UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

UM ESTUDO AVALIATIVO DA QUALIDADE DE ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE AGREGANDO-SE DESCRIÇÃO EM LINGUAGEM OCL

RICARDO FRANCISCATO

ORIENTADOR: PROF. DR. LUIZ EDUARDO GALVÃO MARTINS

PIRACICABA, SP
2006

UNIVERSIDADE METODISTA DE PIRACICABA
FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

UM ESTUDO AVALIATIVO DA QUALIDADE DE ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE AGREGANDO-SE DESCRIÇÃO EM LINGUAGEM OCL

RICARDO FRANCISCATO

ORIENTADOR: PROF. DR. LUIZ EDUARDO GALVÃO MARTINS

Dissertação apresentada ao Mestrado em Ciência da Computação, da Faculdade de Ciências Exatas e da Natureza, da Universidade Metodista de Piracicaba – UNIMEP, como requisito para obtenção do Título de Mestre em Ciência da Computação.

PIRACICABA, SP
2006

UM ESTUDO AVALIATIVO DA QUALIDADE DE ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE AGREGANDO-SE DESCRIÇÃO EM LINGUAGEM OCL

RICARDO FRANCISCATO

ORIENTADOR: **Prof. Dr. LUIZ EDUARDO GALVÃO MARTINS.**

Dissertação de Mestrado defendida e aprovada em 22 de fevereiro de 2006, pela Banca Examinadora constituída dos Professores:

Prof. Dr. LUIZ EDUARDO GALVÃO MARTINS

UNIMEP

Prof. Dr. LUIZ CAMOLESI JÚNIOR

UNIMEP

Prof. Dr. Carlos Miguel Tobar Toledo

PUCCamp

À

Minha namorada Rosimeire pelo apoio e compreensão e

aos

Meus pais Jair e Sonia

AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus que meu deu, entre tantas outras coisas, saúde e perseverança para a elaboração deste trabalho.

Agradeço também a todas as pessoas que estavam ao meu redor, como meus familiares, minha namorada e meus amigos e amigas que ajudaram a conseguir realizar mais uma etapa de minha vida.

Agradeço em especial ao professor Luiz Eduardo Galvão Martins a orientação, compreensão e incentivo dispensado ao desenvolvimento de todas as etapas deste trabalho.

Resumo

No dia-a-dia das empresas que desenvolvem software, é comum o questionamento sobre a prática sugerida pela metodologia de Engenharia de Software para sua documentação. Este trabalho mostra as problemáticas que podem ocorrer na documentação de um software, como a ambigüidade, a imprecisão e a incompletude que essa documentação pode apresentar. Nesse contexto, é fundamental utilizar as metodologias propostas pela Engenharia de Requisitos na elaboração de software, tentando assim minimizar esses problemas. Neste trabalho foi realizado um estudo de caso, no qual se fez uma avaliação sobre a qualidade da especificação de um software e agregou-se a essa especificação uma descrição chamada OCL (Object Constraint Language). A OCL tem como objetivo descrever na especificação as restrições do software utilizando uma linguagem formal, próxima da codificação, mas sem a complexidade de uma linguagem de programação. No estudo de caso, após a especificação de um software comercial, uma avaliação foi realizada através de um questionário que foi respondido por profissionais da área de informática, para avaliar o grau de melhoria que foi obtido através da utilização dessa nova linguagem.

Palavras-Chaves: OCL, Engenharia de Requisitos, UML, Especificação de Requisitos.

Abstract

On a daily basis, in software development companies, it is very common the questioning of which software engineering methodology should be used for documentation. It is the purpose of this job to show the problems that can occur in the process of software documentation as, ambiguity, inaccuracy and a lack of completeness that this documentation can present. In this context, it is fundamental to use the methodologies proposed by the Requirement Engineering when developing software, in order to minimize these problems. A case study was done in this job in which an evaluation was made of the quality of a software specification, and a description OCL (Object Constraint Language) was added to this specification. The objective of OCL is to specify constraints on software using a formal language, close to programming, but without the complexity of a programming language. In the case study, after a commercial software was specified, an evaluation was made through a questionnaire answered by professionals from the information technology area to evaluate the level of improvement obtained through the use of this new language.

Key Words – OCL, Requirement Engineering, UML, Requirement Specifications

Índice

Índice de Figuras

Índice de Tabelas

Índice de Quadros

CAPÍTULO 1 – INTRODUÇÃO

1.1 – CONTEXTUALIZAÇÃO DA PESQUISA

No contexto de desenvolvimento de software, os aspectos não funcionais e organizacionais, em geral, não recebem tratamento preciso. Vários aplicativos oferecem suporte formal às técnicas gráficas de modelagem de requisitos (documentação dos requisitos), uma vez que as descrições contidas nesses métodos são, normalmente, ambíguas e imprecisas, permitindo diferentes interpretações para uma mesma especificação. Importantes informações podem estar escondidas entre detalhes considerados irrelevantes, podendo ocorrer que falhas de requisitos só sejam descobertas muito mais tarde, ou seja, na implementação dos sistemas, tornando suas correções muito caras.

Apesar de muitos ainda considerarem projetos de sistemas aplicativos a personificação do progresso tecnológico, são os usuários experientes que conhecem a realidade de um projeto, e que enfrentam a imaturidade dos engenheiros de sistemas que não adotam ou até mesmo desconhecem um processo de engenharia de software, capaz de organizar e conduzir suas atividades no processo de desenvolvimento.

Dessa forma, diversos são os problemas que afligem o desenvolvimento de software, e neste contexto, a etapa de definição de requisitos é considerada uma atividade importante, decisiva e ao mesmo tempo crítica do desenvolvimento de software. A Engenharia de Requisitos procura sistematizar o processo de definição de requisitos.

1.2 – OBJETIVOS

Este trabalho tem como objetivo avaliar melhorias na especificação de requisitos de software com a utilização das descrições da linguagem OCL (*Object Constraint*

Language) (OMG, 2003). As melhorias esperadas referem-se ao aumento da precisão e da completude da especificação de requisitos, e a diminuição da ambigüidade da mesma.

Para efeito de avaliação podemos definir que uma especificação de requisitos é ambígua quando nela se constata mais de uma interpretação de seus requisitos. Um outro aspecto é a completude da especificação dos requisitos, o qual deve oferecer informações sobre todas as funções a serem implementadas no software. E por final, uma especificação de requisitos é precisa quando descreve o que se espera que seja implementado no software.

1.3 – Importância e Justificativa da Pesquisa

Um dos grandes problemas encontrados em uma especificação de software é que a documentação geralmente traz ambigüidades nas informações, ou seja, pode apresentar várias interpretações, isso porque os tipos de documentações conhecidas para especificação de software, como por exemplo diagramas de classe, cenário de atividades ou qualquer outro tipo de documentação reconhecida na literatura, não faz um tratamento satisfatório no que diz respeito às restrições da especificação.

1.4 – Metodologia do trabalho

A metodologia desse trabalho iniciou com uma revisão bibliográfica sobre Engenharia de Requisito e sobre a descrição da linguagem OCL. Depois de realizar essa revisão, um estudo de caso foi desenvolvido.

No estudo de caso foi gerada uma especificação de um software, a qual foi elaborada de duas maneiras, sendo que uma das especificações foi realizada com os padrões da Engenharia de Requisitos, e à outra especificação foi agregada um item a mais, as descrições da linguagem OCL.

Depois da elaboração da especificação, um questionário foi criado para que pessoas da área de informática pudessem avaliar as melhorias que a aplicação da descrição da linguagem OCL teve na especificação. Esse questionário tenta avalizar as melhorias da completude e da precisão da especificação e a diminuição da ambigüidade que a mesma pudesse apresentar.

1.5 – Resultados e Contribuições

Utilizando OCL, pretende-se construir especificação de software com maior clareza, objetividade e completude.

Com uma documentação mais completa e precisa, é mais provável que o software desenvolvido atinja os resultados esperados pelo usuário. Deve-se considerar que, quando se tem uma estrutura de fácil compreensão, o desenvolvimento do software é mais rápido, sem contar que se conseguir fazer alterações ou modificações com maior rapidez.

1.6 – Organização do Trabalho

Este trabalho está organizado da seguinte forma:

Capítulo 1:	Contexto da pesquisa, os objetivos, importância, justificativa, resultados e contribuições esperados.
Capítulo 2:	Revisão bibliográfica da Engenharia de Requisitos.
Capítulo 3:	Revisão bibliográfica da OCL.
Capítulo 4:	Estudo de Caso
Capitulo 5:	Resultados e Discussões
Capítulo 6:	Conclusão

CAPÍTULO 2 – ENGENHARIA DE REQUISITOS

Esse capítulo tem como objetivo apresentar a metodologia que a Engenharia de Requisitos emprega na elaboração de software e todas as suas vertentes.

2.1 – DESENVOLVIMENTO DE SOFTWARE

O Processo de Desenvolvimento de Software é definido por Kotonya e Sommerville (1998) como um conjunto de atividades e resultados associados que produzem um produto de software.

Em meados dos anos 70, Schwartz (1975) já apontava como fases principais do processo de produção de um sistema de software:

- Especificação de Requisitos: tradução da necessidade ou requisito operacional para uma descrição da funcionalidade a ser executada.
- Projeto de Sistema: tradução dos requisitos em uma descrição de todos os componentes necessários para codificar o sistema.
- Programação (Codificação): produção do código que controla o fluxo do sistema e realiza a computação e lógica envolvidas.
- Verificação e Integração (*Checkout*): verificação da satisfação dos requisitos iniciais pelo produto produzido.

Para cada fase do processo de desenvolvimento de software existe uma série de atividades que são executadas. Estas atividades constituem um conjunto mínimo para se obter um produto de software, segundo Pressman (1997):

- *Engenharia de Sistema*: estabelece uma solução geral para o problema, envolvendo questões extra-software.

- *Análise de Requisitos*: levantamento das necessidades do software a ser implementado. A Análise tem como objetivo produzir uma especificação de requisitos, que convencionalmente é um documento.

- *Especificação de Sistema*: descrição funcional do sistema. Pode incluir um plano de testes para verificar adequação.

- *Projeto Arquitetural*: é desenvolvido um modelo conceitual para o sistema, composto de módulos, que na maioria das vezes, esses módulos são independentes.

- *Projeto de Interface*: cada módulo tem sua interface de comunicação estudada e definida.

- *Projeto Detalhado*: os módulos em si são definidos e possivelmente traduzidos para pseudocódigo.

Apesar de não conseguir eliminar todo o risco de desenvolver um software com erros, o uso de técnicas que visam garantir a qualidade do software, entre elas o uso sistemático da atividade de teste, torna possível aumentar o nível de confiança, evitando, assim, futuros problemas.

Segundo Kitchenham (1996), os desenvolvedores tendem a avaliar a qualidade do software considerando a interação entre a ferramenta de desenvolvimento de software e o resultado obtido com o produto final.

O Planejamento da Qualidade é uma ação gerencial cujo objetivo é incorporar as necessidades e desejos dos clientes aos processos que fazem parte do desenvolvimento de um produto.

Silva (2002) propõe aplicação dos princípios do Planejamento da Qualidade de Software voltado aos processos envolvidos na prestação do serviço de suporte técnico pós-venda, na fase de disponibilização do ciclo de vida de produção e o conseqüente favorecimento das ações centradas no cliente utilizando a

metodologia QFD (Quality Function Deployment – Desdobramento da Função Qualidade).

Segundo Barros (1993), no projeto R-Cycle, a qualidade percebida é agregada ao produto de software através de testes realizados com usuários e também através da re-alimentação dos processos de Produção, Disponibilização e Evolução (PDE) de software com os resultados destes testes e do acompanhamento do usuário no seu ambiente de uso.

2.2 – REQUISITOS DE SOFTWARE

Diversas são as definições de requisito, sendo direcionadas ao processo de desenvolvimento de sistemas.

Segundo Sommerville (2000), um requisito pode descrever uma:

- *Facilidade ao nível de usuário*; por exemplo, um corretor de gramática e ortografia;
- *Propriedade muito geral do sistema*; por exemplo, o sigilo de informações;
- *Restrição específica no sistema*; por exemplo, o tempo de varredura de um sensor;
- *Restrição no desenvolvimento do sistema*; por exemplo: a linguagem JAVA deverá ser utilizada para o desenvolvimento do sistema.

Segundo o padrão IEEE STD. 830-98 (1990), que descreve requisito como:

1.) Uma condição ou uma capacidade de que o usuário necessite para que solucione um problema ou alcance um objetivo;

2.) Uma condição ou uma capacidade que deve ser alcançada por um sistema ou componente de um sistema para satisfazer um contrato, um padrão, uma especificação ou outros documentos impostos formalmente;

3.) Uma representação documentada de uma condição ou capacidade, conforme os itens (1) e (2).

De acordo com Macaulay (1996), requisito é simplesmente algo de que o cliente necessita.

A Engenharia de Requisitos, segundo Leite (1994), pode ser definida como um processo segundo diferentes pontos de vista, no qual "o quê" é para fazer, é capturado e modelado. Neste processo são usados combinações de métodos, ferramentas e atores, sendo produzido, como resultado da modelagem, um documento de requisitos.

Diversas organizações tentam resolver as suas necessidades, utilizando modos de desenvolvimento de software diferentes. Entretanto, as diferenças entre esses processos de desenvolvimento normalmente emergem em processos detalhados, podendo ser descritos por modelos com regras gerais, segundo Kotonya e Sommerville (1998), que representa esse modelo conforme, a figura a seguir:

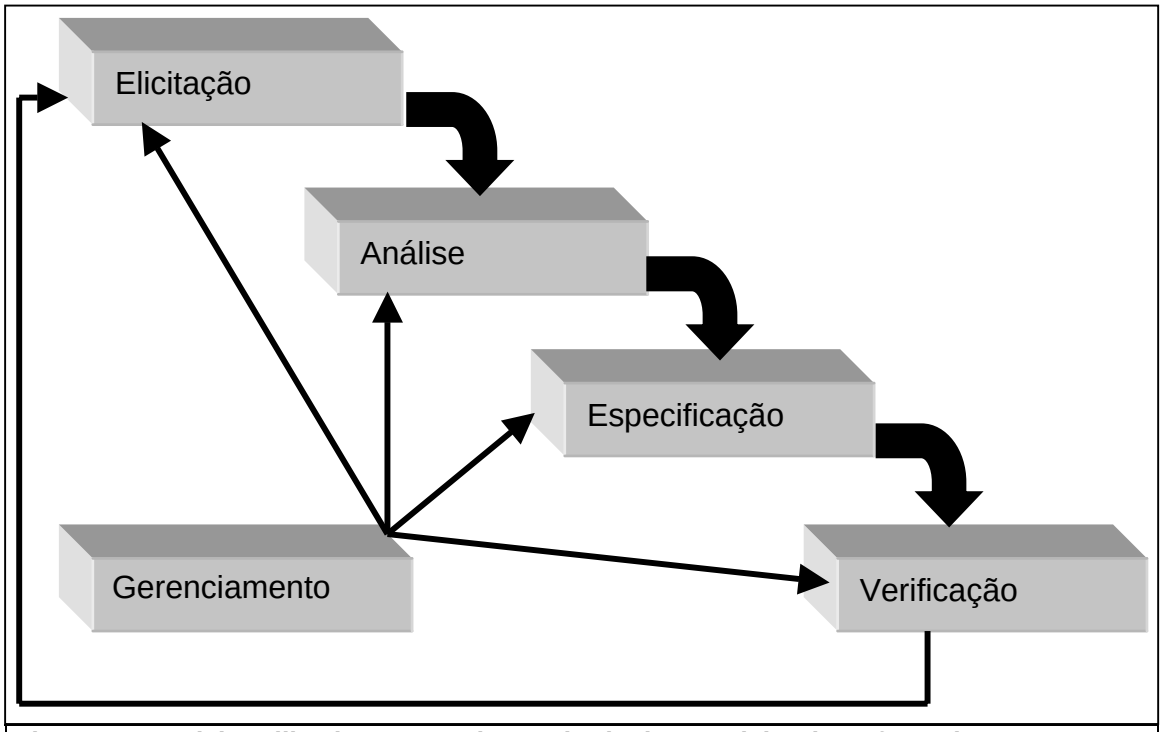

Figura 1 – Modelo utilizado em ERS (Engenharia de Requisito de Software)

O modelo representado acima será explicado nos tópicos 2.3, 2.4, 2.5 e 2.6 desse trabalho.

2.3 – ELICITAÇÃO E ANÁLISE DE REQUISITOS

A atividade de elicitação consiste na realização de ações para aquisição do conhecimento do domínio do problema. O objetivo principal é identificar e coletar informações em um domínio, que forneça o mais completo e correto entendimento das necessidades que demanda o software.

Segundo Goguen e Linde (1997), a elicitação é o processo pelo qual o cliente (comprador ou usuário) e o desenvolvedor (contratado) do sistema de software descobrem, revêem, articulam e entendem as necessidades dos clientes e restringem no software as atividades de desenvolvimento.

A análise de documentos é uma técnica usualmente aplicada na exploração do conhecimento escrito encontrado no universo de informações. Na modelagem dos

requisitos, segundo o modelo proposto na figura 1, essa técnica é muito útil para a definição dos objetos que compõem o modelo. A análise dos documentos permite um contato com o vocabulário utilizado no domínio do problema e auxilia na construção do glossário de termos especializados, que tem por objetivo definir os objetos e equalizar o conhecimento dos *stakeholders*[1], ou seja, a análise dos requisitos é o processo de analisar as necessidades do cliente e usuário para chegar a uma definição dos requisitos do software.

Na visão de Castro, Kolp e Mylopoulos (2002) a análise de requisitos é a fase mais importante do desenvolvimento dos sistemas. Nessa fase são feitas a descobertas das necessidades dos usuários e, conseqüentemente, as funcionalidades do sistema são modeladas através de um equilíbrio entre considerações técnicas, sociais e organizacionais. Não surpreendentemente, esta também é a fase onde os erros mais custosos são introduzidos no sistema.

Segundo Rocco e colegas (2002), a atividade de análise compreende examinar os modelos, buscando detectar e resolver inconsistências e omissões. Essa atividade antecede a atividade de verificação e validação, ao contrário desta última que conta com a participação dos clientes, é desempenhada pelos engenheiros de requisitos. A meta é analisar se o domínio do problema está representado coerentemente.

De acordo com Leite (2000), uma propriedade importante para uma análise bem sucedida é o rastreamento, que consiste na ligação da necessidade do domínio do problema ao requisito representado nos modelos. Esse rastreamento deve perdurar durante todo o ciclo de desenvolvimento, facilitando o gerenciamento do processo de produção e a verificação e a validação do software. Como requisitos são produzidos não apenas na elicitação, mas também durante todas as fases subseqüentes, a introdução desta propriedade aos modelos dos requisitos agrega flexibilidade ao desenvolvimento do software e a manutenções futuras.

Kotonya e Sommerville (1998) afirma que os requisitos são analisados em detalhes por diferentes *stakeholders* que devem negociar para decidir os

[1] Stakeholder é utilizado para se referir a qualquer pessoa que terá alguma influência direta ou indireta sobre os requisitos.

requisitos que devem ser aceitos. Este processo é necessário porque conflitos são inevitáveis entre os requisitos de diferentes fontes, sendo encontradas na maioria das vezes, informações incompletas. O requisito expressado pode também ser incompatível com o orçamento avaliado para desenvolver o sistema.

Segundo Tabeling e Grone (2005), a elicitação é um processo muito heterogêneo, isso porque as pessoas envolvidas em uma elicitação têm conhecimento diferenciado de como acontecem os fatos, torna uma tarefa difícil a abstração das informações de forma correta. Tabeling e Grõne (2005) afirmam também que esse processo jamais será automatizado, pois a interação humana é necessária para obter as informações mais precisa.

2.4 – ESPECIFICAÇÃO

A fase de especificação tem como objetivo descrever conceitualmente o conhecimento elicitado em modelos que expressem o domínio do problema, conforme o interesse dos *stakeholders*, e avaliar a descrição visando à correção, completude e a consistência das necessidades.

Segundo Goguen e Linde (1997), especificação de requisitos de software é o desenvolvimento de um documento que seja claro e preciso registrando cada requisito do sistema de software. A clareza da documentação deve descrever detalhadamente todas as observações do comportamento e característica esperada de um software.

Geralmente, a qualidade ERS (Especificação de Requisitos de Software) é o que contribui para um custo-benefício na criação de software que soluciona as reais necessidades do usuário.

Uma especificação de qualidade ERS deve retratar os seguintes atributos da qualidade:

1. Não ambíguo – Não apresentar mais de uma interpretação;

2. Completo – Apresenta todas as possibilidades;
3. Correto – Isento de erros;
4. Compreensível – De fácil interpretação;
5. Verificável – De fácil verificação;
6. Internamente consistente;
7. Externamente consistente;
8. Conciso – não redundante;
9. Independente da Implementação;
10. Rastreável - Localizável;
11. Modificável – que pode ser modificado;
12. Armazenamento eletrônico;
13. Executável/Interpretável;
14. Anotações relativamente importantes;
15. Anotações relativamente estabilizadas;
16. Anotações de versão;
17. Níveis de detalhamento;
18. Preciso – que apresenta exatidão;
19. Re-Utilizável – que possa ser utilizado em vários locais;
20. Organizado;
21. Referências cruzadas.

Os métodos formais possuem várias ferramentas que auxiliam na construção de uma especificação mais detalhada e precisa, diminuindo o valor agregado de desenvolvimento e melhorando a qualidade dos softwares de forma substancial.

Segundo Castro (1995), uma especificação nada mais é do que um modelo de um sistema em um determinado nível de abstração, de forma que seja possível descrever não só o comportamento do sistema, mas também os tipos de dados, caso seja necessário.

A especificação formal de um problema, está profundamente relacionada com o conceito de abstração. Uma especificação deve explicitar, de forma exata e não ambígua, todas as condições possíveis do problema apresentado de forma satisfatória, expressando o essencial e omitindo o não-essencial, relativos às decisões futuras de implementação.

2.4.1 – Modelos de Especificação de Sistema

Segundo Sommerville (2000), os requisitos de um sistema são obtidos com os *stakeholder*, os quais devem ser escritos em linguagem natural, ou seja linguagem utilizada no dia-dia, de forma que pessoas que não sejam especializadas na área consigam compreender. O resultado obtido através da elicitação é transformado em representações gráficas, sendo que existem várias técnicas para representar as soluções para os problemas de um sistema a ser desenvolvido. Esses modelos de representação gráfica permitem uma maior compreensão do que as descrições detalhadas em linguagem natural dos requisitos, sem contar que é um importante elo entre o processo de análise e de projeto.

Esses modelos podem representar o sistema, a partir de diferentes perspectivas:

- *Perspectiva Externa*: mostra o contexto ou o ambiente do sistema.
- *Perspectiva Comportamental*: modela o comportamento do sistema.
- *Perspectiva Estrutural*: mostra a arquitetura do sistema ou a estrutura dos dados processados pelo sistema.

Outros modelos também podem ser encontrados para especificação de sistema, como o modelo de objetos, cenários, entre outros. Os funcionamentos desses modelos são apresentados nos tópicos a seguir.

2.4.2 – MODELO DE CONTEXTO

O Modelo de Contexto consiste em, obter um processo de análise de requisitos, que deve limitar o ambiente do sistema. O trabalho envolve os *stakeholders*, que têm como função distinguir o que é o sistema e qual é seu ambiente. Uma análise deve ser feita para poder avaliar o contexto da empresa, e assim colocar alguns limites na elaboração do sistema. Esse resultado, normalmente, é um modelo simples da arquitetura, sendo o primeiro passo nesse modelo. O exemplo da figura 2 mostra o contexto de um sistema para uma agência bancária.

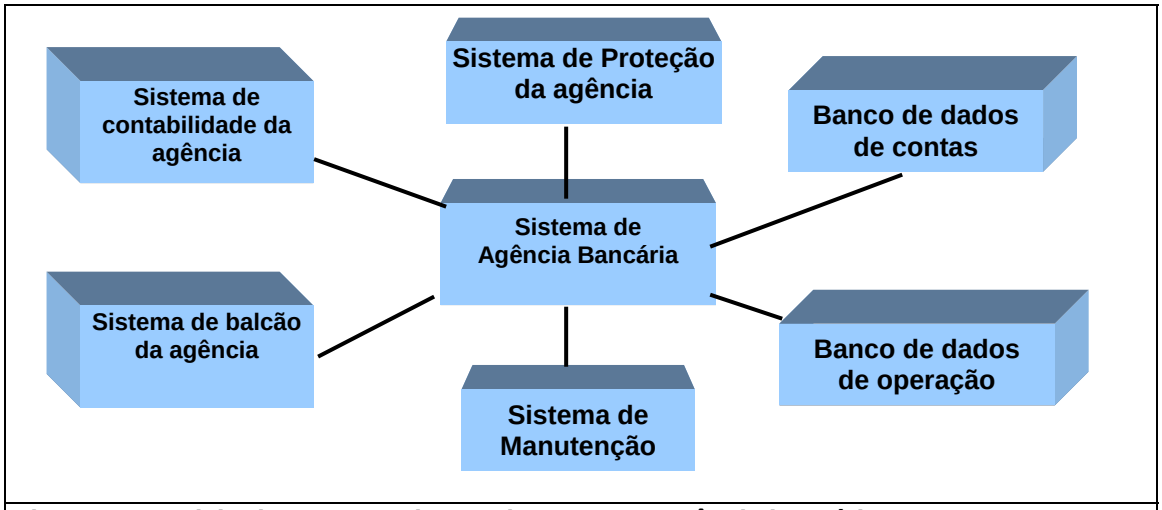

Figura 2 – Modelo de contexto de um sistema para agência bancária

Podemos observar que o contexto do sistema da agência bancária é composto de um banco de dados de contas, de um sistema de contabilidade da agência local, de um sistema de proteção e de um sistema de apoio a manutenção de máquinas, entre outros. Esse modelo mostra o contexto do sistema, mas não mostra a relação entre os contextos.

Portanto, os modelos de contexto são normalmente complementados por outros, por exemplo, os de processos, que descrevem as atividades de processos ligadas ao sistema.

2.4.3 – MODELO COMPORTAMENTAL

O modelo comportamental tem como objetivo retratar o comportamento geral do sistema, sendo que duas técnicas são mais utilizadas para os modelos de comportamento: o modelo de fluxo de dados e o modelo de estados.

O modelo de *fluxo de dados* retrata as etapas do fluxo em que os dados são transferidos e/ou processados dentro do sistema. Os dados são transformados a cada etapa, antes de seguirem para o próximo estágio.

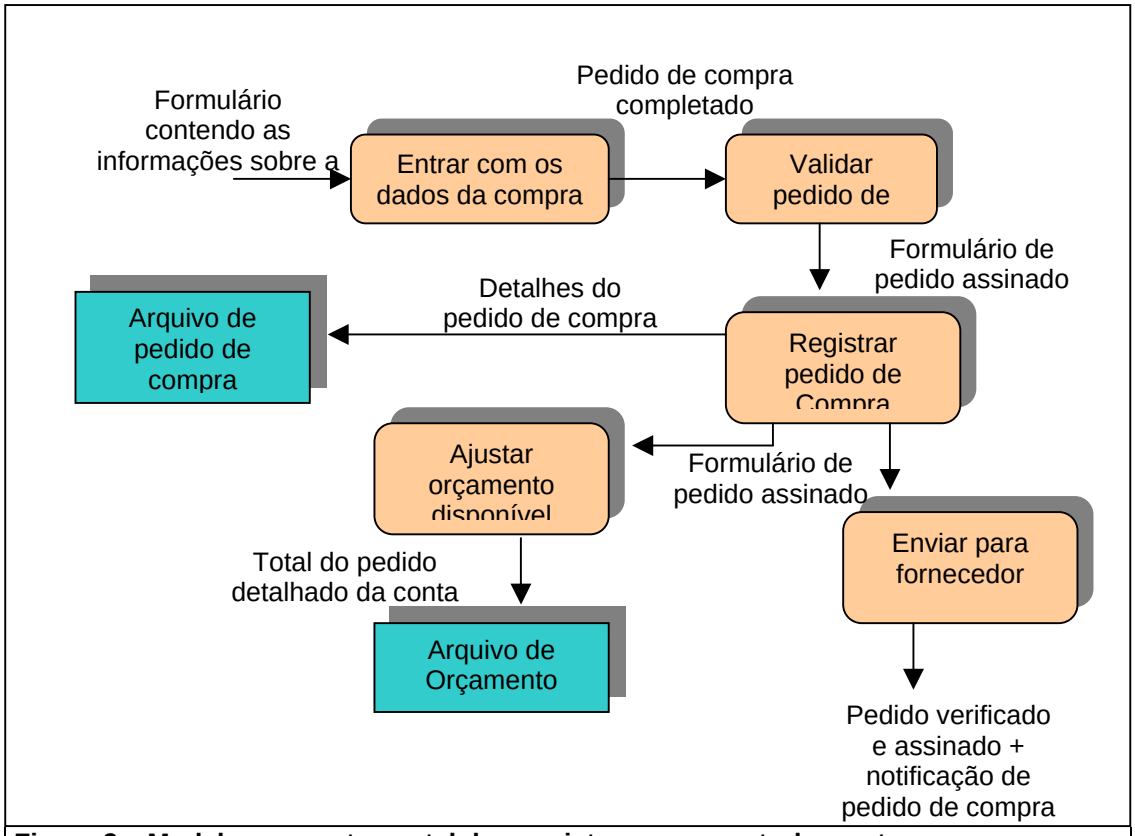

Figura 3 – Modelo comportamental de um sistema que controla o setor compras.

Como mostrado na figura 3, o modelo de fluxo de dados é retratado através de uma perspectiva funcional do sistema, em que cada transformação representa

uma única função. Esse modelo muitas vezes descreve o processamento dentro de um sistema, outras vezes o fluxo de dados de forma contextual, ou a troca de informações entre diferentes sistemas e subsistemas.

Já o modelo de estados modela o sistema a partir das respostas aos eventos externos e internos, sendo que os eventos são definidos com transições de um estado para outro, não mostrando o fluxo de dados dentro do sistema.

Dessa forma, esse tipo de modelo tem uma maior utilização na representação de sistema de tempo real, isso porque os sistemas em tempo real apresentam muitas alterações de estados, conforme o estimulo que recebem.

Um modelo de estados tem como filosofia demonstrar os estados possíveis dentro de um sistema, sendo que, quando o sistema recebe um estímulo, uma transição é realizada alterando o estado de um objeto.

Um exemplo é um sistema controlador de alarme em tempo real, outro exemplo é o funcionamento de um forno que pode ter mudanças de estado conforme os estímulos externos.

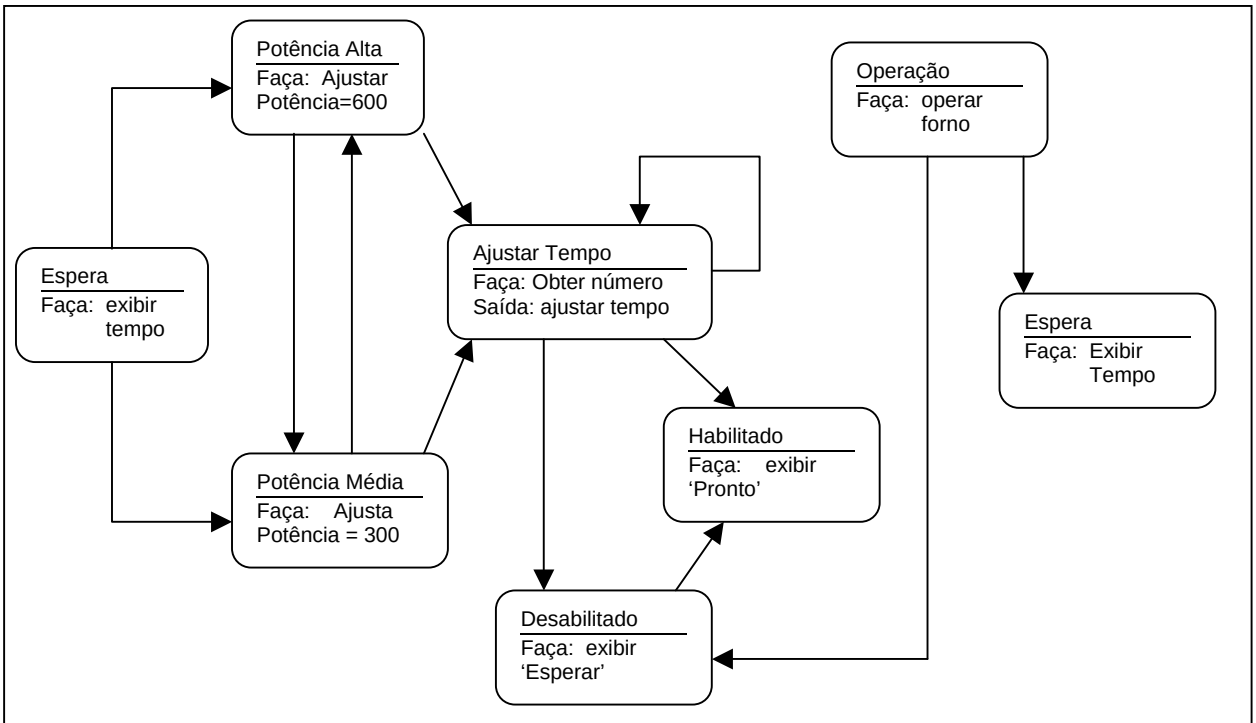

Figura 4 – Modelo de estado de um forno, Sommerville (2000).

No exemplo ilustrado na figura 4, são mostradas essas alterações de estados de um objeto forno. O estado de Espera fica aguardando a interação do usuário, o qual deve selecionar a potência do forno, sendo que, a potência será outro estado possível dentro do sistema. A potência pode ter dois estados: Média e Alta. Dependendo da potência selecionada o sistema irá solicitar um tempo de funcionamento e ficará aguardando esse tempo.

2.4.4 – MODELO DE DADOS

O modelo de dados tem como objetivo retratar a estrutura de armazenamento lógico e/ou físico de um sistema, sendo que, o modelo de dados apresenta um fator importante que é a modelagem de uma estrutura lógica dos dados processados pelo sistema.

Uma técnica muito utilizada para modelagem de dados é chamada de Modelo Entidade-Relacionamento (MER), que retrata as entidades do sistema e o relacionamento, mostra as ligações e interações dentro de um sistema. Essa técnica consiste também na definição de atributos para cada entidade.

Algumas importâncias de se utilizar o modelo de dados para especificação de um sistema:

- Através de um modelo de dados, fica mais fácil identificar as entidades, evitando erros comuns de nomes duplicados, algo que não é permitido em um modelo.
- Entender como as entidades se relacionam dentro do sistema.
- É utilizado como uma estrutura organizada para obter informações sobre o sistema.

A figura 5 mostra um exemplo de um modelo de dados de um sistema que gerencia as mensagens internas que os funcionários trocam dentro de uma empresa. Nesse exemplo são representadas as entidades que irão armazenas as informações dentro do sistema, como a entidade tab_funcionário que deve

armazenar todos os funcionários da empresa. A entidade tab_pasta irá armazenar as pastas de um determinado funcionário. A entidade tab_msg irá armazenar as mensagens que cada pasta deve ter. A entidade tab_perfil e tab_setor, iram classificar o funcionário dentro do sistema.

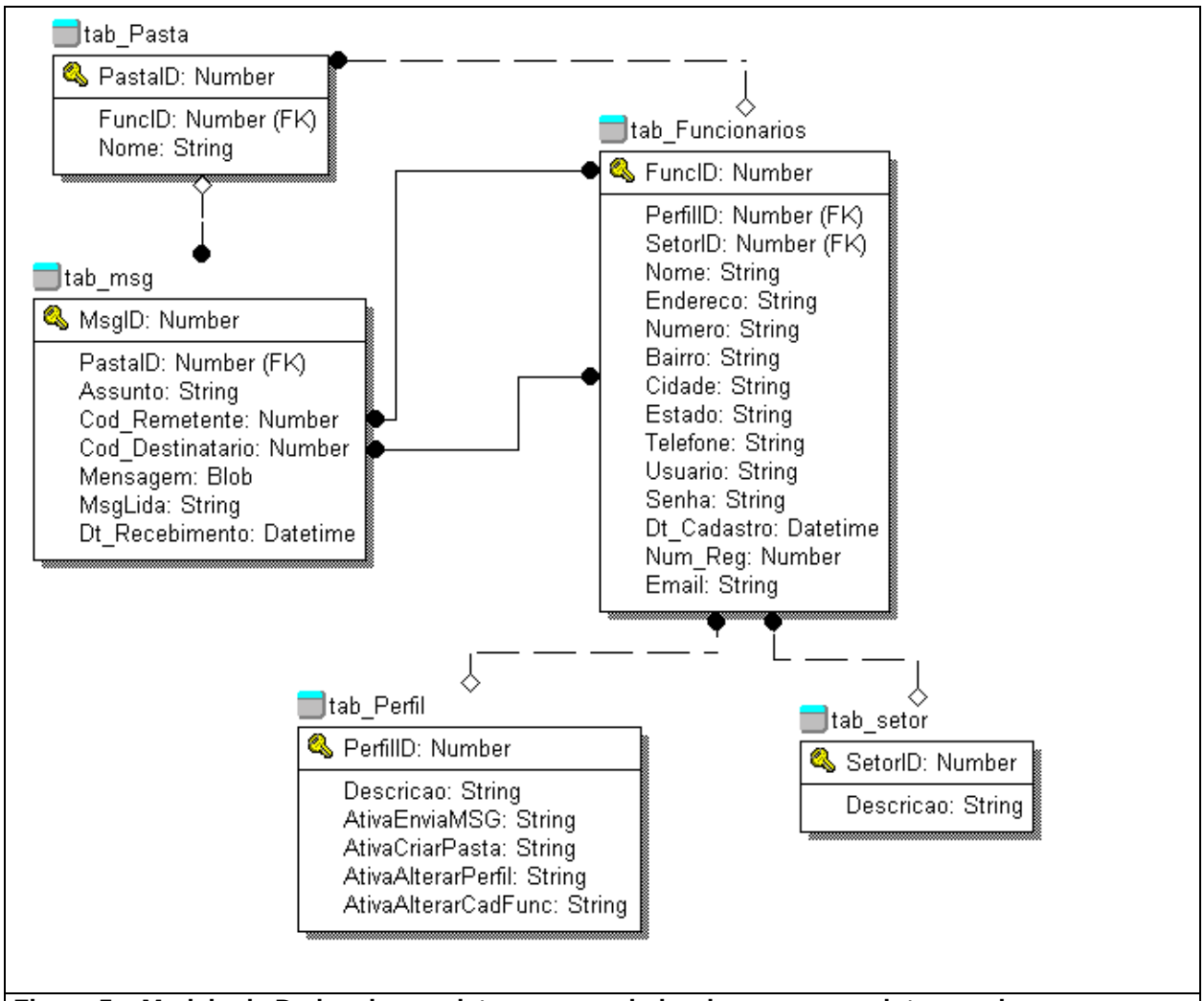

Figura 5 – Modelo de Dados de um sistema gerenciador de mensagens internas de uma empresa.

2.4.5 – MODELO DE OBJETOS

O modelo de objetos retrata o sistema com uma abordagem orientada a objetos. Atualmente esse tipo de modelo é o mais comum, por documentar sistema através de objetos que, na maioria das vezes, têm alguma relação com o mundo real. Assim, fica mais claro para os usuários que não têm experiência.

Esse modelo pode representar entidades e atributos, mas nesse caso chamamos de objetos e propriedades. Esse modelo possui uma estrutura que permite representar ações entre objetos que são chamados de eventos que são acionados de duas formas: através de uma intervenção do usuário ou através de uma alteração do ambiente.

O objetivo desse modelo é tentar especificar o sistema através de classes, com uma estrutura que represente uma situação do mundo real e que possibilite a manipulação pelo sistema.

Segundo Kotonya e Sommerville (1998), uma classe de objetos é uma abstração com relação a um conjunto de informações que identificam atributos comuns ou operações que são fornecidos pelos objetos.

Vários métodos de análise orientados a objetos foram propostos na década de 90, sendo Booch (1994), Yourdon (1990) e Rambaugh (1991). Esses modelos tinham muito em comum. Os principais desenvolvedores desses métodos decidiram unificar, surgindo assim a UML (Unified Modeling Language), que tornou-se o mais utilizado para a modelagem de objetos.

Um diagrama de classes da UML possui algumas estruturas muito importantes, sendo elas a Herança de objetos e a Agregação de objetos, entre outras. Na figura a seguir está representado um diagrama de classes de um sistema de biblioteca.

Na figura 6, é apresentado um exemplo de como seria um diagrama de classe em um sistema de biblioteca. Como podemos observar distribuímos as informações do sistema em classes e essas classes irão armazenar as características e as ações das mesmas. Na classe Aluno, podemos observar a possibilidade de armazenar características como, nome, endereço sexo entre outras e podemos realizar as seguintes ações como, inserir_alunos(), excluir_alunos(), Movimentar_livros() entre outras. Outro fato importante, são as interações que uma classe tem com a outra. Essas interações mostram os relacionamentos que as classes podem ter.

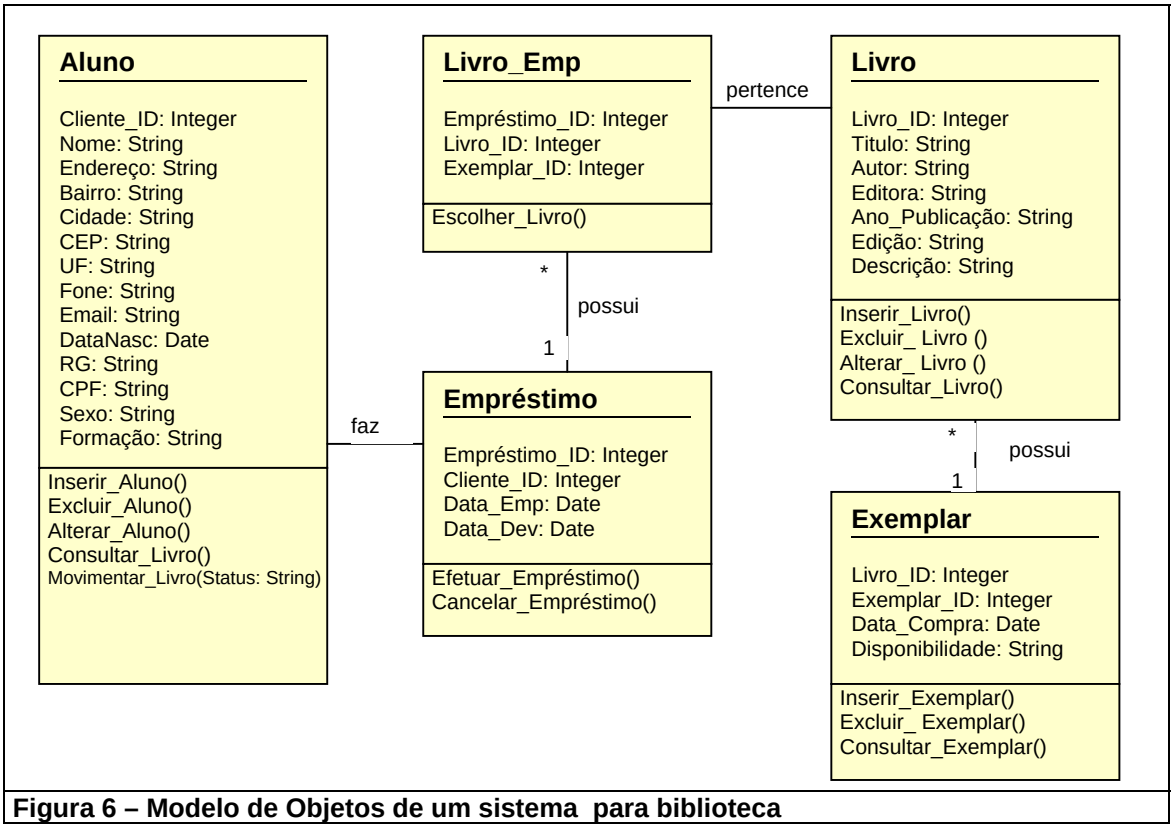

Figura 6 – Modelo de Objetos de um sistema para biblioteca

2.4.6 – MODELOS DE CASO DE USO

O modelo de casos de uso retrata as interações dos atores com o sistema, escrevendo um conjunto de ações e levando em conta as variações possíveis de serem realizadas pelo sistema.

Segundo Booch, Rumbaugh e Jacobson (2000), o modelo de caso de uso é uma descrição de um conjunto de seqüência de ações, inclusive variantes, que um sistema executa para produzir um resultado de valores que um ator pode observar.

Segundo Kotonya e Sommerville (1998), atores geralmente são pessoas que estão envolvidas em um processo. Normalmente atores são identificados como

papéis e, antes de modelar um processo, primeiramente identifica o papel que pode ser associado com o processo a ser modelado.

O exemplo da figura 7 mostra um modelo de caso de uso, no qual o ator passa por um caso de uso de identificação para acessar um sistema bancário.

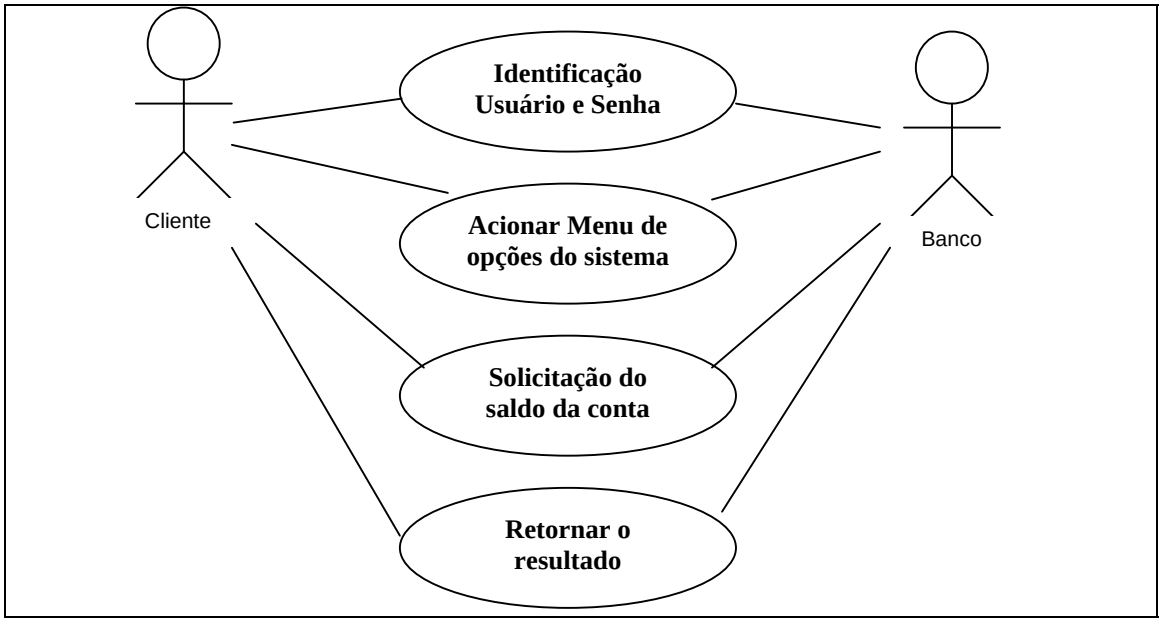

Figura 7 – Modelo de caso de uso que mostra a interação de um sistema bancário.

Na figura 7 os casos de uso representam as interações que um sistema bancário pode ter com o Cliente (Ator). Como podemos observar o Cliente realiza a sua identificação através de usuário e senha, o sistema bancário verifica e retorna um menu de opções para o Cliente. O Cliente seleciona uma opção, que na representação seria a opção de solicitar saldo. O sistema bancário retorna a seleção selecionada que na representação seria o saldo do cliente.

Segundo Lee e Tepfenhart (2001), caso de uso captura o requisito funcional e as informações relevantes, que irão direcionar a construção do sistema, com suas atividades associadas, que irão atingir esta especificação.

Para Alvares (2004) um caso de uso descreve interações em alto nível entre o sistema a ser desenvolvido e os elementos que interagem com ele. Caso de uso são descrições narrativas do processo de domínio concebendo a seqüência de

eventos dos atores que usam um sistema para executar um processo de domínio através de interações

De acordo com Paula (2001), caso de uso geralmente possui seu fluxo principal e vários sub-fluxos alternativos. Entretanto, notações especiais são utilizadas para facilitar as descrições mais complexas. Entre estas notações destacam-se os casos de uso primários através dos mecanismos de inclusão e extensão (<<Include>> e <<Extend>>).

Um caso de uso é uma pequena história que delineia uma seqüência esperada de solicitações e respostas entre um usuário e o sistema. É utilizado para comunicação entre um usuário específico (um ator), no contexto do sistema para atingir um objetivo proveitoso.

Segundo Pagliosa (2000), modelo de casos de uso é uma técnica de modelagem usada para descrever o que um novo sistema deve fazer ou o que um sistema existente já faz. Modelos de casos de uso são construídos através de um processo interativo entre cliente e desenvolvedores do sistema, obtendo assim uma especificação do que o sistema deve fazer, sendo que esse processo é descrito na UML em diagramas de casos de uso. Um diagrama de caso de uso contém elementos gráficos que representam o sistema, como é apresentado na tabela 1.

Tabela 1 – Diagrama de Casos de Uso

	*Caso de Uso* → Utilizado por um ou mais atores
	*Atores* → Agentes externos ao sistema, podendo ser um usuário ou um sistema.
	Relacionamento → Relaciona atores com Casos de uso.
	*Fronteira do Sistema* → Limita o que faz parte e o que não faz parte do sistema.

Algumas características de um caso de uso são:

- Um caso de uso sempre é inicializado por um ator, seja direta ou indiretamente.

- Um caso de uso oferece um resultado para um ator, não necessariamente distinto, mas apreciável.

- Um caso de uso é completo. Um caso de uso deve ser uma descrição completa. Um caso de uso não é completo enquanto não fornece um resultado para um ator.

Casos de uso são conectados a atores através de associações, as quais são algumas vezes chamadas de associações de comunicação. As associações mostram com quais atores o caso de uso se comunica, incluindo o ator que inicializa a execução do caso de uso.

O processo de determinação dos casos de uso começa com a definição dos atores, sendo que existem algumas perguntas chave que se deve fazer para identificar os atores, tais como:

- Quais funções do sistema o ator necessita ? O que o ator necessita fazer ?

- O ator necessita criar, destruir, modificar, armazenar ou recuperar informações do sistema ?

- O ator tem que ser notificado sobre eventos ocorridos no sistema, ou o ator necessita notificar o sistema sobre alguma coisa ? O que esses eventos representam em termos de funcionalidade ?

- O trabalho diário do ator poderia ser simplificado ou tornado mais eficiente com novas funções no sistema ?

2.4.7 – CENÁRIOS

Segundo Lee e Tepfenhart (2001), cenário é uma instância de um caso de uso, que mostra uma série particular de interações entre objetos em uma execução exclusiva do sistema. Essa execução individual do sistema tipicamente constitui uma transação entre o objeto externo e o sistema.

Os cenários podem ser ilustrados de formas diferentes:

1. *Diagrama de seqüência*: mostra a interação entre um conjunto de objetos em ordem temporal, facilitando o entendimento relacionado às questões de tempo. Uma forma alternativa é um diálogo sob a forma de texto;

2. *Diagrama de colaboração*: mostra as interações entre um conjunto de objetos inter-relacionados em um gráfico que facilita o entendimento da estrutura do software, pois todo tipo de interação que afeta um objeto é localizado em torno dele.

Um diagrama de seqüência tem como elementos básicos atores e o sistema envolvido. Seu objetivo é mostrar a ordem que as mensagens são trocadas entre os atores e o sistema; sua estrutura temporal de leitura é de cima para baixo.

O exemplo da figura 8 mostra as interações realizadas em uma agência bancária para criar uma nova conta para o cliente. A seqüência do sistema seria a agência bancária solicita a abertura da conta do cliente. O sistema bancário solicita as informações do cliente. A agência fornece os dados do cliente. O sistema bancário solicita que tipo de conta será aberta (conta simples ou especial), a agência informa o tipo de conta. O sistema bancário solicita um saldo inicial e a agência informa qual será o depósito inicial da conta. Depois de todos esse passos o sistema bancário cria a nova conta e retorna para agência a confirmação da abertura da conta.

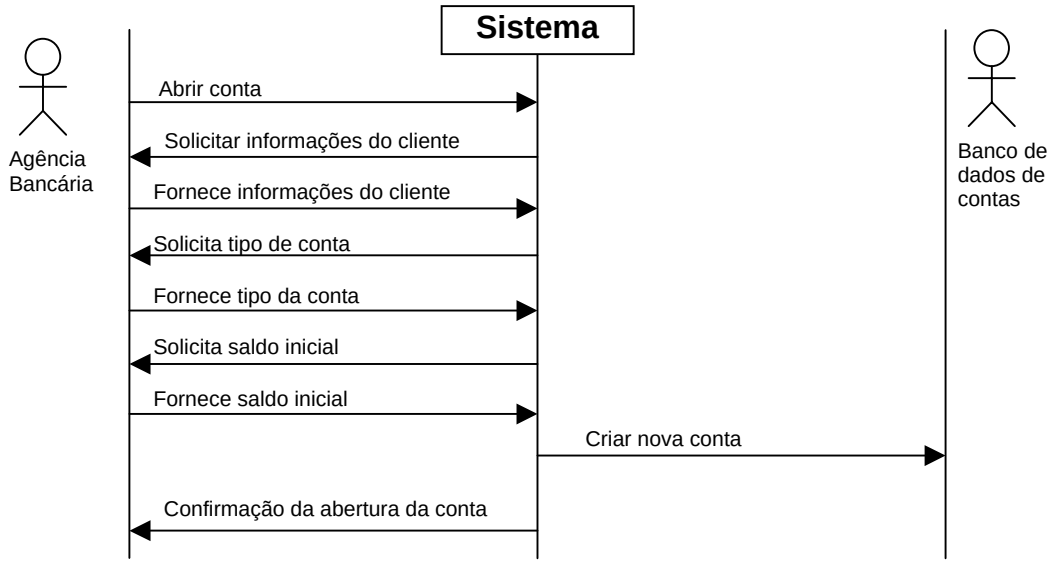

Figura 8 – Representação de um diagrama de seqüência.

Um diagrama de colaboração mostra o fluxo de controle, enfatizando os relacionamentos estruturais entre as instâncias, bem como as mensagens transmitidas entre elas. Esse diagrama mostra os objetos e seu vínculo existente exatamente antes do serviço iniciar, ou mesmo durante a sua realização.

Outra característica desse diagrama é que ele é muito semelhante ao diagrama de seqüência. Entretanto, ele retrata informações adicionais sobre como os objetos estão mutuamente relacionados. O exemplo da figura 9, mostra esse inter-relacionamento existente entre os objetos.

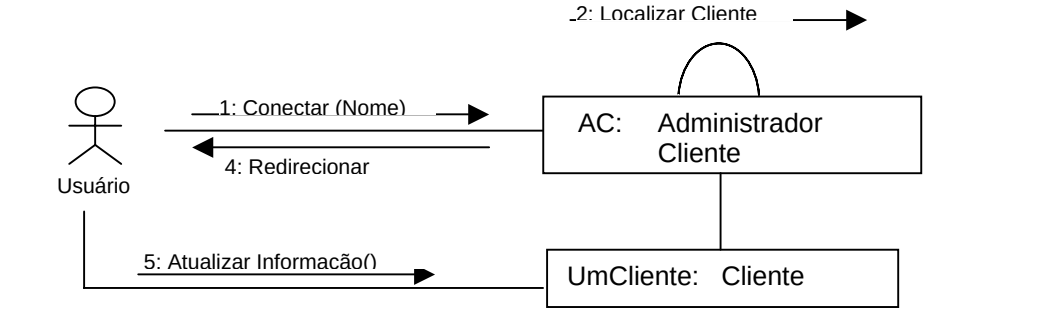

Figura 9 – Modelo de um diagrama de colaboração

Segundo Jones (2000), diagrama de seqüência é um tipo de diagrama de interações entre objetos que enfatiza mais a seqüência temporal do que o relacionamento estático do objeto. Os objetos interagem por meio de mensagens, sendo representado como “caixas”, conforme os padrões da UML.

Breitman e Leite (1998) define cenários como sendo um conjunto de figuras ou representações em movimento o qual é utilizado em diferentes disciplinas como Estratégia Militar, Marketing, Economia, etc. Já na área de SI (Sistema de Informação), cenários começaram a ser usados para as interfaces Homem/Computador, na Engenharia de Software. O cenário fornece uma visão da situação da tarefa juntando uma maneira eficaz de comunicação entre os atores envolvidos no assunto estudado.

Para a construção de cenários, primeiramente, define-se o problema central, o qual deve representar cenários com algumas funções principais dos sistemas. O segundo passo é desmembrar o primeiro cenário em vários temas, demonstrando, dessa forma, o comportamento que o software deve apresentar.

2.5 – VALIDAÇÃO E VERIFICAÇÃO DE REQUISITOS

De acordo com Rocco (2002), a atividade de verificação e validação, no contexto da engenharia de requisitos, consiste na execução de ações com o objetivo de confirmar o conhecimento adquirido. Procura-se fortalecer que o quê foi especificado realmente represente o domínio do problema, isto é, retrate as necessidades. Sob esse ponto de vista, a verificação e validação pode ser considerada como uma atividade de negociação. Os clientes procuram explorar o completo entendimento dos requisitos, de acordo com o que eles querem e necessitam, enquanto que os desenvolvedores procuram a confiança em resolver o problema certo. Também busca assegurar-se que os requisitos funcionais sejam verificáveis, almejando garantir a correção da especificação, e que os requisitos

não funcionais de qualidade, quantidade e restrições de software estejam contemplados na definição.

De acordo com Sommerville (2000), validação e verificação são duas atividades que devem estar presentes nessa fase, pois para ele a definição desses dois termos é:

- *Validação*: estamos construindo o produto certo?
- *Verificação*: estamos construindo certo o produto?

Segundo Goguen e Linde (1997), a fase de verificação e validação tem como objetivo assegurar que o processo de especificação dos requisitos de software seja coerente com os requisitos do sistema, consolidando uma documentação padrão na fase de requisitos.

Isso mostra que os dois termos, apesar de serem similares, nesse contexto, mostram que verificação e a validação são processos que checam se o software realiza realmente o que foi especificado. Validação e a verificação são processos muito amplos, pois é necessário assegurar que o software atende às expectativas junto ao cliente, ou seja, além de checar a conformidade da especificação, temos de mostrar que o software faz o que o cliente espera que faça.

Segundo Wallace e Ippolito (1997), verificação e validação de software (V&V) são formas para determinar a implementação correta, com a possibilidade de rastrear os requisitos do sistema.

O objetivo maior do processo V&V de software é a análise compreensiva e o teste de software durante o desenvolvimento para determinar qual é a performance do software, determinando corretamente as funções e a obtenção de informações sobre a qualidade e a confiança.

O conceito sobre métodos formais é utilizado meramente para verificação se o sistema está correto. Muitos trabalhos, especialmente em instituições acadêmicas, têm concentrado o uso de métodos formais para essa verificação. Fazer verificação de sistema utilizando método formal é uma técnica muito difícil,

isso porque o método formal é percebidamente impopular, sendo na maioria das vezes confusa para provar que o programa está correto Saiedian (1997). Entretanto, verificação de sistema é somente um aspecto do método formal e talvez tenha um significado menor. Verificação e validação de sistema normalmente é aplicada por uma pequena fase no desenvolvimento, ou seja, depois que o sistema tenha sido modelado e especificado. O uso primário do método formal é para escrever a especificação que precisamente define o que o sistema pretende fazer, de acordo com Saiedian (1997).

2.6 – GERENCIAMENTO DE REQUISITOS

Novos requisitos surgem em qualquer etapa no processo de desenvolvimento de software, resultando assim em alterações. Para diminuir essa dificuldade, é necessário gerenciar os requisitos para que a documentação seja coerente em todo o processo.

O gerenciamento de requisitos é um processo que tem um relacionamento em paralelo com as demais fases citadas acima, com o objetivo de gerenciar as ocorrências alteradas nos requisitos do sistema sendo essas alterações por motivos diversos, como por exemplo, mudança no ambiente do sistema, ou uma compreensão melhor, por parte do usuário, sobre sua real necessidade, entre outras.

O gerenciamento de requisitos tem como direcionamento algumas questões que devem ser levadas em conta, como:

- Gerenciamento de mudanças de requisitos já resolvido anteriormente
- Gerenciar os relacionamentos entre requisitos
- Gerenciar as dependências entre os documentos de requisitos e os produzidos durante o processo de engenharia de software.

Requisitos não são gerenciados efetivamente sem serem rastreados. Um processo de rastreamento pode mostrar o relacionamento que o requisito informado tem com o sistema, sendo que essa informação é usada para encontrar outros requisitos que podem ser afetados ou até mesmo alterados.

Segundo Pinheiro (2000), o processo de realizar rastreamento de um software tem como objetivo definir, capturar e acompanhar os rastros deixados pelos requisitos em todas as etapas de desenvolvimento de software. O processo de rastreabilidade pode apresentar um grau de dificuldade dependendo do volume de informações gerado pelos requisitos, sendo necessário catalogar e associar as informações dos processos com seus respectivos requisitos.

Assim, uma boa prática de gerenciamento de requisitos busca manter dependências entre os requisitos, tendo benefícios de longo prazo e obtendo assim uma satisfação maior do cliente e um custo menor de desenvolvimento do software. Este retorno não é imediato. Assim, o gerenciamento de requisitos às vezes pode ser visto como não necessário e sem valor.

Segundo Goguen e Linde (1997), a fase do gerenciamento de requisitos de software é o planejamento e controle dos requisitos elicitados, especificados, analisados e verificados ativamente.

CAPÍTULO 3 – OBJECT CONSTRAINT LANGUAGE

A estruturação inicial da linguagem OCL foi desenvolvida na IBM por volta de 1995 por Jos Warmer e Steve Cook, com o propósito de ter uma linguagem de modelagem de negócio, sendo que essa nova linguagem foi incorporada como parte do padrão UML (Unified Modeling Language), segundo Warmer e Kleppe (1999), pela OMG (Object Management Group). Os usuários de UML e outras linguagens de modelagem podem utilizar OCL, com o objetivo de especificar restrições através de expressões vinculadas ao modelo.

OCL como definição (OMG, 2003) é uma linguagem de expressões textuais e precisas, utilizada na descrição de restrições em modelos orientados a objeto, como forma de complementar a parte gráfica dos modelos para descrever restrições, que não conseguem ser diagramaticamente representadas.

Esta linguagem foi fortemente influenciada pelo método Syntropy – linguagem de modelagem orientada a objeto que combina OMT (Object Modeling Technique) com a formalidade de um subconjunto da linguagem formal Z. Dessa forma, por essa influência de Z, classifica-se a OCL como uma linguagem formal baseada em modelos, muito embora adote uma sintaxe simples, não diagramática, e que utiliza símbolos matemáticos que compõem a teoria de conjuntos e lógica, porém de fácil compreensão para quem não possui conhecimento matemático aprofundado.

Um dos mais importantes aspectos de OCL é que se tornou parte do padrão UML, segundo Booch, Rumbaugh e Jacobson (2000), para especificação. O seu objetivo foi o de acrescentar à definição de UML a possibilidade de especificar restrições aos modelos.

Embora vários métodos de modelagem, utilizem o conceito de restrições, esta é uma idéia relativamente nova. Por restrições pode-se entender limitações em um ou mais valores de parte ou do todo de um objeto, em um modelo orientado a objeto ou em um sistema. Na prática são detalhes e informações extras, que não

conseguem ser expressos pelos modelos gráficos. Antes de OCL, estas restrições eram descritas em linguagem natural, resultando em ambigüidades e imprecisões.

A utilização da idéia de restrições acrescenta algumas vantagens:

- Melhorar a Documentação

> por acrescentar aos modelos informações sobre os elementos e seus relacionamentos. Um modelo gráfico pode conter algumas restrições, como é o caso da multiplicidade nos relacionamentos; no entanto, alguns detalhes não são possíveis de serem representados como, por exemplo, validar um determinado campo através de uma ligação entre duas classes;

- Aumentar precisão da informação

> as restrições não podem ser interpretadas de forma diferente por várias pessoas, tornando assim o modelo ou sistema não ambíguo, obtendo uma maior precisão;

- Possibilitar Comunicação sem Enganos

> em função da existência de precisão na descrição das informações, os desenvolvedores passam a ser capazes de comunicar suas intenções de forma não ambígua, solucionando mais cedo, no processo de desenvolvimento, possíveis defeitos, diminuindo custos e evitando-se frustrações.

A OCL apresenta também recursos para representar restrição através de navegação, ou seja, através de relacionamento entre os objetos. Desta forma, as expressões OCL permitem escrever restrições no comportamento dos objetos identificados, navegando-se entre estes objetos. Na fase da especificação, as expressões aparecem nas invariantes, pré e pós-condições.

3.1 – PRINCIPAIS CONCEITOS QUE ENVOLVEM OCL

A linguagem OCL (OMG, 2003) é dividida em quatro conceitos principais, sendo eles: self, Invariantes, pré-condições e pós-condições. A definição desses conceitos é:

Self: Cada expressão OCL é escrita em um contexto de instância de um tipo específico. Em cada expressão OCL, a palavra reservada self é usada para referenciar o contexto da instância.

Invariantes: são restrições que estabelecem uma condição que sempre deve ser atendida, ou seja, deve ser verdadeira para todas as instâncias da classe, tipo ou interface. São descritas usando-se uma expressão que será avaliada como verdadeira, se a invariante é satisfeita durante todo o tempo.

Exemplo para validar um atributo em uma classe ALUNO, a qual define que a nota do mesmo deve ser especificada entre os valores 0 e 10:

```
context aluno inv:
        self.nota >= 0  and  self.nota <=10
```

Figura 10 – Exemplo de OCL, aplicando uma restrição simples.

Pré-Condições: especificam as condições obrigatórias que devem ser verdadeiras (satisfeitas) no momento anterior de sua execução, sendo que essa validação é associada a uma operação ou método.

Pós-Condições: especificam as condições obrigatórias que devem ser verdadeiras (satisfeitas) assim que a operação acabar de ser executada. Como na pré-condição, a pós-condição também associa essa execução a partir de uma operação ou método.

Os conceitos de pré e pós-condição são muito úteis porque, através deles, pode-se colocar uma infinidade de informações para se obter um resultado mais claro. A ilustração abaixo mostra um exemplo de como se pode definir um método que deve calcular a data de vencimento de uma determinada prestação.

```
Context Prestação::CalculaData(Dt: date): date
     pre: result = if Dt = Null
                   then date()
                   else Dt endif
    post: result = Dt + 30
```

Figura 11 – Exemplo de OCL, aplicando uma pós e pré-condição.

3.2 – VALORES E TIPOS DE DADOS

A linguagem OCL apresenta pré-definições de tipos de dados, sendo classificados como os tipos básicos e os tipos coleção, (OMG, 2003), sendo definidos em um modelo e considerados em um formato invariável para todo modelo. Isso significa que, quando se define que um objeto possui um atributo com um determinado tipo, isso permanecerá por todo o modelo. A tabela abaixo mostra de forma ilustrativa os tipos e valores básicos existente em OCL:

Tabela 2 – Tipos e Valores básicos em OCL.

Tipo	Valor
Boolean	TRUE / FALSE
Integer	1, 5, 8, 20, 43, -34...
Real	2.23, 10.32, -105.89...
String	“Linguagem OCL”, “Exemplo de String”, ...

A definição de cada tipo seria:

- *Boolean* → Armazena valores do tipo True/False (Verdadeiro/Falso)
- *Integer* → Armazena valores numéricos no formato inteiro, ou seja, não é possível armazenar valores decimais.

- *Real* → Armazena valores numéricos no formato inteiros e decimais.

- *String* → Armazena um conjunto de caractere. Um caractere é um valor alfanumérico, ou seja, letras [A..Z] e números [0..9] ou caracteres especiais.

Os tipos básicos podem utilizar operações para determinar o tipo de validação possível em OCL. Segue abaixo uma tabela ilustrativa contendo algumas das operações mais utilizadas:

Tabela 3 – Tipos e Operações básicas em OCL

Tipo	Operação
Boolean	AND, OR, XOR, NOT
Integer	*, /, +, -, ABS(), …
Real	*, /, +, -, FLOOR(),...
String	toUpper(), toLower(), ...

As operações descritas na tabela acima são uma pequena amostra das operações possíveis a partir de um determinado tipo (OMG, 2003).

Um outro tipo existente em OCL muito importante é o tipo coleção (OMG, 2003), que tem como característica armazenar coleção de valores do mesmo tipo. Em OCL é definido por três tipos de coleção que são ilustradas na tabela a seguir:

Tabela 4 – Tipos de coleção em OCL e exemplos de valores.

Tipo	Valor	Onde
Set	Set{1,2,7,50} Set{‘Maça’, ‘Laranja’ }	Coleção que contém elementos sem duplicação
Sequence	Sequence { 1, 3, 45, 2, 3 } Sequence { 'ape', 'nut' }	Coleção de elementos duplicados, mas ordenados.
Bag	Bag{1,2,45,2,7,4}	Coleção de elementos duplicados, mas sem ordenação.

Como coleção é um tipo especial em OCL, esse tipo apresenta operações específicas para obter valores desse tipo, como é mostrado na tabela abaixo:

Tabela 5 – Operações do tipo coleção em OCL.

Operações relacionadas aos tipos definidos com coleção	
Size	O Número de elementos na coleção
Count(objeto)	O Número de ocorrência do objeto na coleção
Include(objeto)	Verdadeiro se o objeto é um elemento da coleção
includesAll(coleção)	Verdadeiro se todos os elementos da coleção estiverem presentes na coleção
IsEmpty	Verdadeiro se a coleção não contém elementos
NotEmpty	Verdadeiro se a coleção contém um ou mais elementos.
Iterate(expressão)	Expressão é avaliada para todos os elementos na coleção. O tipo do resultado depende da expressão.
Sum()	A adição de todos os elementos na coleção. Os elementos devem ser do tipo que admitem adição
Exists(expressão)	Verdadeiro se a expressão é verdadeira para pelo menos um elemento na coleção
ForAll(expressão)	Verdadeiro se para todos os elementos a expressão for verdadeira.
Select (expressão)	Se verdadeiro seleciona todos os elementos da expressão.

O tipo coleção é muito importante e essencial para a linguagem OCL, pois, a partir dessa estrutura, consegue-se obter muitos valores que ficam difíceis de identificar somente em um modelo diagramaticamente representado. Para ilustrar melhor esse tipo, a figura abaixo mostra alguns exemplos de como utilizar o tipo coleção.

```
context Empresa inv:
        self.empregado -> Select( Idade > 50 )
```

Figura 12 – Aplicando operador SELECT para obter um conjunto de valores.

No exemplo acima, a expressão OCL representa que o atributo empregado da classe Empresa irá receber todos os empregados que tiverem mais que 50 anos. Já o exemplo abaixo mostra a utilização de uma outra operação “ForAll()”,que tenta obter para todos os objetos o valor “Silva”.

```
context Empresa inv:
        self.nome_empregado -> forall( sobrenome = ‘Silva’ )
```

Figura 13 – Aplicando operador ForAll para obter um conjunto de valores.

3.3 – CONSIDERAÇÕES

A linguagem OCL, como é mostrado acima, é uma linguagem de fácil assimilação, sendo possível utilizá-la em qualquer tipo de documentação com a finalidade de obter restrições na especificação.

Sua principal característica é ser uma linguagem formal usada para expressar restrições. Tipicamente ela especifica condições invariantes que devem assegurar que o sistema foi modelado. Note que as expressões OCL avaliadas não têm nenhum efeito colateral, isto é, não podem alterar o estado de execução do sistema correspondente. Além disso, quando se especifica invariantes em um modelo de classe, por exemplo, objetiva se a utilização de uma estrutura de restrição em uma aplicação.

CAPÍTULO 4 – ESTUDO DE CASO

Para o estudo de caso foi elaborada uma especificação de um software, usando todos os conceitos de Engenharia de Requisito, mencionado nesse trabalho. Foram geradas duas especificações do mesmo software, sendo que em uma delas foram aplicados os conceitos de OCL, e na outra, não foram aplicados tais conceitos.

A especificação gerada no estudo de caso foi denominada Documento de Especificação de Requisitos do Módulo de Cálculo de IPVA do Sistema de Despachante, que é mostrado nos itens abaixo.

4.1 – APRESENTAÇÃO DA PESQUISA

Essa pesquisa tem como objetivo mostrar uma especificação de software a qual tenta melhorar o entendimento através de algumas técnicas.

Uma especificação de software é na verdade uma documentação que explica o quê o software irá fazer, conforme as exigências do usuário, sendo que, para realizá-la, pode-se utilizar diversas técnicas da Engenharia de Requisitos.

Nesse contexto, a primeira etapa da elaboração do documento de especificação de requisitos foi a elicitação dos mesmos, o quê mostra a real necessidade do usuário. A partir dessa elicitação, foi possível gerar uma lista de requisitos do sistema.

A segunda etapa foi gerar a documentação do software, utilizando os padrões propostos pela UML. Ela é composta por um diagrama de classe, um diagrama de casos de uso e um diagrama de atividade. Esses diagramas foram escolhidos pois iram representar com maior clareza as informações obtidas na elicitação, isso porque o diagrama de classe organizará as entidades que o sistema deve ter contendo suas características e ações que os mesmo poderão realizar. O diagrama de casos de uso irá representar as interações que os atores poderão

realizar com os casos de uso e o diagrama de atividade especifica com mais detalhes as atividades que um caso de uso deve fazer.

A terceira etapa foi escrever uma especificação formal, utilizando a linguagem de restrição chamada OCL (Object Constrants Language – Linguagem de Restriação de Objetos).

Na quarta etapa foi a criação de um questionário, para que pessoas da área de desenvolvimento de software pudessem opinar sobre a técnica de especificação utilizando OCL. A realização do questionário foi feita de duas formas: através de envio de e-mails ou de cópia impressa entregue pessoalmente.

Foram selecionadas 20 pessoas, das quais 14 responderam ao questionário. Os selecionados deveriam apresentar obrigatoriamente um perfil de desenvolvedor, algum conhecimento em UML e instrução mínima de graduado na área de informática. Os entrevistados escolhidos são, professores universitários e profissionais que desenvolvem software em grandes empresas.

O critério de avaliação obedeceu à seguinte forma: metade dos entrevistados recebeu a documentação contendo a especificação com OCL e a outra metade recebeu a documentação sem a especificação OCL. Utilizando essa estratégia, tentou-se não induzir o entrevistado a opinar em favor da especificação OCL, pois o intuito foi o de avaliar qual a melhoria obtida com a utilização da OCL.

4.2 – CONTEXTUALIZAÇÃO DO ESTUDO DE CASO

O contexto deste estudo de caso foi trabalhar uma parte específica (um módulo) de um sistema aplicativo para despachante, o qual controla o processo de cálculo de IPVA (Imposto sobre a Propriedade de Veículos Automotores). O funcionamento básico desse módulo exige o conhecimento dos dados do veículo, para que, a partir dessas informações, se possa realizar o cálculo do imposto para qualquer veículo.

4.3 – Etapa de Desenvolvimento do Documento de Especificação de Requisitos

A primeiro atividade se constituiu da elicitação dos requisitos, e posteriormente da criação de uma lista dos mesmos que mostra a funcionalidades do módulo de cálculo de IPVA do software.

4.3.1 – Elicitação

Na etapa de elicitação foi feito através de entrevistas junto aos usuários do sistema, tendo como objetivo mostrar as etapas necessárias para realizar o cálculo de IPVA para todos os tipos de veículos.

– Cálculo de IPVA de Veículos Usados

O cálculo de IPVA é feito para todos os veículos automotores, incluindo carros, motos e caminhões. Para realizar o cálculo, são necessárias as seguintes informações: marca do fabricante do veículo, modelo do veículo, ano de fabricação, tipo de combustível e data do dia que será realizado o cálculo. Todas essas informações são necessárias e obrigatórias para se calcular o valor do imposto.

Para realizar o cálculo primeiramente é necessário um instrumento chamado *tabela de valores venais*. Essa tabela contém todos os fabricantes de veículos automotores, como também todos os seus modelos fabricados, nacionais e importados e, para cada modelo descrito, são atribuídos valores médios de mercado referentes ao veículo, conforme seu ano de fabricação e seu tipo de combustível.

Depois de obter o valor venal do veículo, o segundo passo é classificar o veículo conforme tabela de categoria com os seguintes itens: moto, utilitário, misto, caminhão e carro. Para cada categoria, o valor do imposto é feito de forma diferente.

– INFORMAÇÕES NECESSÁRIAS PARA CALCULAR O IPVA DE QUALQUER VEÍCULO

Como regra geral, para se conseguir calcular o valor de IPVA de um veículo, é necessário ter em mãos a marca do fabricante, o modelo do veículo, o ano de fabricação, seu tipo de combustível, a data atual e o ano de cálculo de IPVA.

– CÁLCULO DE IPVA PARA CATEGORIA MOTO.

Para se calcular o IPVA para a categoria MOTO é necessário a cilindrada da moto, que é descrita na *tabela de valores venais,* independente do tipo da moto. Para se obter o valor do IPVA, basta aplicar a seguinte base de cálculo:

IPVA = ValorVenal * 2%

Uma observação importante é que se a moto descrita não se enquadrar em nenhuma classificação da *tabela de valores venais*, ela será classificada com base na sua cilindrada e atribuída a ela um valor venal base .

– CÁLCULO DE IPVA PARA CATEGORIA UTILITÁRIO E MISTO

Primeiramente deve-se definir o que é um veículo utilitário e um veículo misto. Como regra geral, um utilitário é um veículo de passeio que leva uma pequena quantidade de carga, com um limite médio de 500 KG. Já o considerado misto caracteriza-se por utilizado tanto como veículo de passeio veículo de carga. Nesse tipo de categoria enquadram-se os furgões, as Vans, etc.

Para calcular o IPVA de um veículo que apresente a categoria “Utilitário” ou “Misto”, obtém-se o valor venal do veículo e aplica-se a seguinte base de cálculo:

IPVA = ValorVenal * 2%

Observação importante nesse tipo de categoria é que se não for encontrado nenhum valor do veículo na *tabela de valores venais*, ele será classificado em uma categoria denominada ***"demais modelos do fabricante"***.

– CÁLCULO DE IPVA PARA CATEGORIA CAMINHÃO

Para calcular o IPVA de um veículo que apresenta a categoria do tipo caminhão, é necessário obter o valor venal, na *tabela de valores venais*, e aplicar a seguinte base de cálculo:

IPVA = ValorVenal * 1.5%

Observação importante nesse tipo de categoria é que se não for encontrado nenhum valor do veículo na *tabela de valores venais*, ele será classificado em uma categoria denominada ***"demais modelos do fabricante"***.

– CÁLCULO DE IPVA PARA CATEGORIA CARRO

A categoria "Carro" apresenta uma classificação especial, se a compararmos com as demais categorias, pois o valor do IPVA desse tipo de categoria apresenta-se variável, dependendo do combustível.

Para se iniciar o cálculo, primeiramente é necessário obter o valor venal do veículo na *tabela de valores venais*. Depois de se obter esse valor, deve-se fazer a classificação do veículo baseado em seu combustível e aplicar a base de cálculo que é descrita no tópico abaixo.

– CATEGORIA CARRO COM O COMBUSTÍVEL A GASOLINA

Nesse caso é aplicada a seguinte base de cálculo:

IPVA = ValorVenal * 4%

– CATEGORIA CARRO COM O COMBUSTÍVEL A ÁLCOOL

Nesse caso é aplicada a seguinte base de cálculo:

IPVA = ValorVenal * 3%

– CATEGORIA CARRO COM O COMBUSTÍVEL A DIEESEL

Nesse caso é aplicada a seguinte base de cálculo:

IPVA = ValorVenal * 4%

– CATEGORIA CARRO COM O COMBUSTÍVEL A GÁS (GNV)

Nesse caso é aplicada a seguinte base de cálculo:

IPVA = ValorVenal * 4%

Observação importante nesse tipo de categoria é que se não for encontrado nenhum valor do veículo na *tabela de valores venais*, ele será classificado em uma categoria denominada ***"demais modelos do fabricante"***.

– CÁLCULO DE IPVA DE VEÍCULOS NOVOS

O veículo **0 KM** apresenta algumas características próprias, se o compararmos aos veículos usados, pois o valor para base de cálculo é baseado no valor emitido na Nota Fiscal.

Para realizar o cálculo de IPVA de um veículo "zero", é necessário ter em mãos as seguintes informações: valor da nota fiscal, data de emissão da nota fiscal, marca do fabricante do veículo, modelo do veículo e combustível.

A classificação de um veículo **0 KM** para se obter a alíquota de IPVA é feita da mesma maneira já descrita para os veículos usados. Mas sua base de cálculo é diferente, como é mostrado abaixo:

IPVA = ((ValordaNotaFiscal * Aliquota) / 12) * Número_Mês_Restante

A base de cálculo nos mostra que o valor do IPVA para veículo 0 KM é proporcional. Isso quer dizer que, dependendo da data de emissão da nota fiscal, o valor do IPVA é feito de forma proporcional. Exemplo: Um veículo de categoria “Carro” com o combustível do tipo gasolina cuja nota fiscal foi emitida no mês de março do ano atual, com um valor de 35.000,00 reais, tem como valor de IPVA: 1.166,66

Veja: IPVA = ((35.000,00 * 4.0%) / 12) * 10
IPVA = (1.400,00 / 12) * 10
IPVA = 116,66 * 10
IPVA = 1.166,66

– FORMA DE PAGAMENTO

O IPVA apresenta alguns tipos de formas de pagamento que serão descritos a seguir, quer seja de veículo usado ou de veículo 0 KM.

– FORMA DE PAGAMENTO PARA VEÍCULOS USADOS

A forma de pagamento para veículos usados tem, como regra, apresentar um desconto de 3 % do valor de IPVA para os usuários que desejam pagar o valor integral antecipadamente, ou seja, no mês de janeiro do IPVA vigente. Caso os usuários não desejem antecipar seu pagamento, eles têm até o mês de fevereiro para pagarem seu IPVA total, sem desconto. Existe uma terceira opção que é o pagamento parcelado em até 3 (três) vezes, com vencimento nos meses de janeiro, fevereiro e março.

Independente da forma de pagamento e seu respectivo mês de vencimento, o IPVA apresenta outra validação, no que diz respeito aos vencimentos de IPVA. A

data de vencimento do IPVA está atrelada ao final da placa do veículo, ou seja, cada dígito final da placa do veículo tem uma data diferente de vencimento de seu IPVA. Essas datas são definidas pelo Estado e divulgadas alguns meses antes do pagamento.

– FORMA DE PAGAMENTO PARA VEÍCULOS NOVOS

Se o contribuinte pagar o IPVA, integralmente, em até 5 dias úteis, a partir da data de emissão da nota fiscal, terá um desconto de 3% no valor do IPVA. Caso contrário, o contribuinte deverá pagar em 3 parcelas, com data de vencimento atrelada à data de emissão da Nota Fiscal.

4.3.2 – LISTA DE REQUISITOS

A Lista de Requisitos mostrou as funcionalidades do software, sendo que essas informações foi possível criar classes para identificar esses dados da seguinte forma:

- **Cliente -** Retrata o que o sistema necessita do cliente para realizar um cálculo de IPVA.
- **Veículo -** Retrata quais informações são essenciais para realizar o cálculo de IPVA.
- **IPVA e ValorVenal -** Armazenamento dos dados do veículo para sererem utilizados no grupo de Cálculo de IPVA.
- **Cálculo -** Responsável pela realização do cálculo do IPVA.

Nos quadros abaixo são retratados os requisitos, os quais têm como objetivo mostrar As funcionalidades dentro do sistema, representados pela sua descrição, suas entradas, processamentos e saídas.

Requisitos para Clientes

No quadro abaixo são retratados os requisitos para Cliente. Nessa elicitação, foram coletadas as opções através das quais o Cliente pode interagir com o sistema, seus campos chaves e as funcionalidades básicas que o sistema deve ter com os requisitos para Cliente.

Quadro 1 – Requisitos para Clientes

1.) O sistema deve ter um menu com as opções:
   a. Cadastro de Clientes
   b. Cadastro de Veículo
   c. Orçamento de IPVA
   d. Sair

Entrada →	O Atendente escolhe uma das opções
Processamento →	Processa o item selecionado
Saída →	Mostra no monitor os menus para o item selecionado

2.) O sistema deve gravar os clientes, sendo que a chave de identificação seria o nome e CPF/CNPJ. O CPF/CNPJ deve ter um algoritmo de verificação.

Entrada →	O Atendente deve digitar os dados do cliente
Processamento →	Processa os campos para não serem vazios e verifica a validade do CPF/CNPJ.
Saída →	Grava as informações no Banco de dados.

3.) O sistema deve ter uma função que consiga visualizar o veículo que pertence a um determinado Cliente.

Entrada →	O Atendente deve selecionar o cliente desejado.
Processamento →	Processa todos o(s) veículo(s) que pertencem ao cliente selecionado.
Saída →	Mostra uma listagem do(s) veículo(s) do cliente selecionado.

4.) O sistema deve ter um recurso que possibilite a emissão de mala direta para os clientes. A listagem para mala direta deve ser feita através dos filtros:
   a. Por faixa de Nome
   b. Por data de nascimento
   c. Por Final de Placa do veículo

Entrada →	O Atendente deve selecionar o tipo de filtro que deseja imprimir.
Processamento →	Processa o tipo de filtro selecionado e agrupa em um subconjunto para ser impresso.
Saída →	Imprime os clientes selecionados nas etiquetas.

Requisitos para Veículos

No quadro abaixo são retratados os requisitos para Veículos. Nessa elicitação, foram coletados os campos e as associações necessárias, campos chaves, tipo de filtros e as funcionalidades básicas que o sistema deve ter com os requisitos para Veículo.

Quadro 2 – Requisitos para Veículos

5.) O sistema deve gravar o veículo tendo algumas regras:
   a. É necessário associar um cliente para um veículo
   b. Ter o número da placa do veículo
   c. Saber informações básicas como: Marca, Modelo, Ano, Combustível. Essas informações são chaves para o cálculo de IPVA.
   d. O combustível só é permitido do tipo Álcool, Gasolina, Gás ou Diesel.
   e. O veículo deve ter seu ano de Fabricação e Modelo, sendo que o modelo deve ser igual ao ano de fabricação ou um ano superior.

Entrada → O Atendente deve digitar os dados do Veículo.
Processamento → Processa os dados respeitando as regras mencionadas.
Saída → Grava os dados no Banco de dados.

6.) O sistema deve ter um processo para Procurar os clientes, com a finalidade de associar o veículo cadastrado, com um cliente, também cadastrado. O sistema deve permitir que o cliente tenha mais de um veículo cadastrado em seu nome.

Entrada → O Atendente digita os dados do Cliente.
Processamento → Filtra os clientes conforme a entrada.
Saída → Listar dados do cliente.

7.) O sistema deve ter um processo para Procurar pelas Placa do veículo cadastrado

Entrada → O Atendente digita a placa do veículo.
Processamento → Filtra os veículos conforme a entrada.
Saída → Listar dados do veículo.

8.) Um veículo obrigatoriamente deve ser do tipo (Carro, Moto, Utilitário, Caminhão)

Entrada → O Atendente seleciona o tipo do veículo.
Processamento → Filtra os tipos existentes de veículos.
Saída → Relaciona o veículo com o tipo selecionado.

9.) Um veículo obrigatoriamente deve apresentar um status (0KM ou Usado)

Entrada → O Atendente seleciona o status do veículo.
Processamento → Filtra os status existentes do veículo.
Saída → Relaciona o veículo com o status selecionado.

Requisitos para IPVA

No quadro abaixo são retratados os requisitos para IPVA. Nessa elicitação, foram coletados os dados necessários para realizar os cálculos, os procedimentos de cálculos, verificações e algumas regras básicas que o sistema deve ter com os requisitos para IPVA.

Quadro 3 – Requisitos para IPVA

10.) Para calcular um IPVA, é necessário saber qual exercício/ano deseja-se calcular, sendo que só é permitido calcular o IPVA do ano atual ou de até 5 (cinco) anos anteriores.

Entrada →	O Atendente digita o Ano de Exercício do IPVA.
Processamento →	Seleciona a tabela correspondente para cálculo.
Saída →	Mostra se o Ano de IPVA é válido ou não para cálculo.

11.) O sistema deve permitir cálculos de IPVA mesmo que o veículo não esteja cadastrado no sistema, nesse caso é necessário informar os dados básicos do veículo.

Entrada →	O Atendente digita os dados básicos do veículo.
Processamento →	Faz a validação dos dados e calcula o valor do IPVA.
Saída →	Mostra o valor do IPVA do veículo selecionado.

12.) Para calcular o IPVA é necessário ter o valor venal do veículo. O valor venal é um valor médio do veículo determinado pelo Governo do Estado. O sistema deve obter esse valor a partir de informações básicas do veículo.

Entrada →	Informar dados do Veículo
Processamento →	Processar os dados do veículo com base no ano de Exercício do IPVA
Saída →	Obter o Valor venal do Veículo

13.) O sistema deve verificar se o IPVA está atrasado com base no final da placa do veículo.

Entrada →	Informar a data do dia
Processamento →	Processar os dados é verificar se está atrasado ou não
Saída →	Valor da Multa, caso tiver.

14.) O sistema deve gravar o veículo, tendo algumas regras:

a. Saber informações básicas como: Marca, Modelo, Ano, Combustível. Essas informações são chaves para o cálculo de IPVA.
b. O combustível só é permitido do tipo Álcool, Gasolina ou Diesel.
c. O veículo deve ter seu ano de Fabricação e Modelo, sendo que o modelo deve ser igual ao ano de fabricação ou um ano superior.

Entrada →	O Atendente deve digitar os dados do Veículo.
Processamento →	Processar os dados respeitando as regras mencionadas.
Saída →	Gravar os dados no Banco de dados.

Requisitos para o Valor Venal do Veículo

No quadro abaixo são retratados os requisitos para Valor Venal. Nessa elicitação, foram coletados os dados necessários para atualizar os valores venais dos veículos, suas regras e filtros que o sistema deve ter com o requisito Valor Venal.

Quadro 4 – Requisitos para o Valor Venal.

15.) Todo início de ano é necessário acrescentar os valores venais dos veículos que são promulgados no Diário Oficial da União, sendo necessário ter uma opção no sistema para acrescentar ou até mesmo alterar o valor venal dos veículos.

Entrada →	Informar os dados do veículo e seu valor venal
Processamento →	Processar se todas as entradas são válidas
Saída →	Gravar os dados no Banco de Dados.

16.) Como regra, é necessário ter armazenado o valor venal do veículo do ano atual até 10(dez) anos anteriores, sendo que se for calcular um veículo que não estiver nessa faixa, será enquadrado como “Demais Modelos”.

Entrada →	Informar os dados do veículo e seu valor venal
Processamento →	Processar se todas as entradas são válidas
Saída →	Validar os valores processados.

17.) O sistema deve ter uma opção de filtrar os valores venais conforme o ano de exercício do IPVA.

Entrada →	Informar o Ano de Exercício do IPVA
Processamento →	Filtrar todos os valores venais do Ano de Exercício
Saída →	Armazenar o resultado filtrado em memória, com a possibilidade de exibição na tela.

Requisitos para Calcular o IPVA

No quadro abaixo são retratados os requisitos para Calcular. Nessa elicitação, foram coletados os dados necessários para identificar as alíquotas dos veículos, calcular seu imposto e as verificações das regras de validação que o sistema deve ter com o requisito Cálculo.

Quadro 5 – Requisitos para Cálculo

18.) O sistema deve conseguir calcular a alíquota de imposto para cada veículo.
   a. Se o tipo do veículo for um Carro

Entrada →	Informar o tipo do veículo igual a Carro e o Combustível (Gasolina, Álcool ou Diesel).
Processamento →	Processar os dados e verificar qual é a alíquota de imposto do veículo.
Saída →	Armazenar a alíquota no atributo *ValorAliquota* na classe *Cálculo*.

19.) O sistema deve calcular a alíquota de imposto para cada veículo.
   a. Se o tipo do veículo for uma Moto

Entrada →	Informar o tipo do veículo igual a Moto e o Combustível (Gasolina, Álcool ou Diesel).
Processamento →	Processa os dados e verificar qual é a alíquota de imposto do veículo.
Saída →	Armazenar a alíquota no atributo *ValorAliquota* na classe *Cálculo*.

20.) O sistema deve calcular a alíquota de imposto para cada veículo.
   a. Se o tipo do veículo for um Caminhão

Entrada →	Informar o tipo do veículo igual a Caminhão e o Combustível (Gasolina, Álcool ou Diesel).
Processamento →	Processar os dados e verificar qual é a alíquota de imposto do veículo.
Saída →	Armazenar a alíquota no atributo *ValorAliquota* na classe *Cálculo*.

21.) O sistema deve calcular o valor do IPVA, para isso são necessárias as seguintes informações:
   a. Valor Venal do Veículo
   b. Valor da Alíquota ou porcentagem do Imposto

Entrada →	Informar os dados necessários.
Processamento →	Aplicar a fórmula de cálculo de IPVA. **IPVA = ValorVenal * Aliquota**
Saída →	Armazenar o resultado no atributo *ValorIPVA*

22.) O sistema deve verificar se o IPVA está atrasado ou não. Em caso de atraso o sistema deve calcular o valor da multa.

Entrada →	Informar os dados necessários.
Processamento →	Processar a data de Vencimento (verifica se está atrasado, caso tenha multa realiza o calcula)
Saída →	Retornar o valor da multa, caso tenha.

4.3.3 – DIAGRAMA DE CLASSES

A representação do diagrama de classes proposta pela UML tem como objetivo mostrar os relacionamentos que as classes têm dentro do sistema. Uma das características do diagrama de classes é mostrar uma visão estática do software.

A representação abaixo mostra o diagrama de classes do sistema aplicativo para Despachante, relacionado ao módulo de cálculo de IPVA.

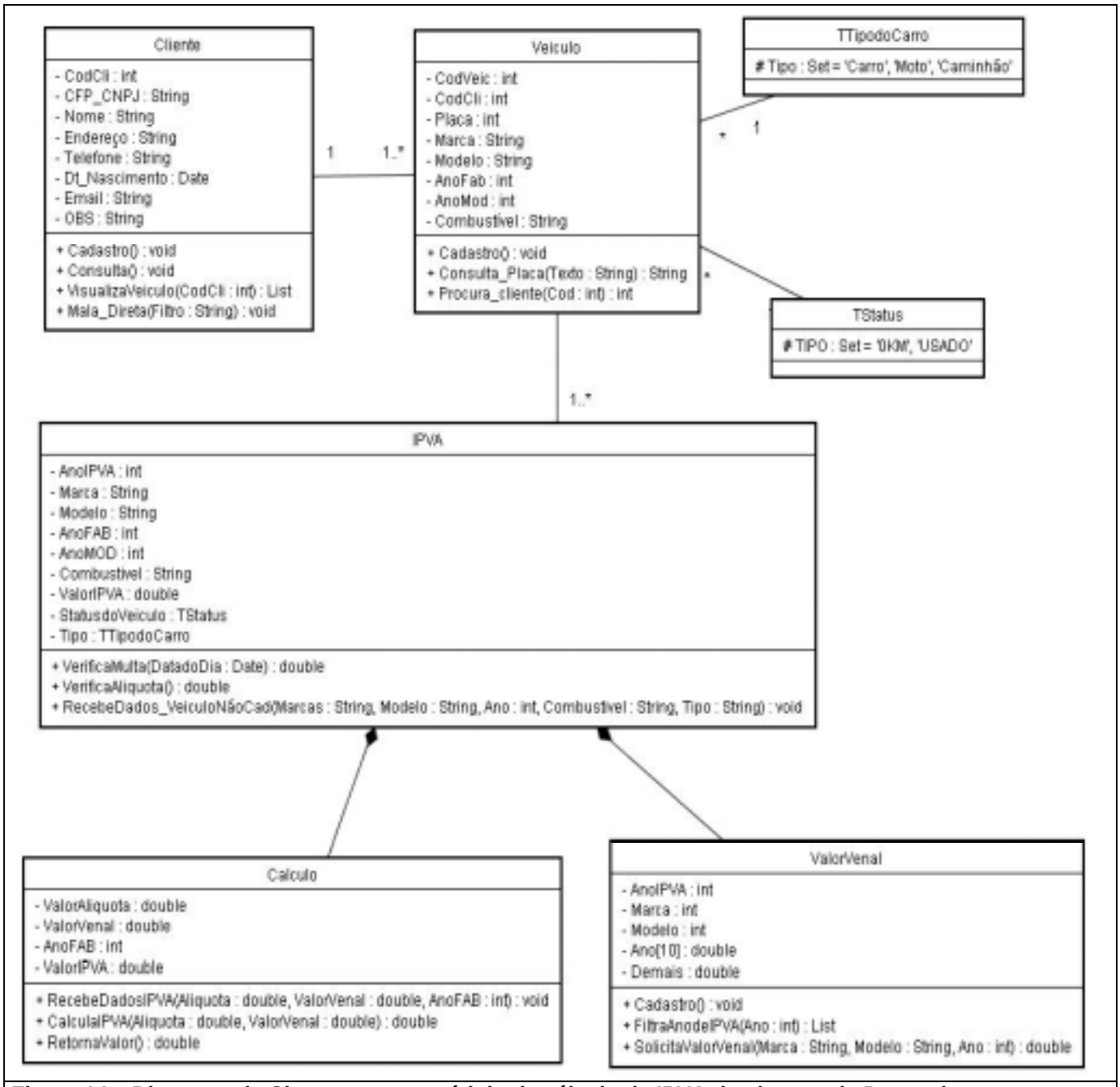

Figura 14 – Diagrama de Classes para o módulo de cálculo de IPVA do sistema de Despachante

4.3.4 – DIAGRAMA DE CASOS DE USO

O Diagrama de Casos de Uso retrata a interação entre os atores (Usuários do Sistema) e os Casos de Uso do sistema.

Nesse contexto do estudo de caso foi atribuída à função de ator ao atendente, o qual realiza as interações com os casos de uso.

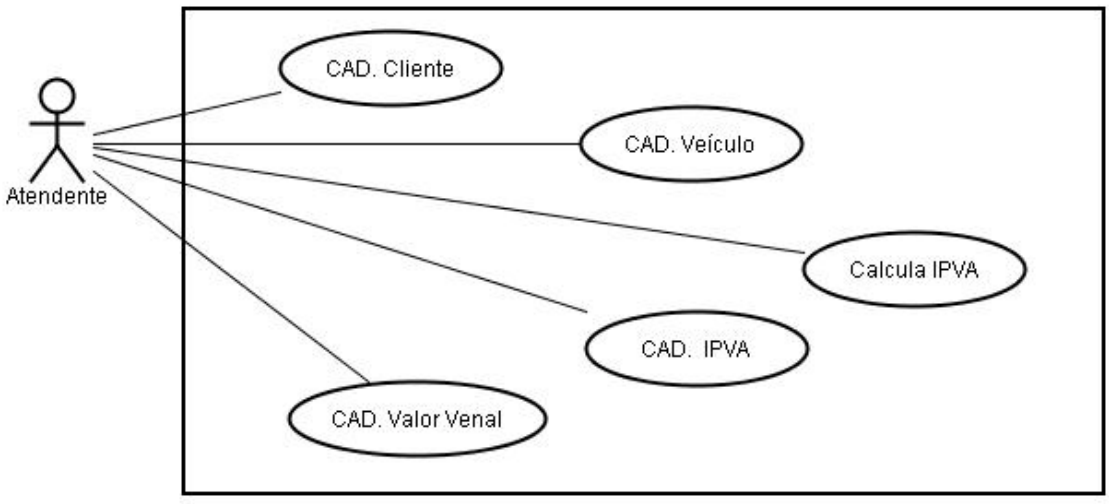

Figura 15 – Diagrama de Casos de Uso do Módulo de cálculo de IPVA

O diagrama acima mostra a interação que o atendente deve ter com os caso de uso, sendo que para realizar um cálculo de IPVA. Nesse caso o atendente deve cadastrar as informações do Cliente e seu Veículo. Depois o atendente informa o veículo cadastrado e o sistema realiza o cálculo de IPVA. Os Casos de Usos Cadastro de IPVA e Cadastro de Valor Venal, só terão interações se tiver que inserir novos modelos ou se tiver alteração no cálculo do IPVA.

4.3.5 – DIAGRAMA DE ATIVIDADES

O diagrama de atividades tem como objetivo mostrar o fluxo de funcionamento do sistema. No contexto do estudo de caso essa representação é muito importante, pois mostram quais são os fluxos possíveis dentro do módulo de cálculo de IPVA.

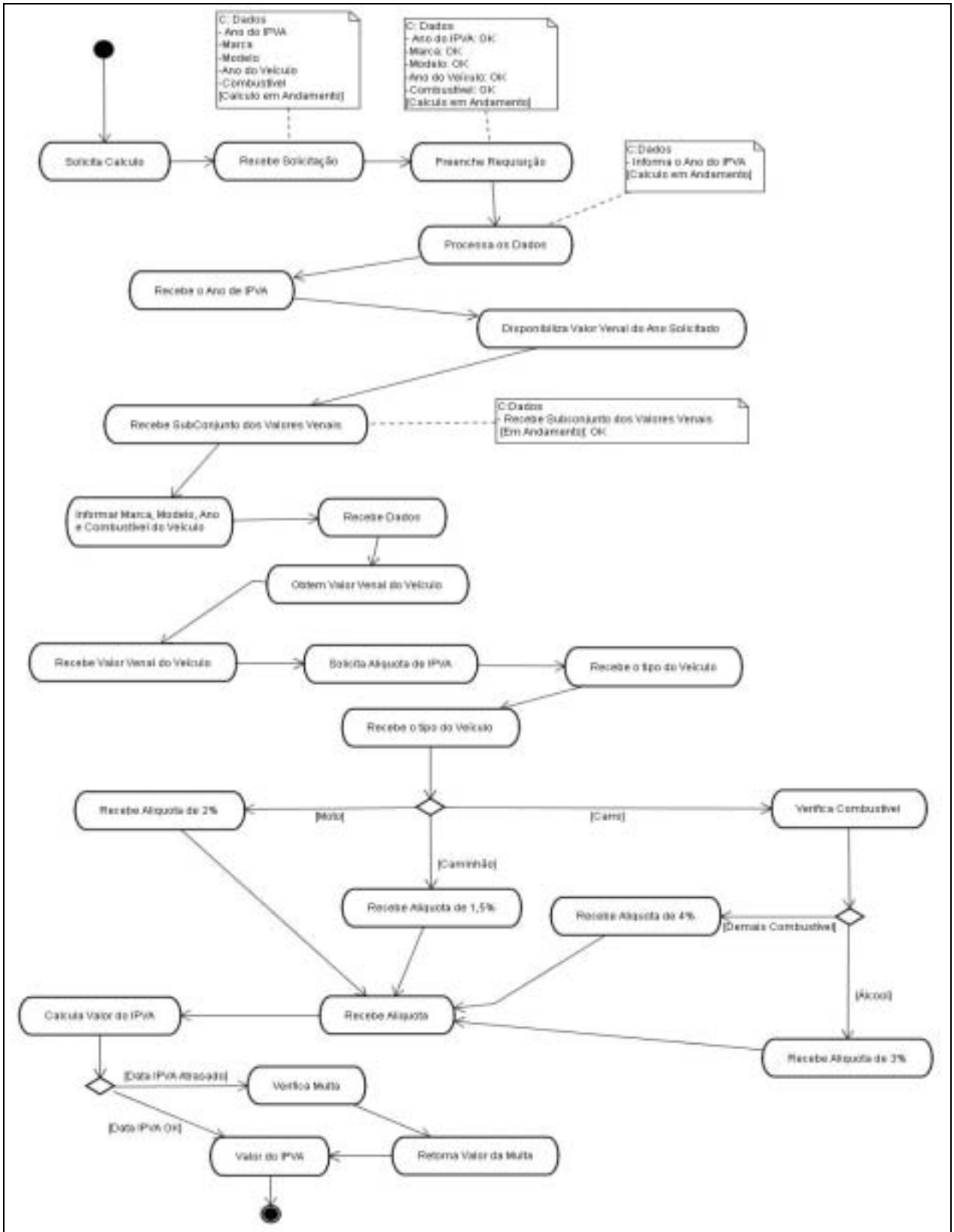

Figura 16 – Diagrama de Atividades do Módulo de cálculo de IPVA

4.4 – ESPECIFICAÇÃO COM A LINGUAGEM OCL

A Linguagem OCL permite a criação de uma especificação denominada especificação formal, pois o seu resultado constitui uma informação mais precisa e detalhada do quê o software deve fazer.

Uma das principais características dessa linguagem é a utilização de uma escrita próxima às notações matemáticas (simbólicas), e sua função é a de representar restrições em atributos e métodos.

Para que a compreensão da especificação em OCL tenha uma amplitude maior, faz-se necessária uma explicação mínima de suas palavras-chaves nas expressões. A tabela abaixo mostra essa explicação, de forma simples, sobre o significado de cada palavra-chave na linguagem OCL.

a) **context** <classe> ➔ Essa palavra-chave mostra que a especificação OCL está restringindo, métodos e Atributos, da classe especificada.
b) **context** <classe>::Método ➔ Essa é a sintaxe que representa um método pertencente à classe especificada. Um método nada mais é do que uma ação da classe que será acionada em alguma circunstância.
c) **inv**: ➔ Demonstra que a restrição feita em um atributo é invariável, ou seja, não sofre alteração em nenhuma instância nessa classe nesse contexto.
d) **pre**: ➔ Demonstra uma restrição que deve atender a uma determinada condição antes de sua execução.
e) **post**: ➔ Demonstra uma restrição que deve atender a uma determinada condição depois de sua execução.

Na representação abaixo foi feita uma especificação em Linguagem OCL para o módulo de cálculo de IPVA do sistema aplicativo para Despachante.

Quadro 6 – Especificação OCL no contexto Cliente

package DESPACHANTE

context Cliente

inv: self.CPF_CNPJ->exists(self.CPF_CNPJ)

context Cliente::VisualizaVeiculo(CodCli: Int) :List

inv: notEmpty(CodCli)

pre: self.CodCli->exists(VeiculoCodCli)

post: --none

context Cliente::Mala_Direta(Filtro: String): Void

post: return = Cliente->Select(Filtro)

Quadro 7 – Especificação OCL no contexto Veículo

context Veiculo

inv: self.codcli->exists(Cliente.codcli)

inv: notEmpty(self.Placa)

inv: (self.Marca->exists(ValorVenal.Marca) and self.Modelo->exists(ValorVenal.Modelo))

inv: (self.AnoFab > 1900) and (self.anoFab <= year(Date()))

inv: (self.AnoMOD > 1900) and (self.anoMOD <= year(Date())+1)

inv: (self.AnoMOD = self.AnoFab) or (self.AnoMOD = self.AnoFab+1)

inv: (self.Combustivel = Sequence{'Gasolina', 'Alcool', 'Diesel', 'Gás' }) and (notEmpty(self.Combustivel))

context Veiculo::Consulta_Placa(Texto: String): String

inv : notEmpty(texto)

pre : Texto->exists(Veiculo.Placa)

post: return = Veiculo->Select(Texto)

context Veiculo::Procura_Cliente(Codigo: int): Int

inv : notEmpty(Codigo)

pre : (Codigo > 0) and (Codigo <= Cliente.size)

post: -- none

Quadro 8 – Especificação OCL no contexto IPVA

context IPVA

inv: (self.AnoIPVA = year(date())) and (self.AnoIPVA = year(date())-5)

inv: (self.Marca->exists(ValorVenal.Marca) and self.Modelo->exists(ValorVenal.Modelo))

inv: (self.AnoFab > 1900) and (self.anoFab <= year(Date()))

inv: (self.AnoMOD > 1900) and (self.anoMOD <= year(Date())+1)

inv: (self.AnoMOD = self.AnoFab) or (self.AnoMOD = self.AnoFab+1)

inv: (self.Combustivel = Sequence{'Gasolina', 'Alcool', 'Dieesel' }) and
(notEmpty(self.Combustivel))

inv: (self.ValorIPVA > 0)

inv: (self.statusdoveiculo = Set{'0KM', 'USADO'}) and (notEmpty(self.StatusdoVeiculo))

context IPVA::VerificaMulta(DatadoDia: Date): Double

pre : --none

context IPVA::VerificaAliquota():Float

inv : notEmpty(self.Combustivel) and notEmpty(self.AnoFaB) and isnotNull(self.tipo)

pre: --none

post: return = (if (self.tipo = 'Moto') then 0.02
else if (self.tipo = 'Caminhao') then 0.015
else if (self.tipo = 'Carro') and (self.Combustivel = 'Alcool') then 0.03
else 0.04
endif
endif
endif)

context IPVA::RecebeDados_VeiculoNaoCad(Marcas : String, Modelo : String, Ano : int, Combustivel : String, tipo: String) : void

inv: notEmpty(Marca) and notEmpty(Modelo) and notEmpty(Ano) and notEmpty(Combustivel)

pre: (self.Marca->exists(ValorVenal.Marca) and self.Modelo->exists(ValorVenal.Modelo))

Quadro 9 – Especificação OCL no contexto Valor Venal

```
context ValorVenal
        inv: notEmpty(self.AnoFAB)
context ValorVenal::FiltraAnodeIPVA( ANO: Int ): void
        post: return = Select( self.AnoFAB = ANO )
context ValorVenal::SolicitaValorVenal(Marca: String, Modelo: String, Ano: Int): Double
        inv     : notEmpty(Marca) and notEmpty(Modelo) and notEmpty(ano)
        pre     : --none
        post: return = (if ( ( year(date()) - ano)+1 = 1 ) then ValorVenal.Ano1
                        else if ( ( year(date()) - ano)+1 = 2 ) then ValorVenal.Ano2
                                else if ( ( year(date()) - ano)+1 = 3 ) then ValorVenal.Ano3
                                        else if ( ( year(date()) - ano)+1 = 4 ) then ValorVenal.Ano4
                                                else if ( ( year(date()) - ano)+1 = 5 ) then ValorVenal.Ano5
                                                        else if ( ( year(date()) - ano)+1 = 6 ) then ValorVenal.Ano6
                                                                else if ( ( year(date()) - ano)+1 = 7 ) then
                                                                        ValorVenal.Ano7
                                                                        else if ( ( year(date()) - ano)+1 = 8 ) then
                                                                                ValorVenal.Ano8
                                                                                else if ( ( year(date()) - ano)+1 = 9 ) then
                                                                                        ValorVenal.Ano9
                                                                                        else if ( ( year(date()) - ano)+1 = 10 )
                                                                                                then ValorVenal.Ano10
                                                                                                else ValorVenal.Demais

                                                                                        endif
                                                                                endif
                                                                        endif
                                                                endif
                                                        endif
                                                endif
                                        endif
                                endif
                        endif)
```

Quadro 10 – Especificação OCL no contexto Valor Venal

```
context Calculo
        inv: (self.valorAliquota >= 1.5) and (self.valorAliquota <= 4)
        inv: Self.ValorVenal > 0
        inv: ( self.AnoFab > 1900 ) and (self.anoFab <= year(Date()) )
        inv: self.ValorIPVA > 0
context Calculo::CalculaIPVA( Aliquota: Double, ValorVenal: Double) Double
        inv: notEmpty(Aliquota) and notEmpty(ValorVenal) and notEmpty(AnoFAB)
        pre: ( Aliquota > 0 ) and ( ValorVenal > 0 )
        post: Return = ValorVenal * Aliquota

endpackage
```

4. 5 - QUESTIONÁRIO

A técnica escolhida para a avaliação da Documentação de Especificação de Requisitos foi a do questionário, a fim de se avaliar o grau de ambigüidade, completude e precisão que a documentação pudesse apresentar.

Para efeito de avaliação da especificação de requisitos, foi considerada a seguinte definição: uma especificação apresenta-se ambígua quando nela se constata mais de uma interpretação de um requisito seus requisitos.

Para que uma especificação de requisitos seja considerada completa, está deve oferecer informações sobre as funções a serem implementadas no software. Portanto, o conjunto de informações presentes na especificação obriga a fornecer plenas condições para o entendimento do que necessita ser implementado.

Finalmente, considera-se que uma especificação de requisitos é precisa quando descreve, o que se espera que seja implementado no software. Quando possível, espera-se uma descrição numérica (quantitativa) do comportamento do software.

CAPÍTULO 5 – RESULTADOS E DISCUSSÕES

Realizada a pesquisa, os resultados obtidos pelos entrevistados foram separados em duas categorias: os que responderam à especificação com OCL e sem OCL, conforme tabelas abaixo:

Tabela 6 - Respostas dos entrevistados à especificação sem OCL

Ambiguidade	Muito Baixa	Baixa	Regular	Alta	Muito Alta	Total
Cliente		3	3	1		7
Veículo		4	3			7
IPVA		5	2			7
Valor Venal		7				7
Cálculo		7				7

Completude	Muito Baixa	Baixa	Regular	Alta	Muito Alta	Total
Cliente		1	3	3		7
Veículo			3	4		7
IPVA		1	3	3		7
Valor Venal		1	2	4		7
Cálculo			4	3		7

Precisão	Muito Baixa	Baixa	Regular	Alta	Muito Alta	Total
Cliente			4	3		7
Veículo		1	4	2		7
IPVA			2	5		7
Valor Venal		1	2	4		7
Cálculo			3	4		7

Tabela 7 - Respostas dos entrevistados à especificação com OCL

Ambiguidade	Muito Baixa	Baixa	Regular	Alta	Muito Alta	Total
Cliente	4	3				7
Veículo	1	5	1			7
IPVA	3	3		1		7
Valor Venal	1	4		2		7
Calculo	2	3	1	1		7

Completude	Muito Baixa	Baixa	Regular	Alta	Muito Alta	Total
Cliente			2	4	1	7
Veículo		1	3	3		7
IPVA			3	4		7
Valor Venal			2	5		7
Calculo		2	2	2	1	7

Precisão	Muito Baixa	Baixa	Regular	Alta	Muito Alta	Total
Cliente			2	4	1	7
Veículo		1	4	1	1	7
IPVA		1	1	5		7
Valor Venal			3	4		7
Calculo			2	4	1	7

Baseando-se nos dados obtidos pelas entrevistas, consegue-se realizar a seguinte totalização:

Tabela 8 – Estatística Geral dos Resultados Obtidos

Estatística Geral	Muito Baixa	Baixa	Regular	Alta	Muito Alta
Ambiguidade com OCL	31%	51%	6%	11%	0%
Ambiguidade sem OCL	0%	74%	23%	3%	0%
Completude com OCL	0%	9%	34%	51%	6%
Completude sem OCL	0%	9%	43%	49%	0%
Precisão com OCL	0%	6%	34%	51%	9%
Precisão sem OCL	0%	6%	43%	51%	0%

Pode-se analisar, através dos gráficos abaixo, que os resultados obtidos entre as respostas da especificação com OCL e sem OCL foram diferentes em vários aspectos, como se observa.

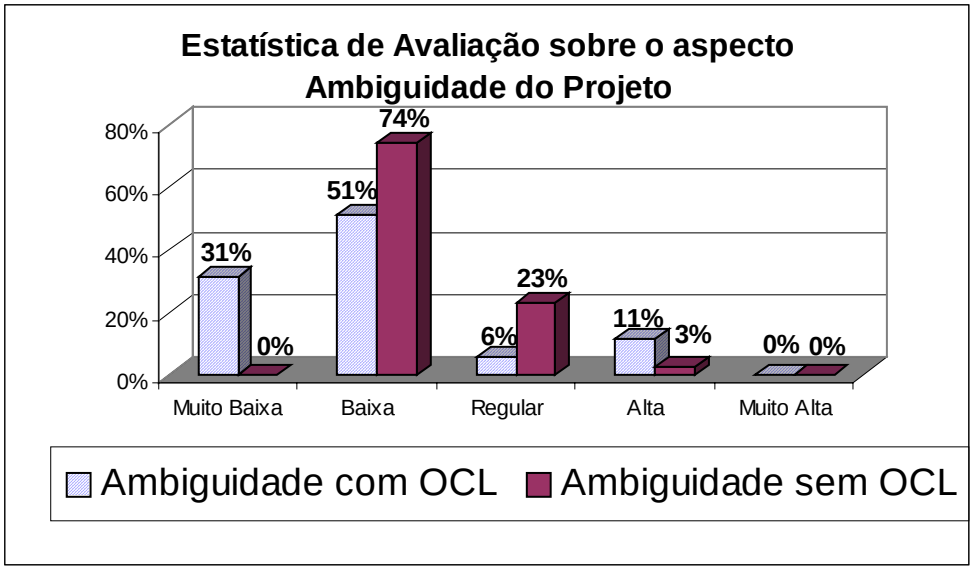

Figura 17 – Estatística de Ambiguidade

No item Ambigüidade, analisando as respostas dos entrevistados, conclui-se que a especificação sem OCL apresentou uma ambigüidade baixa, significando que os entrevistados tiveram um bom entendimento da especificação; mas pode-se observar claramente que as respostas dos entrevistados à especificação com OCL, revelam um nível de entendimento melhor, porque mais de 80% dos entrevistados responderam que a ambigüidade estava baixa ou muito baixa, demonstrando assim um entendimento melhor do que a especificação estava retratando. Um aspecto muito importante é que nenhum dos entrevistados pelo que responderam o questionário sem OCL, informou que a especificação estava muito baixa. Ao passo que 31% dos que responderam ao questionário com OCL consideraram que a ambigüidade estava muito baixa.

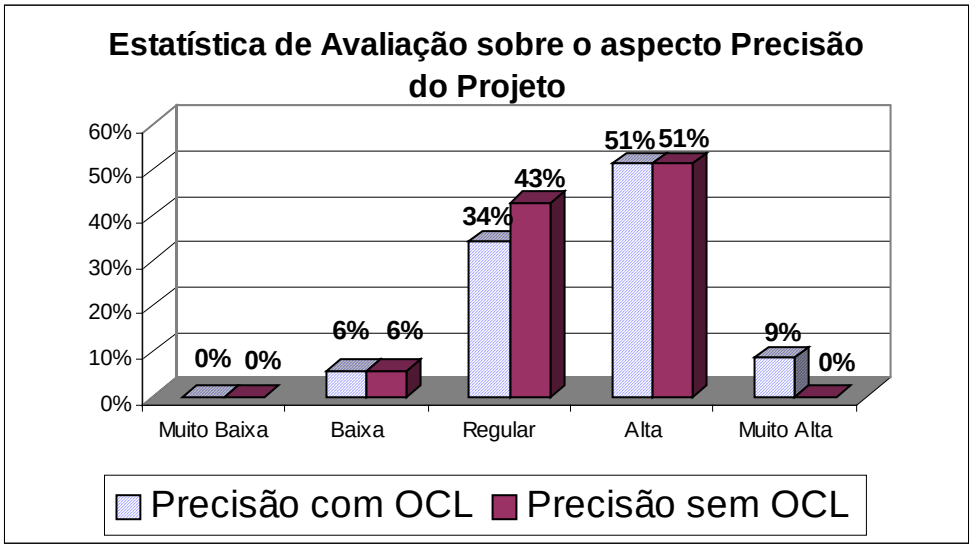

Figura 18 – Estatística de Precisão

No item Precisão, analisando as respostas dos entrevistados, conclui-se que a especificação sem OCL e a com OCL obtiveram resultados muito parecidos, ficando entre regular e alta a precisão do projeto. Mas importa chamar a atenção sobre os resultados no item “Muito Alta”, com ocorrência maior para a especificação com OCL, obtendo resultados de 9%, e 0% para a especificação sem OCL. Conclui-se que a precisão teve um aumento utilizando OCL.

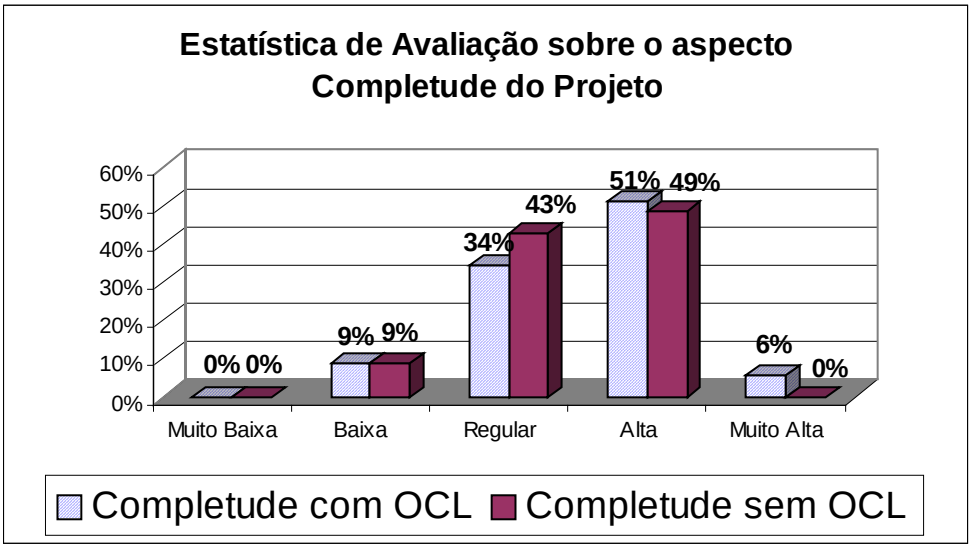

Figura 19 – Estatística de Completude

No item Completude, analisando as respostas dos entrevistados, conclui-se que a especificação com OCL e sem OCL, obtiveram resultados muito semelhantes ao da precisão, ou seja, ficando entre regular e alta a completude do projeto. Mas pode-se observar que o resultado das respostas “Alto” e “Muito Alto” tem uma ocorrência maior para a especificação com OCL, com índices de respostas de 51%, e 49% para a especificação sem OCL. Já no item Completude Muito Alta, a diferença entre os índices foi mais acentuada, pois obteve 6% a especificação com OCL e 0% a especificação sem OCL, concluindo-se que houve um aumento na completude, utilizando OCL.

CAPÍTULO 6 – CONCLUSÃO

Após finalizar a especificação de requisitos do software utilizando a Linguagem OCL, alguns pontos tiveram destaque interessante. Entre esses pontos, um deles seria quanto à utilização da linguagem OCL, que é relativamente rápida, mas deve-se ter um conhecimento prévio das informações sobre o sistema, isso porque quando se coloca as informações de acordo com a estrutura da linguagem OCL, consegue-se observar falhas de consistência das informações recebidas na elicitação dos dados, isso faz com que se realizem novas elicitações para entender melhor e conseguir “amarrar” essas informações.

Um outro aspecto foi a etapa de especificação que levou um tempo maior de elaboração do que a especificação sem OCL. Nessa variação de tempo deve ser levado em conta o tamanho do projeto analisado. Baseando-se nos resultados mostrados, pode-se concluir que o projeto com OCL teve um tempo superior na sua elaboração de 50% a mais do que o projeto sem OCL. Esse fato é bastante significativo pois, dependendo do tamanho do projeto, torna-se inviável sua utilização.

Outro ponto interessante é que a linguagem OCL é uma linguagem muito próxima a uma linguagem de programação, ou seja, a sintaxe utilizada para a criação de suas sentenças é semelhante à de uma linguagem de programação, sendo assim, um aspecto negativo é que o profissional deve ter um conhecimento prévio de uma linguagem para que seu aprendizado de OCL seja mais rápido.

Os resultados obtidos entre as especificações sem OCL e com OCL se mostraram semelhantes, isso porque quando foi realizada a especificação inicial, ou seja, sem OCL, várias informações não foram capturadas no momento da elicitação, mas somente quando aplicada a descrição OCL. A partir do uso da OCL foi possível observar inconsistências nas especificações iniciais, levando a um novo processo de elicitação para que o sistema se

tornasse mais completo. Esse fato foi relevante na criação da especificação, pois mostrou que a descrição OCL ajudou a tornar a especificação mais completa, mais precisa e menos ambuiga. Porém, acreditamos que isto acabou afetando os resultados da comparação junto aos entrevistados, pois a documentação já tinha sofrido modificações necessárias para que ela se tornasse mais completa, precisa e menos ambuiga, resultando assim em respostas muito semelhantes dos entrevistados que responderam o questionário com OCL e dos entrevistados que responderam o questionário sem OCL.

Esses pontos são interessantes, mas, para as empresas que desenvolvem software mais do que a consistência das informações, procuram-se outros aspectos como rápidez e baixo custo na elaboração do software. Nesse caso, a linguagem OCL apresentada pode ajudar ao longo do tempo, isso porque, quando-se utliza OCL em uma especificação de software, seu tempo de criação é maior, mas em contra partida seu tempo de elaboração é um pouco menor, essa equação precisa ser balanceada, ou seja, dependendo do tamanho do projeto e o tempo necessário para sua entrega.

Para o futuro seria interessante criar ferramentas que auxiliem na elaboração das sentenças OCL, as quais possam fazer as verificações das sentenças como também apresentar ajuda dos comandos que podem ser utilizados. Também seria interessante fazer comparações com outras linguagens formais como a Linugagem Z, Object-Z, OhCircus entre outras, para fins de analisar se a Linguagem OCL é mais eficiente ou não perante essas outras linguagens. É para concluir, poderia ser feito um estudo das possibilidadedes de criar um software que consiga integrar a UML com a Linguagem OCL.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVARES, JUAN CARLOS GRANJA. – **Software Maintenance: Analysis of estimates based on requirement specification in the context of New Technologies, a Case Study**, Editora IEEE, 2004.

BARROS, OSCAR. **Requeriments Elicitiation and Formalization through external design and Object-Oriented Specification**, Editora IEEE, 1993.

BOOCH, G. – **Object-Oriented Analysis and Design: With Applications (second edition)**. Redwood City, California, U.S.A.: The Benjamin/Cummings Publishing Company. 1994.

BOOCH, G., RUMBAUGH, J. e JACOBSON, I. – **UML Guia do Usuário**. Editora Campus, 2000.

BREITMAN, KARIN K e LEITE, JÚLIO C. S. P. – **A Framework for Scenario Evolution**. ICRE'98 Third International Conference on Requirements Engineering. Colorado Springs, Colorado, USA. 1 ed. USA: IEEE CSP, Los Alamitos, CA. Proceedings, pp. 214-221. Abril, 1998.

CASTRO, J. F. **Introdução a Engenharia de Requisitos**. XV Congresso da Sociedade Brasileira de Computação, XIV Jornada de Atualização em Informática - JAI 95. Canela, RS. 1995.

CASTRO, J., KOLP, M. e MYLOPOULOS, J. – **"Towards Requirements-Driven Information Systems Engineering: The Tropos Project"**., Elsevier, Amsterdam, The Netherlands., 2002.

GOGUEN J. e LINDE, CHARLOTTE. – **Techniques for Requirements Elicitation**. – IEEE In: Software Requirements Enginnering Second Edition, 1997.

JONES, MEILER PAGE; **Fundamentos do desenho orietado a objeto com UML** . Editora Makron Books, 2000.

KITCHENHAM, BARBARA ANN. – **Evaluating Software Engineering Methods and Tool Part 3**: Selecting an appropriate evaluation method - practical issues. ACM – july 1996.

KOTONYA, P. e SOMMERVILLE, I. **Requirements Engineering: Processes and Techniques**. 1ed England: John Wiley & Sons Ltd. 1998.

LEE, RICAHRD e TEPFENHART, WILLIAM. – **UML e C++ - Guia Prático de Desenvolvimento Orientado a Objeto**. Editora Makron Books, 2001.

LEITE J.C.S.P. -, "**Engenharia de Requisitos- Notas de Aula**", 1994.

LEITE J.C.S.P., et al. **A Scenario Construction Process.**, Springer-Verlag London Limited. 2000.

MACAULAY, LINDA A. – **Requirements Engineering 1ed**. Great Britain: Springer-Verlag London Limited. 1996.

OMG 2003 (Object Management Group). **Object Constraints Language Specification version 2.0**. URL http://www.omg.org/docs/ptc/03-10-14.pdf (2003)

IEEE STD. 830-98. – **“Recommended Practice for Software Requirements Specification"**. The Institute of Electrical and Electronics Engineers. New York, 1990.

PAGLIOSA, PAULO A. – **Projeto de Sistemas Orientado a Objetos.** Departamento de Computação e Estatística da Universidade Federal de Mato Grosso do Sul, Outubro de 2000.

PAULA FILHO, WILSON DE PÁDUA. - **Engenharia de Software: Fundamentos, Métodos e Padrões**. LTC Editora. Rio de Janeiro - RJ, 2001.

PINHEIRO, F. A. C. – **Formal and Informal Aspects of Requirements Tracing.** In: III Workshop de Engenharia de Requisitos, Anais... PUC-Rio p. 1-21, Rio de Janeiro - RJ, 2000.

PRESSMAN, ROGER S. **Software Enginnering: A practitioner Approach**. Editora McGraw Hill, 1997.

RAMBAUGH, J. et al. – **Object-Oriented Modeling and Design**. Prentice-Hall, 1991.

ROCCO, NICOLAS DE, et al. **Software: Pratice & Experience volume 32**, issue 14, nov. ACM, 2002.

SAIEDIAN, HOSSEIN. – **Formal Methods in Information Systems Engineering**. PhD Department of Computer Science of University of Nebraska at Omaha. – IEEE in: Software Requirements Enginnering Second Edition, 1997.

SCHWARTZ, J.T. – **Automatic Data Structure Choice in a Language de Very High Level**. New York University. New York, ACM, 1975.

SILVA, A. – **Requirements, Domain and Specification: A Viewpoint-Basead Approach to Requirements Enginnering**, - International Conference on Software Engineering - Proceedings of the 24th international conference on Software engineering, Orlando, Florida, 2002. Society Publisher ACM Press New York, NY, USA. Pages: 94 – 104. ISBN:1-58113-472-X

SOMMERVILLE, IAN – **Engenharia de Software 6ª edição**, Editora Addison Wesley, 2000

TABELING, PETER, GRONE, BERNHARD - **Integrative Architecture Elicitation for Large Computer Based Systems -** IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2005.

WALLACE, DOLORES R. e IPPOLITO, LAURA M. – **Verifying and Validation Software Requirements Specifications**. – IEEE In: Software Requirements Enginnering Second Edition, 1997.

WARMER, JOS B. e KLEPPE, ANNEKE G. **The Object Constraint Language: Precise Modeling with UML**. Addison-Wesley. USA, 1999.

YOURDON, EDWARD. – **Análise Estruturada Moderna. 3a ed.**, Rio de Janeiro, Editora Campus, 1990.

ANEXO.

A Especificação do Sistema de Despachante do módulo de IPVA foi elaborada, e mostrada no capítulo 4 – Estudo de caso, sendo que o questionário das especificações com descrição OCL e sem a descrição OCL são os mesmos como e apresentado abaixo.

1) A ambigüidade da especificação de requisitos do sistema referentes às informações de CLIENTE está: ( ) Muito Baixa ( ) Baixa ( ) Regular ( ) Alta ( ) Muito Alta	4) Há ambigüidade da especificação de requisitos do sistema referentes às informações de VEÍCULO está: ( ) Muito Baixa ( ) Baixa ( ) Regular ( ) Alta ( ) Muito Alta
2) A Completude da especificação de requisitos do sistema referentes às informações de CLIENTE está: ( ) Muito Baixa ( ) Baixa ( ) Regular ( ) Alta ( ) Muito Alta	5) A Completude da especificação de requisitos do sistema referentes às informações de VEÍCULO está: ( ) Muito Baixa ( ) Baixa ( ) Regular ( ) Alta ( ) Muito Alta
3) A Precisão da especificação de requisitos do sistema referentes às informações de CLIENTE está: ( ) Muito Baixa ( ) Baixa ( ) Regular ( ) Alta ( ) Muito Alta	6) A Precisão da especificação de requisitos do sistema referentes às informações de VEÍCULO está: ( ) Muito Baixa ( ) Baixa ( ) Regular ( ) Alta ( ) Muito Alta

7) Há ambigüidade da especificação de requisitos do sistema referentes às informações de IPVA está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta

8) A Completude da especificação de requisitos do sistema referentes às informações de IPVA está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta

9) A Precisão da especificação de requisitos do sistema referentes às informações de IPVA está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta

10) Há ambigüidade da especificação de requisitos do sistema referentes às informações de VALORVENAL está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta

11) A Completude da especificação de requisitos do sistema referentes às informações de VALORVENAL está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta

12) A Precisão da especificação de requisitos do sistema referentes às informações de VALORVENAL está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta

13) Há ambigüidade da especificação de requisitos do sistema referentes às informações de CALCULO está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta

14) A Completude da especificação de requisitos do sistema referentes às informações de CALCULO está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta

15) A Precisão da especificação de requisitos do sistema referentes às informações de CALCULO está:

( ) Muito Baixa
( ) Baixa
( ) Regular
( ) Alta
( ) Muito Alta