabela 27 - Descrição do roteiro da atividade "Identificar cardinalidades"**

<table>
<tr><td>Atividade</td><td>Identificar Cardinalidades</td></tr>
<tr><td>Roteiro</td><td>1. Crie uma Lista com a seguinte estrutura:
<table>
<tr><th colspan="4">Lista de Cardinalidades/Multiplicidades</th></tr>
<tr><td>C1. Id</td><td>C2. Relacionamento</td><td>C3. Cardinalidade</td><td>C4. Multiplicidade</td></tr>
</table>
2. Para cada registro de relacionamento entre 2 classes nas listas de associações, heranças e agregações, identificar suas cardinalidades.<br>3. Para cada cardinalidade identificada, insira um registro na lista criada no passo 1, com as seguintes considerações:<br>a. Id: Sequencial (1,2,3,4)<br>b. Relacionamento = (Nome do relacionamento)<br>c. Cardinalidade = (ex: 1 para 1,1 para muitos, muitos para muitos)<br>d. Multiplicidade = (ex: 1 para 1,1 para muitos, muitos para muitos)</td></tr>
</table>

**Fonte: Elaborado pelo autor**

**Tabela 28 - Descrição do roteiro da atividade "Construir esquema orientado à objetos"**

| Atividade | Construir esquema orientado à objetos |
|---|---|
| Roteiro | 1. Crie um diagrama de classes com as seguintes considerações:<br>a. Para cada classe de objetos, construa uma classe respectiva com o mesmo nome e os mesmos atributos sendo todos públicos,<br>b. Para cada relacionamento na tabela de associações, herança e agregação, construa no diagrama um relacionamento entre as classes<br>c. Para cada cardinalidade entre as relações, expressar a multiplicidade entre o relacionamento. |

**Fonte: Elaborado pelo autor**

### 4.1.3 Roteiro para recuperação de diagrama de caso de uso

**Tabela 29 - Descrição do roteiro da atividade "Gerar código abstrato"**

| Atividade | Gerar Código Abstrato |
|---|---|
| Roteiro | 1. Abra o Visual Studio, e na janela de “explorador da solução” Identificar classes que manipulam as classes de objetos identificadas no item 4.1.2 (Classes de controller e services) ex: public class [Nome da classe][Service]).<br>1. Para cada classe identificada, gerar seu código abstrato seguindo as seguintes considerações:<br>a. Uma classe possui 0 ou muitas declarações seguida de 0 ou muitas instruções.<br>b. Declarações são declarações de atributos, métodos e construtores, a partir de cada declaração, gere sua sintaxe que deve ser [nome da classe]. [nome do método].[declaração])<br>c. Instruções são alocações, assinatura e invocações de métodos, para cada instrução, gere sua sintaxe que deve ser [nome da classe]. [nome do método].[instrução]. |

| | |
|---|---|
| | d. (obs) Desconsidere condições, loops, retornos e importações. |

**Fonte: Elaborado pelo autor**

**Tabela 30 - Descrição do roteiro da atividade "Executar algoritmo de recuperação de caso de uso"**

<table>
<tr><td>Atividade</td><td>Executar algoritmo para análise estática</td></tr>
<tr><td>Roteiro</td><td>1. Criar uma tabela com a seguinte estrutura de colunas:<br><table><tr><th colspan="3">Tabela do Algoritmo de Recuperação de Caso de Uso</th></tr><tr><td>Caso de Uso</td><td>Dependências</td><td>Generalizações</td></tr></table><br>2. A partir do código abstrato, aplicar o algoritmo de recuperação de caso de uso:<br>3. Para os casos de uso, generalizações e dependências identificados no passo 2, insira um registro na tabela criada no passo 1</td></tr>
</table>

**Fonte: Elaborado pelo autor**

### 4.1.4 Roteiro para recuperação de diagrama de sequência

**Tabela 31 - Descrição do roteiro da atividade "Gerar código abstrato"**

| Atividade | Gerar Código Abstrato |
|---|---|
| Roteiro | 1. Abra o visual studio, e na janela de “explorador da solução” Identificar classes que manipulam as classes identificadas correspondentes aos requisitos (Classes de controller e services), ex: public class [Nome da classe][Service]).<br>2. Para cada classe identificada, gerar um código abstrato de cada classe com as seguintes considerações:<br>a. Uma classe possui 0 ou muitas declarações seguida de 0 ou muitas instruções.<br>b. Declarações são declarações de atributos, métodos e construtores, a partir de cada declaração, gere sua sintaxe |

| | |
|---|---|
| | que deve ser [nome da classe]. [nome do método].[declaração])<br>c. Instruções são alocações, assinatura e invocações de métodos, para cada instrução, gere sua sintaxe que deve ser [nome da classe]. [nome do método].[instrução].<br>d. (obs) Desconsidere condições, loops, retornos e importações. |

**Fonte: Elaborado pelo autor**

**Tabela 32 - Descrição do roteiro da atividade "Gerar diagrama de fluxo de objetos"**

| Atividade | Gerar diagrama de fluxo de objetos |
|---|---|
| Roteiro | 1. Abra o Microsoft Visio<br>2. Crie um novo grafo<br>3. Identifique as declarações de objetos e instruções do código abstrato<br>4. Crie um nó no selecionando na caixa de ferramentas do Visio para cada objeto declarado e para cada instrução que modifique o fluxo deste objeto<br>5. Para cada relacionamento entre as declarações e as instruções que modificam este objeto, crie uma ponta entre os nós selecionando caixa de ferramentas do Visio. |

**Fonte: Elaborado pelo autor**

**Tabela 33 - Descrição do roteiro da atividade "Construir diagrama de sequência"**

| Atividade | Construir diagrama de sequência |
|---|---|
| Roteiro | 1. Crie uma tabela com as seguintes colunas (Linha, Chamada, Fontes, Alvos)<br>2. Analisar chamadas de métodos no OFG, defina o objeto alvo (objeto que chama o método) e os alvos (objetos influenciados pela chamada do método)<br>3. Analisar o método, mapeando as ocorrências de loops e condições |

| | |
|---|---|
| | 4. Inserir marcas (if, loop, alt, opt ) entre o grupo de mensagens respectivos. |

**Fonte: Elaborado pelo autor**

### 4.1.5 Roteiro para construção de uma descrição arquitetural.

**Tabela 34 - Descrição da atividade "Construir descrição arquitetural"**

| **Atividade** | Construir descrição arquitetural (Apêndice 1) |
|---|---|
| Tarefas | 1. Baixar o documento de modelo de descrição arquitetural que se encontra no site da ISO 42010<br>2. Abrir o documento no Microsoft Word<br>3. Remover as instruções do documento<br>4. Definir o tipo de descrição arquitetural<br>5. Definir o propósito do sistema<br>6. Definir as características do sistema<br>7. Definir os Stakeholders<br>8. Inserir a visão computação<br>9. Inserir os modelos da visão computação<br>10. Inserir o ponto de vista computacional<br>11. Inserir os modelos do ponto de vista computacional<br>12. Escrever notas com considerações sobre o documento<br>13. Inserir a bibliografia<br>14. Remover outros itens que não foram descritos no documento. |
| **Artefato de entrada** | Modelos da visão computação |
| **Artefato de saída** | Descrição Arquitetural |

**Fonte: Elaborado pelo autor**

### 4.1.6 Roteiro para especificação da visão computação

**Tabela 35 - Descrição do roteiro da atividade "Construir estrutura da visão computação"**

| Atividade | Construir estrutura da visão computação (Item 4.1.5 do apêndice 1) |
|---|---|
| Roteiro | 1. Abra o software StarUML.<br>2. Crie um novo projeto chamado: “Descrição arquitetural”<br>3. Crie um pacote com o nome “Especificação visão computação”<br>4. Crie um diagrama de classes e insira o pacote criado no passo 3.<br>5. Crie quatro pacotes dentro do pacote criado no passo 3 com os nomes: “Template de objetos”, ”Assinaturas de interface”, ”Template de interfaces”, ”Tipos de dados” |

**Fonte: Elaborado pelo autor**

**Tabela 36 - Descrição do roteiro da atividade "Construir o diagrama de template de objetos"**

| Atividade | Construir o diagrama de template de objetos (Item 4.1.2 do apêndice 1) |
|---|---|
| Roteiro | 1. Abra o projeto “Descrição arquitetural”<br>2. Crie um diagrama de classes dentro do pacote chamado “Especificação visão computação” chamado “Template de objetos”<br>3. Identifique todas as classes correspondentes aos casos de uso recuperados na etapa 1 do processo.<br>4. Para cada classe identificada no passo 3, crie uma classe com o estereótipo <<CV_Object>> |

**Fonte: Elaborado pelo autor**

**Tabela 37 - Descrição do roteiro da atividade "Construir o diagrama de assinaturas"**

| Atividade | Construir o diagrama de assinaturas (Item 4.1.5 do apêndice 1) |
|---|---|

| | |
|---|---|
| Roteiro | 1. Abra o projeto “Descrição arquitetural”<br>2. Crie um diagrama de classes dentro do pacote chamado “Especificação visão computação” chamado “Diagrama de assinaturas”<br>3. Identifique todas as classes correspondentes aos casos de uso recuperados na etapa 1 do processo.<br>4. Para cada classe identificada no passo 3, crie uma classe com o estereótipo <<CV_Signature>><br>5. Para cada uso correspondente as classes identificadas no passo 3, crie um método dentro da classe <<CV_Signature>> correspondente. |

**Fonte: Elaborado pelo autor**

**Tabela 38 - Descrição do roteiro da atividade "Construir o diagrama de tipo de dados"**

| Atividade | Construir o diagrama de tipo de dados (Item 4.1.4 do apêndice 1) |
|---|---|
| Roteiro | 1. Abra o projeto “Descrição arquitetural”<br>2. Crie um diagrama de classes dentro do pacote chamado “Especificação visão computação” chamado “Diagrama de tipos de dados”<br>3. Identifique as classes do esquema orientado à objetos.<br>4. Identifique os parâmetros das interações dos diagramas de sequência que o nome seja diferente de alguma classe do esquema orientado à objetos, e que não seja tipo primitivo.<br>5. Para cada classe identificada no passo 3, crie uma classe com o estereótipo <<CV_DataType>>, e copie os atributos de forma idêntica.<br>6. Para cada item identificado no passo 4, crie uma classe com o mesmo estereótipo do passo 5.<br>7. Se houver algum relacionamento de herança ou agregação no esquema orientado à objetos, copie o mesmo relacionamento entre as classes correspondentes no diagrama de tipos de dados |

**Fonte: Elaborado pelo autor**

**Tabela 39 - Descrição do roteiro da atividade "Construir o diagrama de comportamento"**

| Atividade | Construir o diagrama de comportamento |
|---|---|
| Roteiro | 1. Abra o projeto “Descrição arquitetural”<br>2. Selecione uma operação do diagrama de assinaturas.<br>3. Crie um diagrama de sequência com o mesmo nome da operação selecionada no passo 2.<br>4. A partir do diagrama de sequência recuperado da operação na etapa 1, especifique o novo diagrama de sequência nas seguintes considerações<br>1. Para cada ciclo de vida, crie um ciclo de vida com o mesmo nome, mas com o estereótipo de <<CV_Object>>, para correlacionar com o objeto criado no diagrama de “Templates de objetos”.<br>2. Copie o fluxo de forma idêntica ao diagrama de sequencias recuperado, incluindo loops e condições.<br>3. Copie as mensagens e iterações entre os ciclos de vida de forma identica ao diagrama recuperado. |

**Fonte: Elaborado pelo autor**

### 4.1.7 Resultados Obtidos

A aplicação do roteiro foi bem-sucedido, pois foi possível construir uma descrição arquitetural da visão computacional da parte do sistema analisado e foi possível identificar os locais de impacto pela implementação dos requisitos informados pelo cliente. Os resultados desta aplicação estão no apêndice deste trabalho, que é a descrição arquitetural do software. Uma facilidade para a aplicação do processo neste caso´, dá-se pelo fato da facilidade de entendimento do código fonte, pois seguia os padrões de escrita da linguagem, facilitando a identificação dos serviços referentes às entidades de domínio. Um obstáculo para a aplicação foi a necessidade de aplicar o processo manualmente, sendo que nenhum passo foi possível de automatizar através de ferramentas de análise de código ou construção da descrição arquitetural. A

descrição arquitetural foi construída e adaptada com base no modelo de documento fornecido pelo site da ISO 42010. Apesar da norma citar a necessidade da validação da arquitetura, está atividade não foi possível de realizar pois o roteiro é focado apenas na etapa de recuperação e especificação da arquitetura.

# 5 CONCLUSÕES

A contribuição deste trabalho foi apresentar quais informações do código fonte são necessárias para realizar um processo de documentar arquiteturas de softwares que já foram construídos. Entre as aplicações deste trabalho, estão sistemas *open-source* muito utilizados e bem avaliados por comunidades de desenvolvimento de software, e softwares legados de organizações, entretanto, a aplicabilidade deste trabalho depende das técnicas de engenharia reversa utilizadas, mas às técnicas podem ser alteradas, possibilitando identificar outras informações para contribuir com a descrição arquitetural, desde que sigam o modelo *Horseshoe*. Foi identificado que as informações obtidas a partir dos modelos construídos neste trabalho, são alguns dos elementos que possibilitam a construção de uma fábrica para tratamento de código *open-source* no futuro.

Os resultados deste trabalho mostram que o objetivo foi alcançado, sendo possível criar uma descrição arquitetural dos serviços computacionais de um sistema a partir do código fonte, permitindo que a equipe de desenvolvimento e novos integrantes utilizem um meio comum para entender quais serviços o software oferece e como estes serviços estão organizados e como foram implementados, com isso, é possível analisar se o sistema atende precisamente as necessidades de futuros Stakeholders e medir o impacto de futuras mudanças no software durante o seu ciclo de vida, sendo assim, a descrição arquitetural atendeu as necessidade dos stakeholders do projeto.

As limitações para este trabalho, são relacionadas com as limitações das técnicas de engenharia reversa utilizadas, pois suas aplicações são limitadas apenas a softwares orientados à objetos e a banco de dados relacionais, além disso, caso o software não utilize adequadamente os conceitos da orientação objeto e padrões de projeto, haverá dificuldades em aplicar as técnicas apresentadas, sendo assim, será necessário utilizar outras técnicas que atendam às necessidades de cada software que será analisado.

## REFERÊNCIAS BIBLIOGRÁFICAS

BORSOI, B. **Arquitetura de processos aplicada na integração de fábricas de software.** Tese (doutorado em engenharia elétrica) – Universidade de São Paulo. 2008.

BREIVOLD, H.; CRNKOVIC, L.; LARSSON, M. **A Systematic Review of Software Architecture Evolution Research**. 2011.

CHADHA, D. **Emergence of Software Product Line**. International Journal of Computer Applications® (IJCA). 2012

DIAS, L.D. **Método de instanciação de uma arquitetura de processos aplicado em fábrica de software**. Dissertação (Mestrado em Engenharia Elétrica) – Universidade de São Paulo. 2010

DING, W.; et al. **How Do Open Source Communities Document Software Architecture: An Exploratory Survey**. Engineering of Complex Computer Systems (ICECCS). 2014

Information technology — Open Distributed Processing — Use of UML for ODP system specifications. **ISO/IEC/IEEE 19793**. 2015.

KILOV, H. et al. **The Reference Model of Open Distributed Processing: Foundations, experience and applications**. Computer Standards & Interfaces. v. 35. .2012

LAGUNA, M.A.; HERNANDEZ, C. **A Software Product Line Approach for E-Commerce Systems**. E-Business Engineering (ICEBE). 2010.

LININGTON, P.; MILOSEVIC, Z.; TANAKA, A.; VALLECILLO, A. **Building Enterprise Systems with ODP – An Introduction to Open Distributed Processing**. Boca Raton: Chapman and Hall/CRC. 2011.

PEREIRA, C.; MARTINEZ, L.; FAVRE, L. **Recovering Use Case Diagrams from Object Oriented Code: an MDA-based Approach**. Information Technology: New Generations (ITNG), 2011

RAMANATHAN, S.; HODGES, J. **Reverse Engineering Relational Schemas to Object-Oriented Schemas.** 1996.

ROZANSKI, N.; WOODS E. **Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.** Boston: AddisonWesley. 2005.

ROMERO, J.R.; VALLECILLO, A. **Modeling the ODP Computational Viewpoint with UML 2.0: The Templeman Library Example.** Enschede: Workshop on ODP for Enterprise Computing. 2005

Systems and Software Engineering -- Architecture Description. **ISO/IEC/ IEEE 42010**. 2011.

Software Engineering ﻩ Software Life Cycle Processes ﻩ Maintenance. **ISO/IEC 14794.** 2006.

TONELLA, P. **Reverse Engineering of Object Oriented Code**. ICSE '05 Proceedings of the 27th international conference on Software engineering. 2005.

TRIPATHY, P.; NAIK, K. **Software Evolution and Maintenance: a practitioner's approach**. Hoboken: John Wiley & Sons, Inc. 2014.

**APENDICE A – Descrição arquitetural Arquitetura computacional para Sistema de ecommerce.**

# Descrição arquitetural

# Arquitetura computacional para

# Sistema de ecommerce

**Conteúdo**

**1** Introdução

Este capítulo descreve itens de informações introdutórias da descrição arquitetural, incluindo a identificação e as informações complementares.

## 1.1 Informação de identificação da arquitetura

A arquitetura descrita neste documento é a arquitetura no ponto de vista computacional do ODP de um Sistema de comércio eletrônico (E-commerce), apresentando as funcionalidades que o Sistema possuí, o comportamento e estrutura destas funcionalidades

## 1.2 Histórico de versões

| Data | Versão | Descrição | Autor |
|---|---|---|---|
| 10/02/2016 | 1.0 | Criação do documento | Leonardo Gasparini Romão |
| | | | |

## 1.3 Contexto

O Cenário do sistema desta descrição arquitetural, envolve uma organização do setor comercial que deseja inserir um sistema de comércio eletrônico em seu ambiente, entretanto, esta organização não possuí uma área de TI preparada para desenvolvimento de aplicações corporativas, portanto, foi necessário que esta empresa contratasse os serviços terceirizados de outra empresa especializada em soluções e consultoria em TI.

A empresa contratada fechou um acordo para criar o sistema, e como solução, apresentou um sistema de comércio eletrônico *open-source*, afirmando que o sistema apoiava diversas funcionalidades que a organização desejava e que permitiam diversas configurações para estar de acordo com as necessidades da organização

após ler às informações que o sistema possuía em seu site e testar o sistema em seu ambiente. O sistema apresenta as seguintes características técnicas

**Outras informações**

- O sistema é *open source*
- O paradigma do sistema Orientado à Objetos.
- O sistema é uma aplicação *web*, utilizando o framework ASP.NET MVC que utiliza a linguagem C#.
- O sistema utiliza um banco de dados relacional, criado na tecnologia SQL 2008 R2.
- O sistema, é hospedado no servidor IIS.
- Como artefatos do sistema, o desenvolvedor possuí o código fonte e o script de criação do esquema SQL do banco de dados

# 2 Stakeholders e preocupações

Este capítulo contém informações apresentando as partes interessadas da arquitetura, suas respectivas preocupações, e à rastreabilidade das preocupações com as partes interessadas.

## 2.1 Stakeholders

Os Stakeholders deste projeto são:

- Desenvolvedores do Sistema;
- Analistas do sistema
- Analistas de manutenção do sistema
- Arquiteto de software
- Dono do projeto

## 2.2 Preocupações

- Identificar as entidades de domínio do sistema
- Identificar e entender as funcionalidades do Sistema
- Mapear as características do sistema
- Entender a estrutura e comportamento do Sistema

- Modificar, adaptar ou evoluir o Sistema
- Realizar manutenções no Sistema

## 2.3 Traceabilidade Preocupações – Stakeholders

| **Preocupações** | **Equipe de desenvolvimento** | **Cliente do projeto** |
|---|---|---|
| **Entender as funcionalidades do Sistema** | X | X |
| **Mapear das características do sistema** | X | X |
| **Entender a estrutura e comportamento do Sistema** | X | - |
| **Modificar, adaptar ou evoluir o Sistema** | X | - |
| **Realizar manutenções no Sistema** | X | - |

# 3 Pontos de vista

## 3.1 Ponto de vista Computacional

## 3.2 Visão Geral

## 3.3 Preocupações e Stakeholders

### 3.3.1 Preocupações

- Mapear as características do sistema
- Entender a estrutura e comportamento do Sistema
- Modificar, adaptar ou evoluir o Sistema
- Realizar manutenções no Sistema

### 3.3.2 Stakeholders típicos

- Desenvolvedores do Sistema;
- Analistas do sistema
- Analistas de manutenção do sistema
- Arquiteto de software

## 3.4 Tipos de modelos

## 3.5 Modelo de objetos

O modelo de objetos apresenta de forma geral, quais são os objetos computacionais especificados.

### 3.5.1 Convenções do modelo de objetos

As convenções deste modelo são baseadas na norma ISO 19793 UML4ODP ou Use of Uml for ODP

## 3.6 Modelo de interfaces

O modelo de interfaces, apresenta quais interfaces os objetos computacionais possuem e como eles estão relacionados.

### 3.6.1 Convenções do modelo de interfaces

As convenções deste modelo são baseadas na norma ISO 19793 UML4ODP ou Use of Uml for ODP

## 3.7 Modelo de assinaturas de interface

O modelo de assinatura de interfaces, representa quais assinaturas cada interface possuí, mostrando quais os contratos ou funções que cada interface oferece, detalhando as interfaces especificadas no modelo de template de interfaces

### 3.7.1 Convenções do modelo de assinaturas de interface

As convenções deste modelo são baseadas na norma ISO 19793 UML4ODP ou Use of Uml for ODP

## 3.8 Modelo de tipos de dados

O modelo de tipos de dados apresenta a estrutura e conteúdo dos objetos que são manipulados através das interfaces e funcionalidades especificadas no ponto de vista computacional.

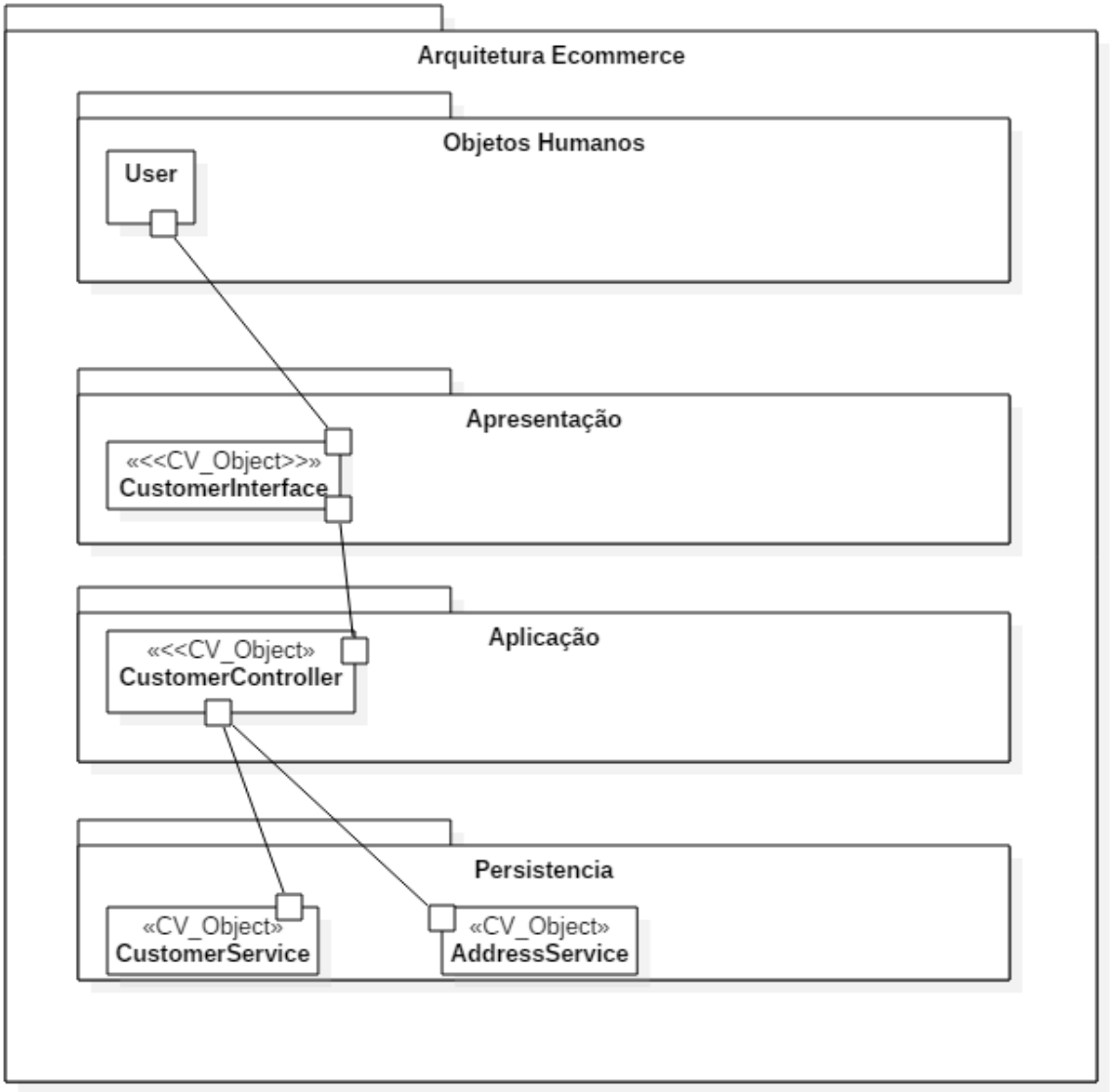
### 3.8.1 Convenções do modelo de tipos de dados

As convenções deste modelo são baseadas na norma ISO 19793 UML4ODP ou Use of Uml for ODP

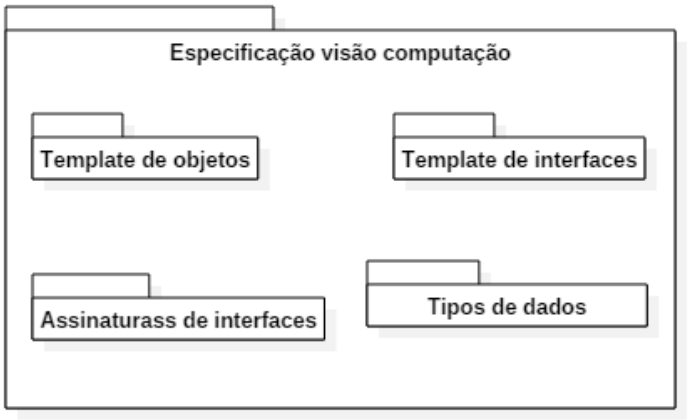
## 3.9 Notas

# 4 Visões

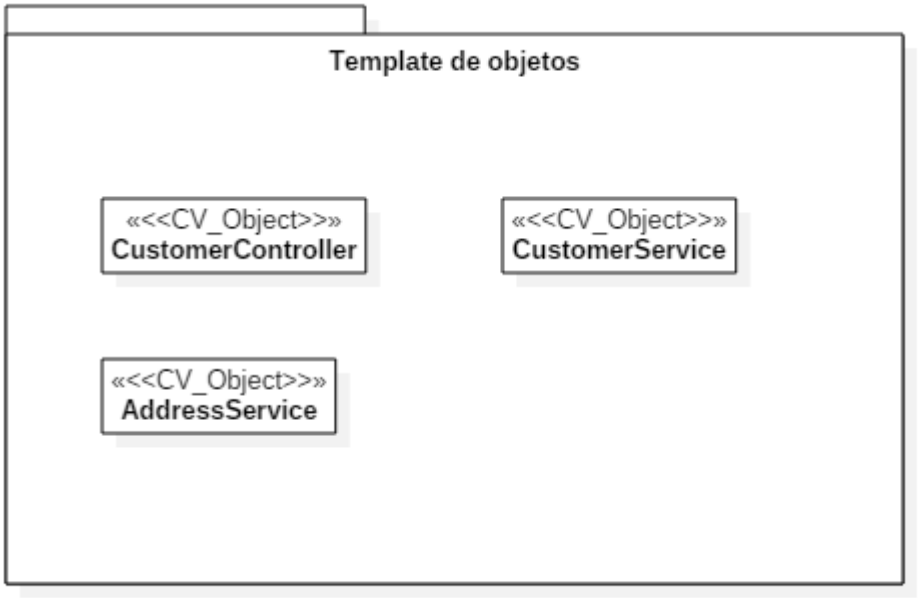
## 4.1 Visão: Computação



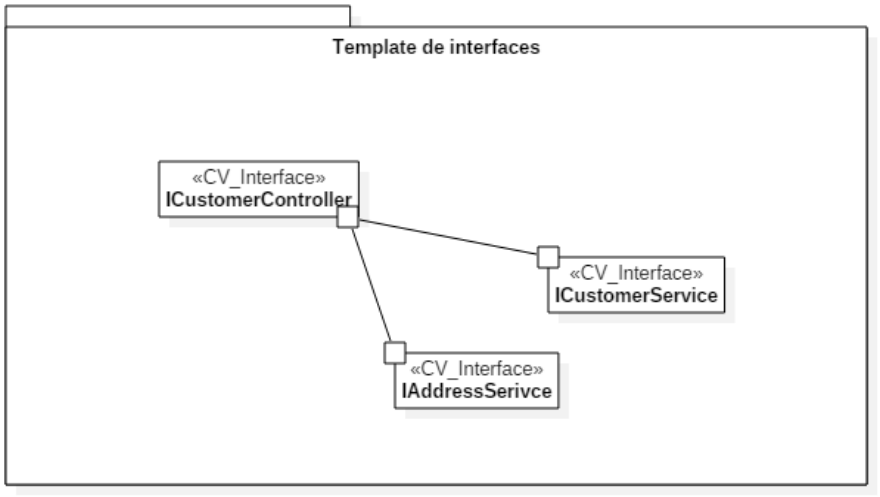
### 4.1.1 Modelos



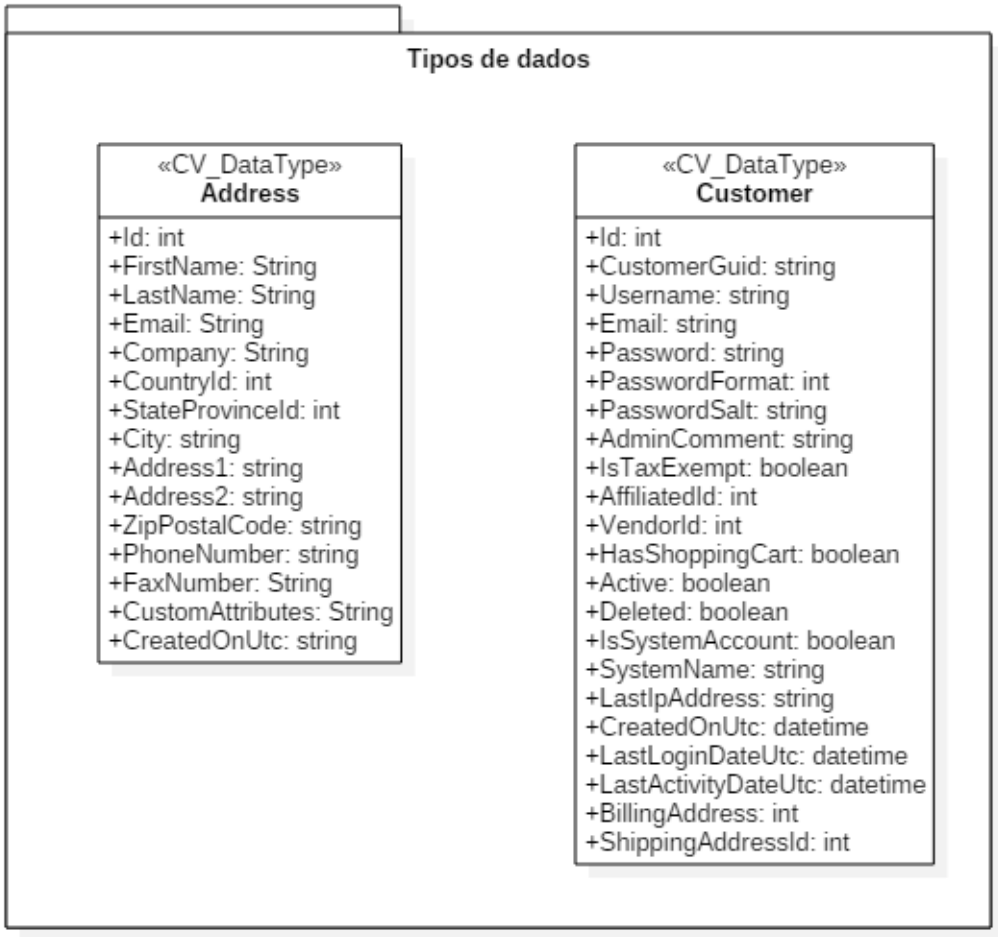

### 4.1.2 Modelo de objetos



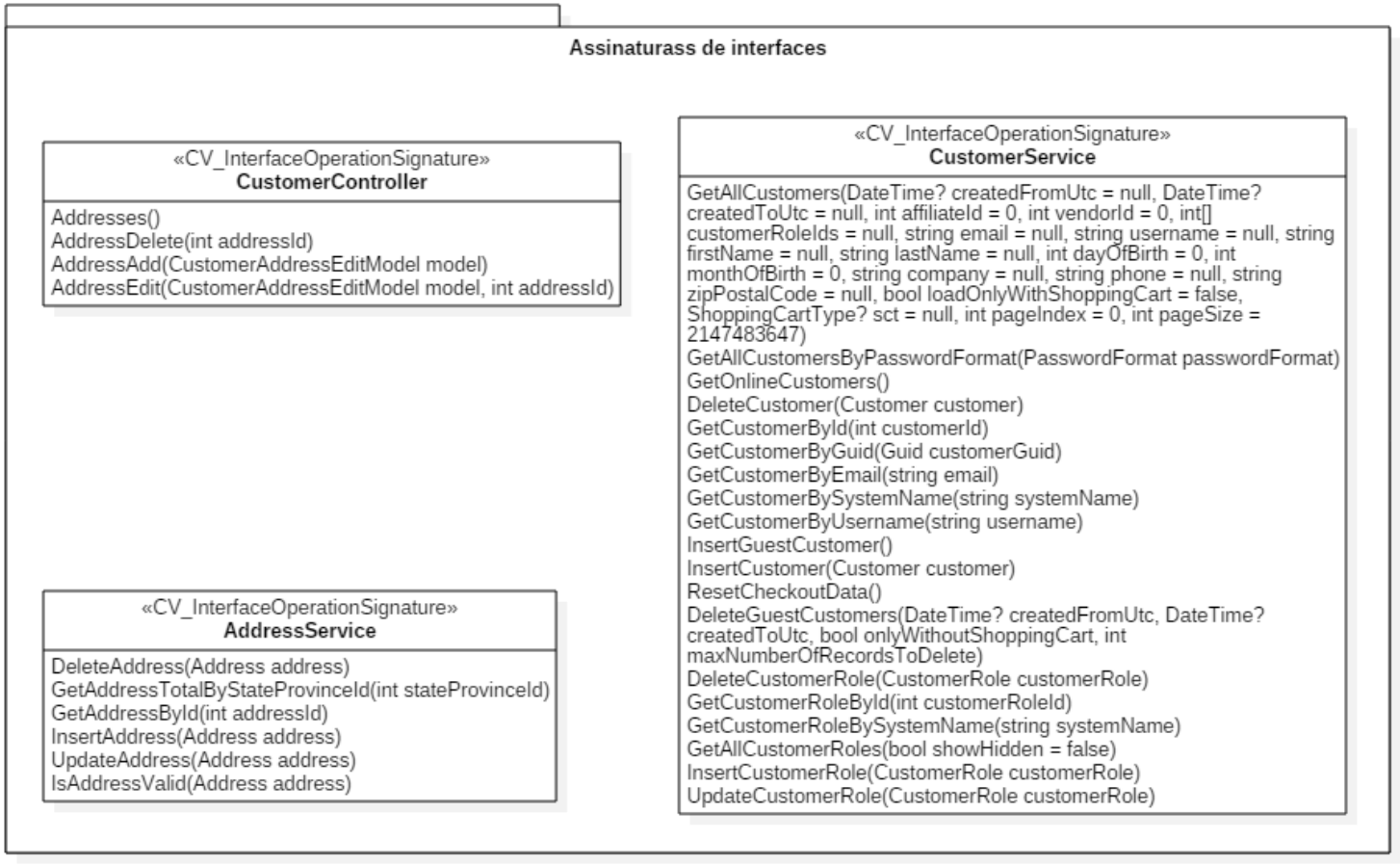
### 4.1.3 Modelo de interfaces



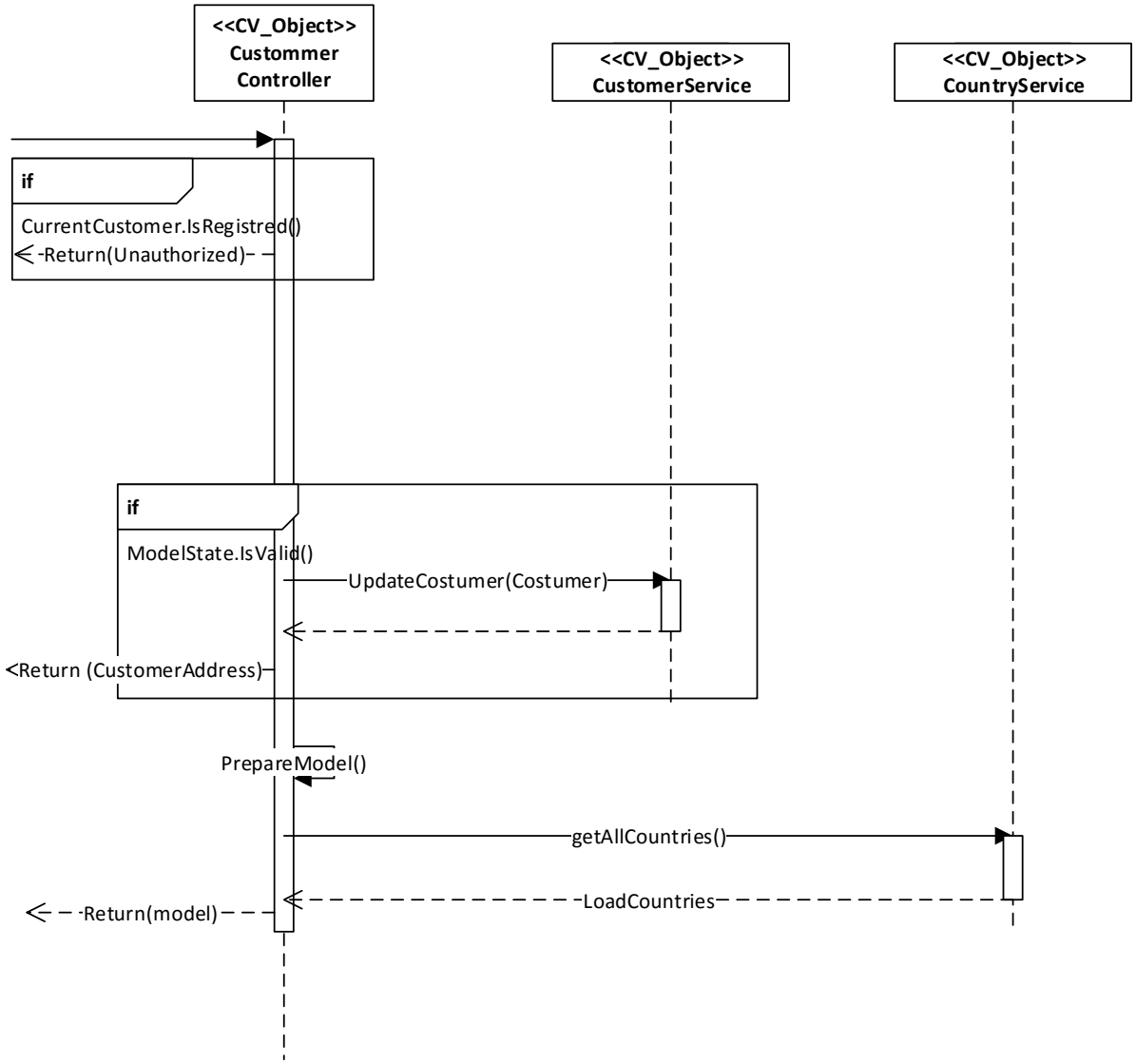

### 4.1.4 Modelo de tipos de dados



### 4.1.5 Modelo de assinaturas de interface



### 4.1.6 Comportamento da assinatura de interface (AddressAdd)



## REFERÊNCIAS BIBLIOGRÁFICAS


BORSOI, B. **Arquitetura de processo aplicada na integração de fábricas de software.** Tese (doutorado em engenharia elétrica) – Universidade de São Paulo. 2008.

BREIVOLD, H.; CRNKOVIC, L.; LARSSON, M. **A Systematic Review of Software Architecture Evolution Research**. 2011.

CHADHA, D. **Emergence of Software Product Line**. International Journal of Computer Applications® (IJCA). 2012

DIAS, L.D. **Método de instanciação de uma arquitetura de processos aplicado em fábrica de software**. Dissertação (Mestrado em Engenharia Elétrica) – Universidade de São Paulo. 2010

DING, W.; et al. **How Do Open Source Communities Document Software Architecture: An Exploratory Survey**. Engineering of Complex Computer Systems (ICECCS). 2014

Information technology — Open Distributed Processing — Use of UML for ODP system specifications. **ISO/IEC/IEEE 19793**. 2015.

KILOV, H. et al. **The Reference Model of Open Distributed Processing: Foundations, experience and applications**. Computer Standards & Interfaces. v. 35. .2012

LAGUNA, M.A.; HERNANDEZ, C. **A Software Product Line Approach for E-Commerce Systems**. E-Business Engineering (ICEBE). 2010.

LININGTON, P.; MILOSEVIC, Z.; TANAKA, A.; VALLECILLO, A. **Building Enterprise Systems with ODP – An Introduction to Open Distributed Processing**. Boca Raton: Chapman and Hall/CRC. 2011.

PEREIRA, C.; MARTINEZ, L.; FAVRE, L. **Recovering Use Case Diagrams from Object Oriented Code: an MDA-based Approach**. Information Technology: New Generations (ITNG), 2011

RAMANATHAN, S.; HODGES, J. **Reverse Engineering Relational Schemas to Object-Oriented Schemas.** 1996.

ROZANSKI, N.; WOODS E. **Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.** Boston: AddisonWesley. 2005.

ROMERO, J.R.; VALLECILLO, A. **Modeling the ODP Computational Viewpoint with UML 2.0: The Templeman Library Example.** Enschede: Workshop on ODP for Enterprise Computing. 2005

Systems and Software Engineering -- Architecture Description. **ISO/IEC/ IEEE 42010**. 2011.

Software Engineering ﻩ Software Life Cycle Processes ﻩ Maintenance. **ISO/IEC 14794.** 2006.

TONELLA, P. **Reverse Engineering of Object Oriented Code**. ICSE '05 Proceedings of the 27th international conference on Software engineering. 2005.

TRIPATHY, P.; NAIK, K. **Software Evolution and Maintenance: a practitioner's approach**. Hoboken: John Wiley & Sons, Inc. 2014.