CENTRO UNIVERSITÁRIO FUNDAÇÃO SANTO ANDRÉ

CARLOS WILKER
CAROLINA CAMARGO GONÇALVES
MATHEUS MARABESI PEREIRA

FERRAMENTA PARA GERENCIAMENTO DE CASO DE USO

SANTO ANDRÉ – SP
2016

CARLOS WILKER
CAROLINA CAMARGO GONÇALVES
MATHEUS MARABESI

FERRAMENTA PARA GERENCIAMENTO DE CASO DE USO

Monografia apresentada como exigência parcial para obtenção do título de Especialista em Engenharia de Software ao Programa de Pós-Graduação do Centro Universitário Fundação Santo André

Orientador:
Professor Mestre Rubens Sales

SANTO ANDRÉ – SP
2016

Ficha catalográfica elaborada pela Biblioteca Comunitária da FSA

Paulino, Carlos

Ferramenta para gerenciamento de classe de uso / Carlos Paulino, Carolina Gonçalves, Matheus Marabesi. - Santo André, 2016. – Quantidade de folhas f. 81

Monografia – Centro Universitário Fundação Santo André. Pós graduação Engenharia de Software
Orientador: Prof.º Ms. Rubens Salles

1.caso de uso 2.requisitos 3.documentação I. Gonçalves, Carolina II. Marabesi, Matheus

Candidato (a):
Título:

Trabalho de conclusão de curso apresentado como exigência parcial para obtenção do título de Especialista em Engenharia de Software ao Programa de Pós-Graduação do Centro Universitário Fundação Santo André, Curso de MBA em Engenharia de Software.

Data: ___/_____/_____

Prof(a) Dr(a): _____ _____
Instituição: Assinatura

Prof(a) Dr(a): _____ _____
Instituição: Assinatura

Prof(a) Dr(a): _____ _____
Instituição: Assinatura

Este trabalho é dedicado aos nossos familiares e amigos por todo apoio cedido durante a elaboração e execução do trabalho. Aos educadores, que com seu conhecimento e experiência nos guiaram e mostraram a melhor forma de abordar nossa proposta. E a todos que, direta ou indiretamente, colaboraram com desenvolvimento diário do trabalho.

Aos nossos amigos que contribuíram com materiais de pesquisa, que possibilitaram que nós conseguíssemos finalizar este trabalho.

RESUMO

Este trabalho descreve um estudo sobre a importância de uma análise aprofundada sobre requisitos, utilizando caso de uso e apresenta uma ferramenta de apoio à documentação de requisitos de software. A partir de uma documentação efetiva, o processo de gerenciamento dos requisitos para desenvolvimento de uma solução de software, torna-se mais produtivo, alinhando a expectativa do cliente com a solução apresentada. O desenvolvimento da ferramenta baseou-se em modelos disponíveis no mercado de software, somado a literatura existente sobre requisitos e caso de uso. Com esses dados devidamente cadastrados na ferramenta, a equipe de projeto terá dados para que os requisitos sejam estudados e avaliados.

Palavras-chave: Caso de uso, requisitos, documentação, software

ABSTRACT

This paper describes a study on the importance of a thorough analysis of requirements using use case and has a support tool for documenting software requirements. From an effective documentation, process management requirements for development of a software solution , becomes more productive , aligning the customer expectation with the proposed solution. The development tool based on models available in the software market , coupled with the existing literature on requirements and use case . With this data properly registered in the tool, the project team will have data so that the requirements are studied and evaluated.

Keywords: Use case, requirements, documentation, software

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

IIBA: International Institute of Business Analysis

NASA: National Aeronautics and Space Administration

OMT: Object Modeling Technique

OOSE: Object-Oriented Software Engineering

SADT: Structured Analysis and Design Technique

UML: Unified Modeling Language

FAQ: Frequently Asked Questions

SUMÁRIO

INTRODUÇÃO

Um dos grandes desafios no desenvolvimento de sistemas é compreender a necessidade do cliente e apresentar uma solução que de fato, o atenda. De acordo com a experiência da indústria, tempo e esforço insuficientes são gastos nas atividades de requisitos relacionados com o desenvolvimento do sistema. Outros problemas são derivados do descaso no acompanhamento, desde a definição até a evolução, dos requisitos durante o processo de concepção e construção do software.

Um software por mais planejado e codificado que seja, se não tiver uma compreensão dos requisitos, acabará por frustrar o cliente, apresentando uma solução que não está alinhada à sua expectativa.

A falta de entendimento de requisitos é eminente. Identificamos que com o caso de uso conseguimos unificar o entendimento de pessoas técnicas com o entendimento de não técnicos, tornando assim a passagem de conhecimento muito mais efetiva na hora de coletar os requisitos para o desenvolvimento de um software.

Existem quatro tipos de notações para a especificação de requisitos. A linguagem natural estruturada, linguagem de descrição de projeto, notações gráficas e especificações matemáticas. Para esse trabalho iremos utilizar a notação gráfica que Caso de Uso da UML[1] (Unified Modeling Language).

Segundo Ivan Jacobson, um caso de uso é “um documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo”. Apesar de apresentarmos os diagramas existentes na UML 2.0[2], nosso foco será o diagrama caso de uso, sendo que este foi o diagrama chave para o desenvolvimento da ferramenta de gerenciamento de requisitos.

A documentação de software poderia ser considerada o grande “calcanhar de aquiles[3]” dos desenvolvedores, mesmo sendo uma atividade extremamente essencial para o gerenciamento do sistema. Muitos projetos deixam de lograr êxito justamente por falta de documentação, o que implica em várias áreas, dificulta processos de manutenção nas aplicações, uma vez que a equipe que desenvolveu o software nem

[1] A Unified Modelling Language (UML) é uma linguagem ou notação de diagramas para especificar, visualizar e documentar modelos de 'software' orientada a objetos. Ela surgiu da fusão de três grandes métodos, do BOOCH, OMT (Rumbaugh) e OOSE (Jacobson).
[2] Atualmente a última versão da UML é a 2.5 de março de 2015 (fonte: http://www.omg.org/spec/UML/).
[3] Calcanhar de Aquiles é uma expressão popular que significa o ponto fraco de alguém. sua fraqueza, vulnerabilidade. A expressão também é aplicavél para organizações, projetos ou tarefas que precisam ser realizadas.

sempre continua na mesma empresa a longo prazo, correção de erros e pontos vulneráveis no sistema demandam mais tempo para serem encontrados e corrigidos.

Por fim, outro problema que as organizações enfrentam é o custo direcionado para investimento em projetos. O investimento médio da indústria no processo de requisitos para um típico sistema é de 2% para 3% do custo total do projeto. Evidencia-se que este montante de investimento é insuficiente e de fato é a causa raiz do fracasso de muitos projetos. Dados fornecidos pela Aeronautics and Space Administration dos EUA (NASA), fornecem uma mensagem clara e poderosa: organizações que utilizam em média de 2% a 3% do total do custo/esforço do projeto possui um aumento de volume de custos de 80% a 200%, enquanto os projetos que investiram de 8% a 14% do total do custo / esforço no processo de requisitos possuem de 0% a 50% de aumento de custo/esforço. Tendo ciência deste cenário a solução aqui proposta pretende ser oferecida para as organizações de forma gratuita, buscando proporcionar uma ferramenta que sem apresentar um custo financeiro, poderá gerar uma rentabilidade maior e melhor aproveitamento de recursos humanos.

Tendo como base estas informações, procuramos desenvolver um protótipo que possa auxiliar a documentação de caso de uso de forma ordenada. Na aplicação, o usuário terá o processo documentado, guiando-se de forma intuitiva por cada um dos menus que somados compreende todo o processo de caso de uso.

PROBLEMA

Fica claro que hoje na indústria não há planejamento e tempo despendido corretamente na análise de requisitos, e isso é um dos principais motivos pelos quais os projetos atrasam ou não são concluídos.

JUSTIFICATIVA

Otimizar o tempo de levantamento e elicitação de requisitos e garantir que as informações “não se percam”, por estarem armazenadas no sistema.

OBJETIVO GERAL

Aplicar os conhecimentos acadêmicos obtidos durante o curso em uma solução que auxilie uma atividade no desenvolvimento de software

OBJETIVO ESPECÍFICO

Desenvolver uma ferramenta para auxiliar o gerenciamento de requisitos diminuindo o custo/esforço total do projeto desenvolvido. Tal ferramenta irá auxiliar no gerenciamento de requisitos facilitando a manutenção dos requisitos, criação e entendimento, provendo um entendimento mútuo entre pessoas técnicas e não técnicas para o desenvolvimento de software de qualidade.

METODOLOGIA

Para a criação deste documento foram utilizadas as metodologias “pesquisa bibliográfica”, para a sustentação teórica escrita e “pesquisa experimental”, pois foi criada uma ferramenta com base nos conceitos aprendidos.

ESTRUTURA DA MONOGRAFIA

O primeiro capítulo analisa requisitos de sistema, detalhando suas variações (funcionais, não funcionais e domínios), explicando desde o levantamento até a documentação deste.

O capítulo dois aborda a UML, apresentando uma visão geral de seu significado, retratando alguns dos principais diagramas, dando ênfase, porém, no diagrama de caso de uso.

O terceiro capítulo retrata a solução proposta neste trabalho. Verifica-se aqui a análise do próprio protótipo em si, por meio da documentação do caso de uso do software, prototipação de telas e explicação das funcionalidades.

O último capítulo aborda as conclusões sobre o trabalho, resultados obtidos e sugestões de melhorias para futuras versões.

1. ENGENHARIA DE REQUISITOS

Pressman (2010) define que em uma perspectiva de processo de software, a engenharia de requisitos é uma grande ação de engenharia de software que começa durante a atividade de comunicação e continua na atividade de modelagem. Não só apenas para o processo, como de uma forma geral, engenharia de requisitos está presente em etapas fundamentais da construção do software.

> Engenharia de requisitos constrói uma ponte da concepção à construção. Mas, de onde é que a ponte se originou? Alguém poderia argumentar que ela começa a partir dos *stakeholders* [4](gerentes, clientes, usuários finais), onde a necessidade do negócio é definida, os cenários do usuário são descritos, funções e características são delineadas e restrições do projeto são identificadas. Outros poderiam sugerir que ela começa com uma definição mais ampla do sistema, onde o software é um componente que faz parte de um sistema maior de domínio. (PRESSMAN, 2010, p.120).

Para que um projeto de software seja executado com sucesso o primeiro passo é entender qual o propósito do software, quem irá utilizá-lo, quem são os envolvidos, qual a importância desse software, quanto irá custar e assim por diante, ou seja, é necessário planejar antes de executar.

> Engenharia de requisitos fornece o mecanismo apropriado para compreender o que o cliente quer, analisando a necessidade, avaliando a viabilidade, negociando uma solução razoável, especificando a solução de forma inequívoca, a validação da especificação e gerenciar os requisitos como eles são transformados em um sistema operacional. (PRESSMAN, 2010, p. 121)

1.1. Levantamento de Requisitos

Para que um sistema possa cumprir o objetivo com o qual foi desenvolvido, é necessária uma análise dos requisitos de software. A primeira resposta é associada com as funcionalidades que o sistema deve contemplar ou mesmo restrições, conforme representado na figura 1.

> O termo requisito não é utilizado pela indústria de software de modo consistente. Em alguns casos, um requisito é visto como uma declaração abstrata, de alto nível, de uma função que o sistema deve fornecer ou de uma restrição do sistema. No outro extremo, ele é definição detalhada, matematicamente formal, de uma função do sistema. (SOMMERVILLE, 2004, p.82).

[4] São pessoas, grupos ou organizações que podem impactar ou serem impactados por decisões, atividades ou entregas do projeto (Guia PMBOK®, 5ª Edição, página 390). Em outras palavras, todos os indivíduos ou organizações que possuem algum tipo de relacionamento com a execução ou os objetivos do projeto.

Segundo Pressman (2010), entender os requisitos de um problema é uma das tarefas mais difíceis que um engenheiro de software poderá enfrentar.

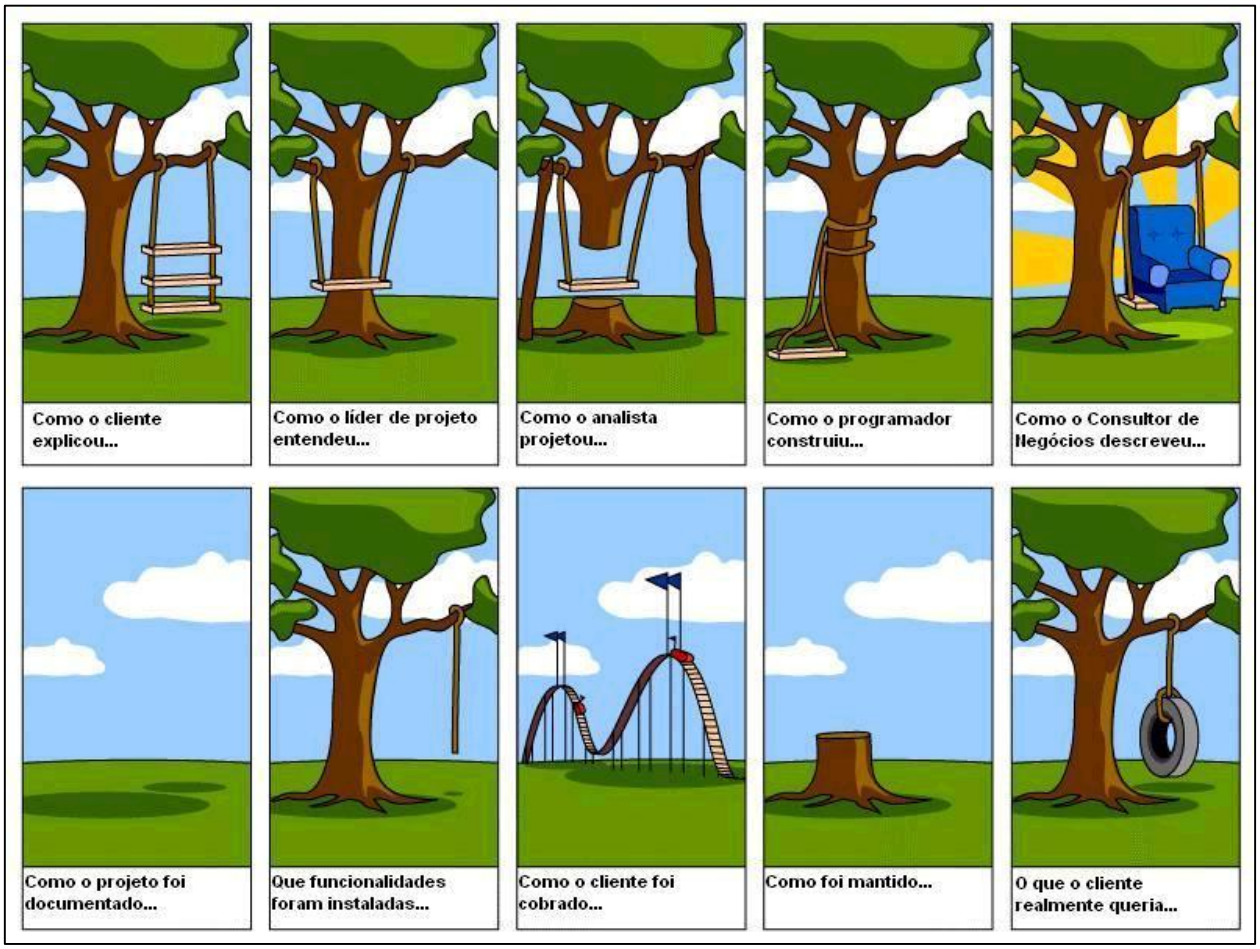

Figura 1 - Problemas com o escopo
(Fonte: scaryideas.com)

Podemos ainda enumerar alguns tópicos raiz que dificultam o entendimento dos requisitos, segundo Pressman (2010):

> "**Problema de escopo**. As fronteiras do sistema não são bem definidas ou o cliente/usuário especifica detalhes técnicos desnecessários que podem mais confundir do que clarear, os objetivos do sistema.
> **Problema de entendimento**. O cliente/usuário não tem certeza do que é necessário, possuindo pouco entendimento sobre a capacidade ou limitações do seu ambiente computacional, não possui um entendimento completo sobre o domínio, possui problemas em comunicar as necessidades para o engenheiro, omite informação que acredita ser "óbvia", especifica requerimentos que conflitam com as necessidades dos clientes/usuários ou especifica requerimentos que são ambíguos ou não testáveis.
> **Problemas de volatilidade**. Os requerimentos mudam ao passar do tempo". (PRESSMAN, 2010, p. 121).

Podemos dizer que um conjunto de requisitos, como um todo, representa um acordo negociado entre todas as partes interessadas no sistema, satisfazendo os

objetivos deste. Os requisitos são classificados em três categorias: requisitos funcionais, requisitos não funcionais e requisitos de domínio.

1.1.1 Requisitos Funcionais

Os requisitos funcionais são as declarações das funcionalidades e serviços que o sistema deve fazer, como o sistema deve reagir, as entradas específicas e como o sistema deve se comportar em determinadas situações. Há situações em que os requisitos funcionais poderão especificar também o que o sistema não deve fazer.

> [...] a especificação de requisitos funcionais de um sistema deve ser completa e consistente. Completeza significa que todos os serviços exigidos pelo usuário devem ser definidos. Consistência significa que os requisitos não devem ter definições contraditórias (SOMMERVILLE, 2008, p.82).

1.1.2 Requisitos Não Funcionais

Requisitos não funcionais, como o nome sugere, são aqueles que não dizem respeito diretamente as funções específicas fornecidas pelo sistema, conforme representado na figura 2.

> Requisitos não funcionais surgem conforme a necessidade dos usuários, em razão de restrições de orçamento, de políticas organizacionais, pela necessidade de interoperabilidade com outros sistemas de software ou hardware ou devido a fatores externos, como por exemplo regulamentos de segurança e legislação sobre privacidade. (SOMMERVILLE, 2008, p.85)

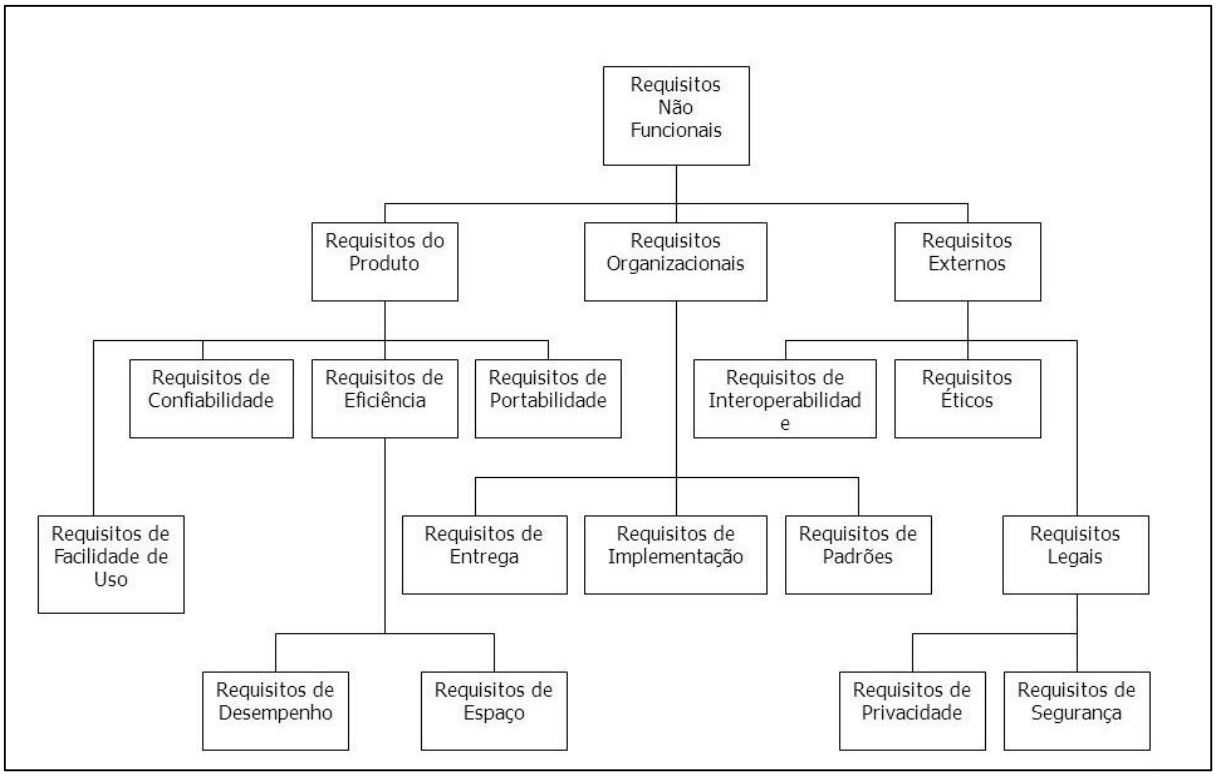

Figura 2 - Requisitos não funcionais
(Fonte: Engenharia de Software, 2004)

Segundo o IIBA[5] (2011) requisitos de produtos especificam o comportamento do produto. Toda comunicação necessária entre o ambiente X e o usuário deve ser mencionada nesses requisitos. Podemos descrever cada tipo de requisito não funcional como:

1.1.2.1. Confiabilidade

Esse tipo de requisito trata sobre a habilidade do aplicativo de manter-se disponível e se recuperar de possíveis falhas.

1.1.2.2. Eficiência de Desempenho

Esse requisito descreve sobre o tempo necessário para executar as atividades, além dos níveis de utilização de recursos. Dentro de requisitos de “eficiência de desempenho”, podemos ter duas subcategorias: desempenho e espaço.

[5] O IIBA (International Institute of Business Analysis) foi fundado em Toronto, Canadá, em outubro de 2003, para apoiar a comunidade de análise de negócios por meio do *Guia BABOK®*. Já no Brasil, o Capítulo São Paulo foi criado em 2008 com o objetivo de trazer para o país as iniciativas do IIBA.

1.1.2.2.1. Desempenho

Associados à eficiência, uso de recursos e tempo de resposta da aplicação.

1.1.2.2.2. Espaço

Está atrelado a capacidade em que o hardware e software precisarão para que o sistema seja colocado em funcionamento sem problemas.

1.1.2.3. Portabilidade

Os requisitos de portabilidade incluem a facilidade de instalação e de desinstalação de aplicativos, além dos tipos diferentes de ambientes nos quais pode rodar e a facilidade de migração para um novo ambiente.

1.1.2.4. Facilidade de Uso

Associados à facilidade de uso da aplicação como, por exemplo, funcionamento de navegadores, teclas de atalho, etc. Já os “requisitos organizacionais” referem-se a políticas e procedimentos nas organizações do cliente e do desenvolvedor. Definem o processo de desenvolvimento de sistema e os documentos a serem entregues deverão estar de acordo com o processo e os produtos a serem entregues definidos. Podemos descrever cada tipo de requisito como: entrega, implementação e padrão.

1.1.2.4.1. Entrega

Descreve sobre prazos de entrega de uma determinada rotina ou ação.

1.1.2.4.2. Implementação

Alterar um componente sem afetar os demais, a habilidade de reutilizar componentes, se o aplicativo pode ser testado com eficácia e seus problemas podem

ser propriamente diagnosticados, a facilidade de fazer mudanças e a habilidade de interagir com outros sistemas.

1.1.2.4.3. Padrões

Informa sobre a padronização do uso de linguagens de programação orientadas ao objeto, além de utilização de padrões para documentação e plataformas de software. Por fim, os “requisitos externos” referem-se a fatores externos ao sistema e ao seu processo de desenvolvimento. O sistema não deverá revelar aos operadores nenhuma informação pessoal sobre os clientes. Podemos descrever cada tipo de requisito como: interoperabilidade, éticos e legais.

1.1.2.4.3.1. Interoperabilidade

Esse requisito deve existir caso o sistema tenha que se comunicar com o banco de dados ou qualquer outro sistema.

1.1.2.4.3.2. Éticos

São requisitos de sistema que não apresentam aos usuários quaisquer dados de cunho privativo.

1.1.2.4.3.3. Legais

Tratam sobre os requisitos de auditoria e privacidade do sistema.

1.2.1.1. Medição

Segundo o IIBA (2011) um requisito não-funcional deve possuir uma medida apropriada para que possa ser adequadamente testada. Alguns requisitos não-funcionais podem parecer muito subjetivos, mas uma reflexão cuidadosa geralmente pode encontrar uma medida de sucesso apropriada.

1.3.1.1. Documentação

Os requisitos não funcionais devem ser apresentados como parte da documentação total de requisitos, geralmente em um item separado aos demais. A vantagem de realizar uma documentação apropriada de requisitos não-funcionais, é que esses terão uma forte influência na aceitação, ou não, pelos usuários do sistema.

Embora muito importantes, esse tipo de requisito é mais trabalhoso e moroso, podendo deixar o levantamento de requisitos demorado, uma vez que essas "necessidades" muitas vezes não estão claras para o usuário solicitante.

1.1.3. Requisitos de Domínio

Segundo Sommerville (2008) assim como os requisitos não funcionais, os requisitos de domínio são aqueles que não dizem respeito diretamente as funções fornecidas pelo sistema, porém requisitos de domínio podem se transformar rapidamente em requisitos funcionais ou não funcionais.

> "Os requisitos de domínio são derivados do domínio da aplicação do sistema, em vez de serem obtidos a partir das necessidades específicas dos usuários do sistema. [...] os requisitos de domínio são importantes porque, muitas vezes, refletem fundamentos do domínio da aplicação. Se esses requisitos não forem satisfeitos, poderá ser impossível fazer o sistema operar satisfatoriamente". (SOMMERVILLE, 2008, p.88)

A lógica envolvendo o requisito é importante, pois não basta apenas que ocorra uma transcrição para a programação enquanto não se questiona o sentido da inclusão, alteração ou exclusão de determinado requisito. Entender a lógica que rege os requisitos, auxilia sua manutenção e proporciona uma melhor avaliação para impactos de mudança.

1.2. Elicitação de Requisitos

Segundo Fernandes (2012) a elicitação de requisitos deve garantir que os mesmos sejam completos, claros, corretos e consistentes, incluindo questões latentes e subjacentes, explicar informações e respostas necessárias. A elicitação de requisitos possui algumas técnicas bem definidas para auxiliar na obtenção dos requisitos. Pressman define e explica a elicitação de requisitos:

> Elicitação de requisitos (também chamado de levantamento de requisitos) combina elementos de resolução de problemas, elaboração, negociação e

> especificação. A fim de incentivar, uma abordagem orientada a equipe colaborativa para coleta de requisitos, as partes interessadas trabalham em conjunto para identificar o problema, propor elementos da solução, negociar diferentes abordagens e especificar um conjunto preliminar de requisitos da solução. (PRESSMAN, 2010, p.128)

Normalmente requisitos são definidos ao longo das fases de elicitação (análise, verificação e validação). Requisitos podem ser elicitados em entrevistas ou *wokshops*[6] de requisitos, sendo que, posteriormente, esses requisitos são usados para construir e verificar modelos, onde podem ser encontrados gaps (lacunas) que irão requerer a elicitação dos detalhes dos requisitos anteriormente identificados.

Uma combinação de técnicas de elicitação é normalmente utilizada para examinar e definir os requisitos de forma completa. Uma variedade de fatores define quais deverão ser usadas, conforme mostra a figura abaixo:

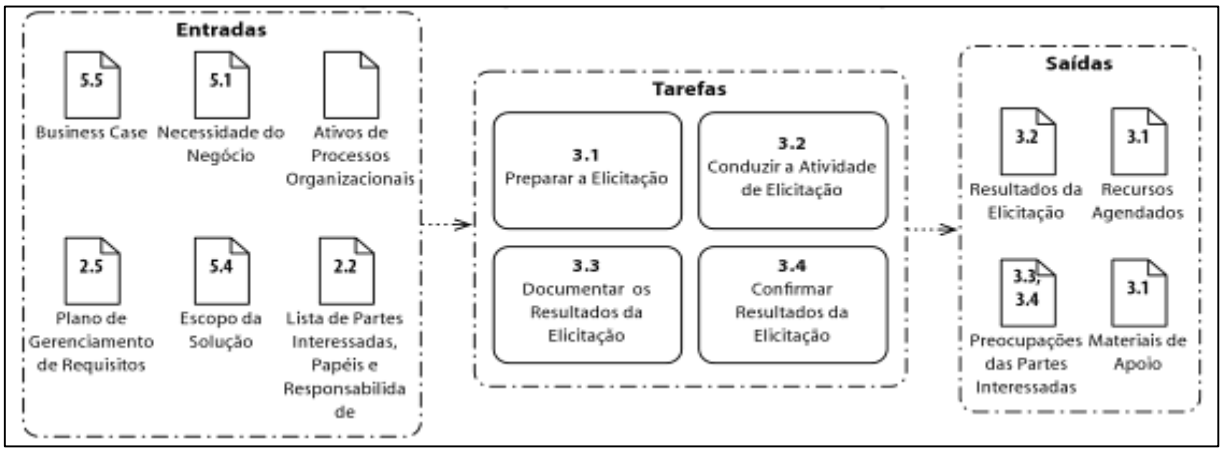

Figura 3 - Diagrama de Entrada/Saída da Elicitação
(Fonte: *Guia* BABOK® Versão 2.0)

Segundo Fernandes (2012), essa tarefa de elicitação de requisitos define tarefas relacionadas a planejamento e ao monitoramento das atividades de análise de negócios.

1.2.1. Planejar a abordagem de análise de negócios

Resume-se em selecionar a melhor maneira para uma organização realizar a análise de negócios, definindo uma metodologia, ferramentas de análise e gerenciamento de requisitos.

---

[6] Um workshop diferencia-se de uma palestra por alguns eixos conceituais básicos. Nele, a plateia não é apenas mera espectadora.

1.2.2. Conduzir análise das partes interessadas

Determinar a influência e autoridade dos envolvidos para a validação e aprovação dos requisitos;

1.2.3. Planejar as atividades de análise de negócios

Identificar o escopo e atividades dentro de um cronograma e estimativa de tempo**;**

1.2.4. Planejar a comunicação da análise de negócios

Estabelecer um plano de comunicação com as atividades para o gerenciamento das reuniões de trabalho, reportes, validações, etc.

1.2.5. Planejar o processo de gerenciamento de requisitos

Definir como e quem aprovará os requisitos, além do gerenciamento de mudanças.

1.2.6. Gerenciar desempenho da análise de negócios

Garantir que as atividades de análise do negócio sejam executadas com eficácia.

1.3. Preparar a Elicitação de Requisitos

Segundo Pressman (2010) a preparação de uma elicitação de requisitos tem como objetivo garantir que todos recursos necessários estejam organizados e agendados para a condução das atividades de elicitação. São fases da preparação de elicitação de requisitos “Business Case”, “Necessidade do Negócio”, “Escopo da Solução” e “Lista de Partes Interessadas, Papéis e Responsabilidades”, conforme ilustrado na figura abaixo.

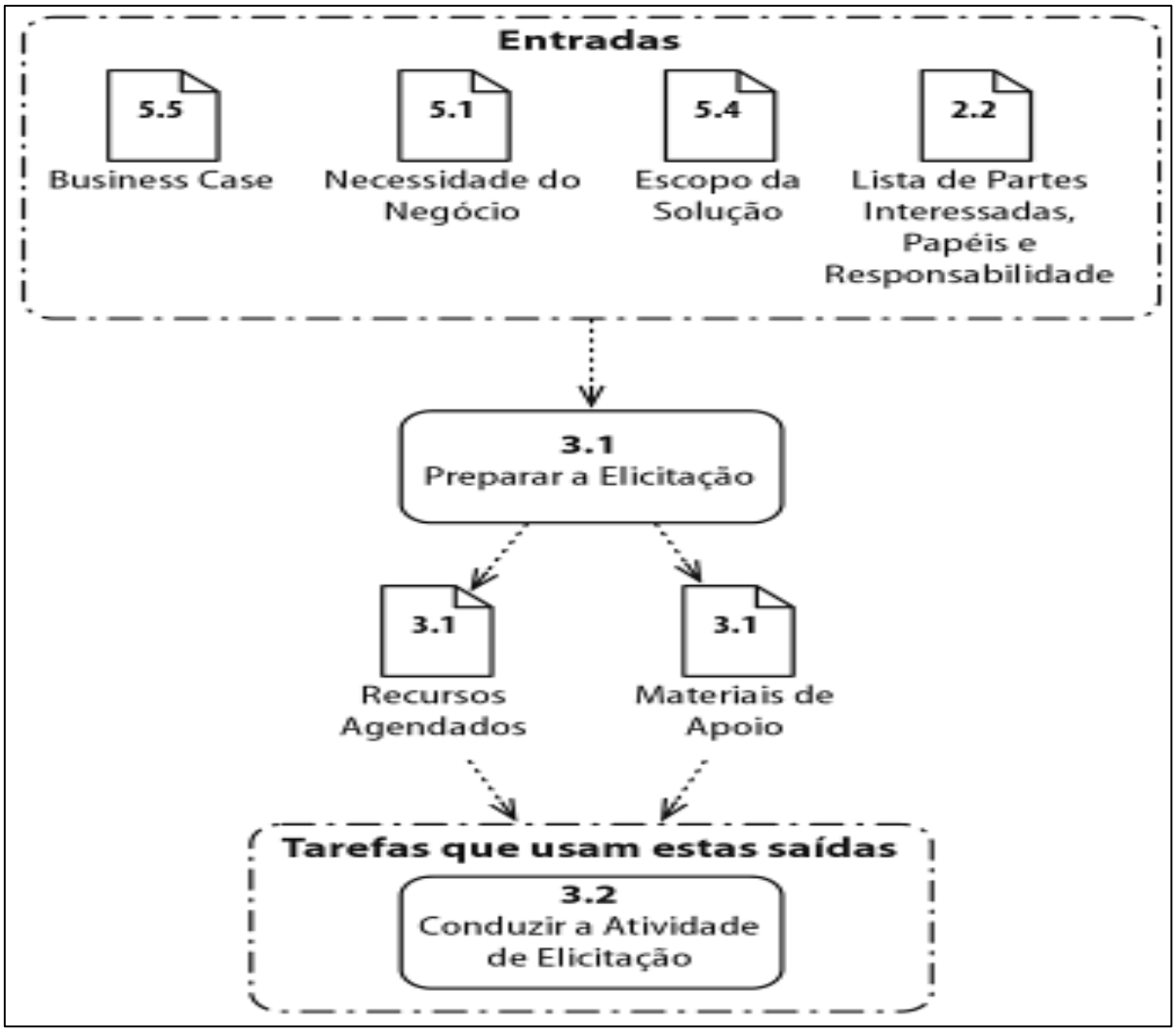

Figura 4 - Diagrama de entrada/saída de preparar a Elicitação
(Fonte: *Guia* BABOK® Versão 2.0)

1.3.1. Entradas

1.3.1.1. Necessidade do Negócio

Essa etapa é necessária para garantir que o analista de negócios compreenda quais informações devem ser elicitadas junto às partes interessadas. Esta entrada é usada ao longo da elicitação dos requisitos de negócio (com exceção da necessidade do negócio em si).

1.3.1.2. Escopo da Solução e Business Case[7]

[7] Business Case (Caso de negócio) fornece as informações necessárias do ponto de vista de um negócio, para determinar se vale ou não a pena investir no projeto.

Necessários para garantir que o analista de negócios compreenda quais informações devem ser elicitadas junto às partes interessadas. Estas entradas são usadas no levantamento dos requisitos das partes interessadas, da solução e da transação.

1.3.1.3. Lista de Partes Interessadas, Papéis e Atribuições de Responsabilidades

Usada para identificar as partes interessadas que devem participar nas atividades de elicitação.

1.3.2. Elementos

Esclarecer o escopo específico da técnica de elicitação escolhida e agrupar todos os materiais de apoio necessários. Agendar todos os recursos (pessoas, instalações e equipamento), além de notificar as partes apropriadas do plano.

1.3.3. Técnicas

Brainstorming[8], Análise documental, Grupos Focais, Análise de Interfaces, Entrevistas, Observação, Prototipagem, *Workshops* de Requisitos e Pesquisa/Questionário.

1.3.4. Partes Interessadas

1.3.4.1. Todas as partes interessadas

Dependendo dos requisitos da atividade de elicitação, qualquer parte interessada pode ser um participante.

1.3.4.2. Gerente de Projetos

[8] Brainstorming é uma técnica criada pelo americano Osborn em 1963 utilizada para auxiliar um grupo de pessoas a criar o máximo de ideias no menor tempo possível. Geralmente traduz-se para o português como “tempestade cerebral” ou “tempestade de ideias”.

O gerente do projeto auxiliará garantindo que os recursos necessários estão disponíveis.

1.3.5. Saídas

1.3.5.1. Recursos Agendados

Isso inclui os participantes, o local no qual a atividade de elicitação ocorrerá e quaisquer outros recursos que possam ser necessários.

1.3.5.2. Materiais de Apoio

Quaisquer materiais necessários para auxiliar na explicação das técnicas usadas ou na sua execução.

1.4. **Conduzir a atividade de elicitar requisitos**

1.4.1. Propósito

Se reunir com as partes interessadas para elicitar informações referentes às necessidades. O evento de elicitação acontece (brainstorming, grupos focais, entrevistas, observação, prototipagem, workshops de requisitos) ou a elicitação é executada (análise documental, análise de interfaces) ou é distribuída (pesquisas, questionários).

1.4.2. Entrada

1.4.2.1. Necessidade do Negócio

Necessária para garantir que o analista de negócios compreenda qual informação deve ser elicitada junto às partes interessadas. Esta entrada é usada ao longo da elicitação dos requisitos do negócio (com exceção da necessidade do negócio em si).

1.4.2.2. Ativos de Processos Organizacionais

Podem incluir modelos ou processos para estas atividades.

1.4.2.3. Plano de Gerenciamento dos Requisitos

Determinar qual informação precisa ser registrada e rastreada como resultado da atividade. Em particular, muitos atributos dos requisitos podem ser elicitados e capturados enquanto se executa essa tarefa.

1.4.2.4. Recursos Agendados

As partes interessadas relevantes, localização e outros recursos devem estar disponíveis.

1.4.2.5. Escopo da Solução Business Case

São necessários para garantir que o analista de negócios compreenda qual informação deve ser elicitada junto às partes interessadas.

1.4.2.6. Materiais de Apoio

Quadros brancos, *flipcharts*, documentos e outros materiais devem estar disponíveis quando a atividade estiver sendo conduzida.

1.4.3. Elementos

1.4.3.1. Rastrear Requisitos

Durante o levantamento dos requisitos é importante evitar o desvio do escopo.

1.4.3.2. Capturar os Atributos dos Requisitos

Documentar os atributos dos requisitos durante a elicitação, como a sua origem, valor e prioridade, auxiliará no gerenciamento de cada requisito ao longo do seu ciclo de vida.

1.4.3.3. Métricas

Acompanhar os participantes da elicitação e o tempo real investido na elicitação dos requisitos provem uma base para planejamento futuro.

1.4.4 Partes Interessadas

São usuários interessados na condução de elicitação de requisitos, do cliente, especialista no assunto, usuário final, fornecedor, patrocinador, especialista na implementação da solução, suporte operacional, gerente de projeto e testador.

1.4.5 Saída

Pode-se concluir desta fase que será gerada uma documentação apropriada para a técnica que deverá ser utilizada no levantamento.

2. UML

2.1. Conceitos Básicos

Guedes (2011) define que a UML - Unified Modeling Language ou Linguagem de Modelagem Unificada - é uma linguagem visual utilizada para modelar softwares baseados no paradigma de orientação a objetos. É uma linguagem de propósito geral que pode ser aplicada a todos os domínios de aplicação. Deve ficar bem claro, porém, que a UML não é uma linguagem de programação, e sim uma linguagem de modelagem, uma notação, cujo objetivo é auxiliar os engenheiros de software a definirem as características do sistema, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e até mesmo sua necessidade física em relação ao equipamento sobre o qual o sistema deverá ser implantado.

A UML surgiu da união de três métodos de modelagem: o método de Booch[9], o método OMT[10] (Object Modeling Technique) de Jacobson, e o método OOSE[11] (Object-Oriented Software Engineering) de Rumbaugh.

"A Unified Modeling Language (UML) é uma linguagem de propósito geral de modelagem visual que é usado para especificar, visualizar, construir e documentar os artefatos de um software". (Rumbaugh, 1998, p.03)

2.2. Linguagem de Modelagem

A UML é uma linguagem de modelagem que nos permite modelar sistemas orientados a objetos. Mas o que podemos entender por modelagem?

Hamilton e Miles definem que uma linguagem de modelagem é:

> "[...] feita de pseudo-código, código real, imagens, diagramas, ou longa descrição; na verdade, é praticamente tudo o que o ajuda a descrever o seu sistema. Os elementos que compõem uma linguagem de modelagem são chamados de notação. [...] Uma linguagem de modelagem pode ser qualquer coisa que contém uma notação (uma forma de expressar o modelo) e uma

[9] A metodologia Booch concentra-se em fase de análise e design e não considera a implementação ou a fase de testes em grande detalhe.
[10] Todo o processo de desenvolvimento de software OMT tem quatro fases: Análise, design de sistemas, design de objeto, e implementação do software. A maior parte da modelagem é realizada na fase de análise.
[11] OOSE

> descrição do que a notação significa (um meta-modelo)". (Hamilton; Miles, 2006, p.15)

2.3. Diagramas

Um diagrama é uma representação gráfica de um conjunto de elementos, geralmente representada como um gráfico conectado de vértices (itens) e arcos (relacionamentos) projetado sob determinada perspectiva em um sistema. A variedade de diagramas que compõem a UML tem por objetivo proporcionar uma visão mais ampla do sistema, ora focando na parte externa, ora na parte interna.

> Cada diagrama da UML analisa o sistema, ou parte dele, sob uma determinada ótica. É como se o sistema fosse modelado em camadas, sendo que alguns diagramas enfocam o sistema de forma mais geral, apresentando uma visão externa do sistema, como é o objetivo do Diagrama de Casos de Uso, enquanto outros oferecem uma visão de uma camada mais profunda do software, apresentando um enfoque mais técnico ou ainda visualizando apenas uma característica específica do sistema ou um determinado processo. A utilização de diversos diagramas permite que falhas sejam descobertas, diminuindo a possibilidade da ocorrência de erros futuros. (Guedes, 2010, p.18)

A seguir apresentaremos um resumo de cada um destes diagramas, destacando suas principais características, dando enfoque principalmente no diagrama de caso de uso.

2.3.1. Diagrama de Classes

Diagrama estrutural que mostra um conjunto de classes, interfaces e seus relacionamentos, conforme figura 5.

> Esse diagrama apresenta uma visão estática de como as classes estão organizadas. Preocupando-se em como definir a estrutura lógica das mesmas. [...] basicamente, o diagrama de classes é composto por suas classes e suas associações existentes entre elas, ou seja, os relacionamentos entre as classes. (Guedes, 2010, p.101)

Figura 5 - Representação de classe
(Fonte: elaborado por Matheus Marabesi)

> [...] uma classe, na linguagem UML é, representada como um retângulo com até três divisões, descrita a seguir:
> A primeira contém a descrição ou nome da classe. A segunda armazena os atributos e seus tipos de dados (o formato que os dados devem ter para serem armazenados em um atributo). Finalmente, a terceira divisão lista os métodos da classe. (Guedes, 2010, p.102).

A anatomia de um diagrama, o diagrama de classe possuí níveis de visibilidade em seus atributos e métodos.

“Os símbolos de sustenido (#) e mais (+) na frente dos atributos e métodos representam a visibilidade dos mesmos, o que determina quais objetos de quais classes podem utilizar o atributo ou o método em questão”. (Guedes, 2010, p.103)

Existem quatro tipos de visibilidade definidas para serem utilizadas no diagrama de classe.

“A visibilidade é utilizada para indicar o nível de acessibilidade de um determinado atributo ou método, sendo representada a esquerda destes”. (Guedes, 2010, p.47)

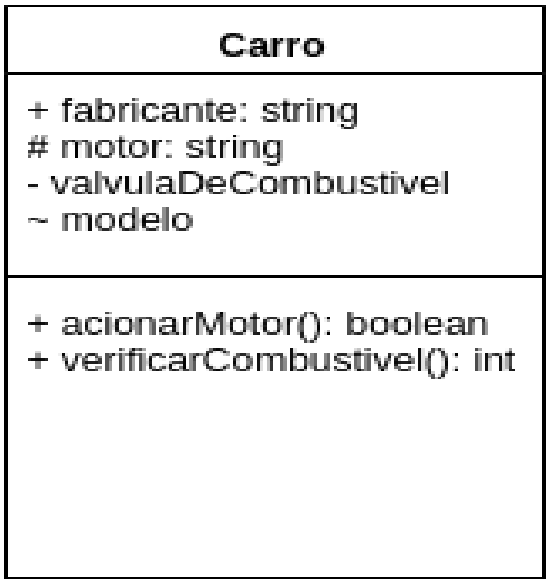

Figura 6 - Visibilidade da classe
(Fonte: elaborado por Matheus Marabesi)

Na orientação a objetos, o conceito de visibilidade é muito bem definido, como ilustra a figura 6 acima.

> A visibilidade é utilizada para indicar o nível de acessibilidade de um determinado método ou atributo. A visibilidade privada é representada por um símbolo de menos (-) e significa que somente objetos da classe detentora do atributo ou método poderão enxergá-lo ou utiliza-lo. A visibilidade protegida é representada pelo símbolo de sustenido (#) e determina que além dos objetos da classe detentora do atributo ou método também os objetos de suas subclasses poderão ter acesso ao mesmo. A visibilidade pública é representada por um símbolo de mais (+) e determina que o atributo ou método pode ser utilizado por qualquer objeto. A visibilidade pacote é representada por um símbolo de til (~) determina que o atributo ou método é visível por qualquer objeto dentro do pacote. (Guedes, 2010, p.47)

Atributos ou propriedades (figura 7) são utilizados para definir características de uma classe, e não o comportamento, como por exemplo cor, tamanho e largura, isso são características (propriedades) de um objeto.

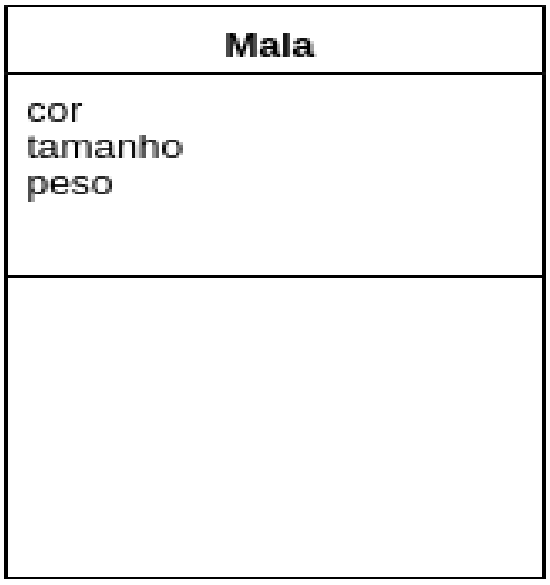

Figura 7 - Atributos de uma classe
(Fonte: elaborado por Matheus Marabesi)

Na orientação a objetos utilizamos métodos (figura 8) para definir comportamentos e utilizamos o mesmo conceito para o diagrama de classes na UML. Andar, ligar ou mover são comportamentos, ou seja, devemos tomar alguma ação para realizar um objetivo.

Figura 8 - Exemplo de métodos de uma classe
(Fonte: elaborado por Matheus Marabesi)

Podemos utilizar relações em nossos diagramas de classe para representar o nível de relacionamento e o seu tipo. “As classes não vivem em um vácuo, elas trabalham em conjunto usando diferentes tipos de relações. Relações entre classes vem em diferentes tipos [..]” (HAMILTON; MILES, 2006, p.122)

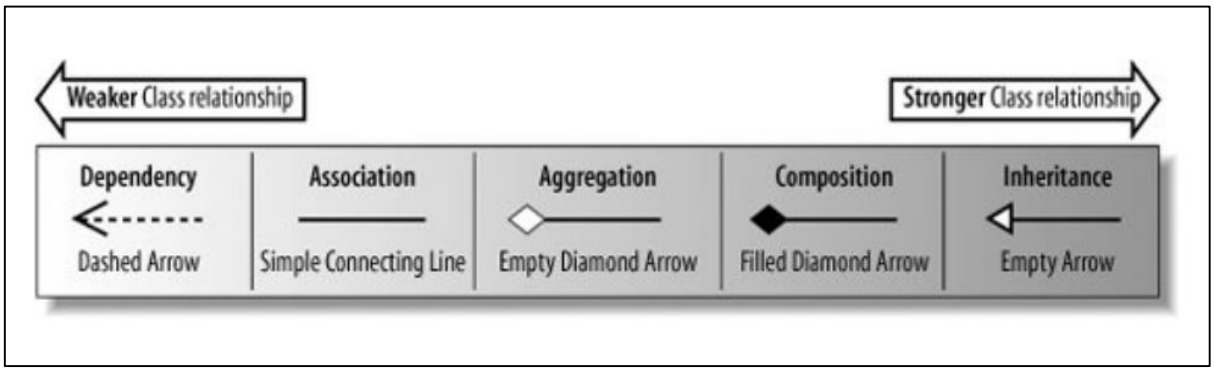

Figura 9 - Representação dos tipos de associação no diagrama de classes
(Fonte: UML 2.0 – Uma abordagem prática)

Podemos observar na figura 9 que quanto mais para esquerda mais fraca a relação entre as classes e consecutivamente quanto mais para a direita mais forte a relação entre as classes. De acordo com Hamilton e Miles (2006) podemos considerar no quadrante totalmente a esquerda *(Dependency*) objetos que trabalham com objetos de outra classe brevemente, já no quadrante seguinte (Association) podemos considerar objetos que trabalham com objetos de outras classes por um período de tempo prolongado. A partir do quadrante Agregação (*Aggregation*) possuimos referências entre objetos, na própria agregação temos uma classe que é a dona mas compartilha a referência com outras classes. O quadrante Composição (*Composition)*, quando uma classe possui um objeto de outra classe tornando assim a referência ainda mais forte, e finalmente chegamos ao quadrante Herança (*Inheritance*) onde representamos uma classe que é do tipo de uma outra classe.

2.3.1.1. Relação de dependência

O primeiro tipo de relação que podemos ter é o de dependência, o que significa que uma classe precisa saber de uma outra classe para ser utilizada, conforme mostra a figura 10.

“Esse relacionamento como o próprio nome já diz, identifica certo grau de dependência de uma classe em relação a outra”. (Guedes, 2010, p.116)

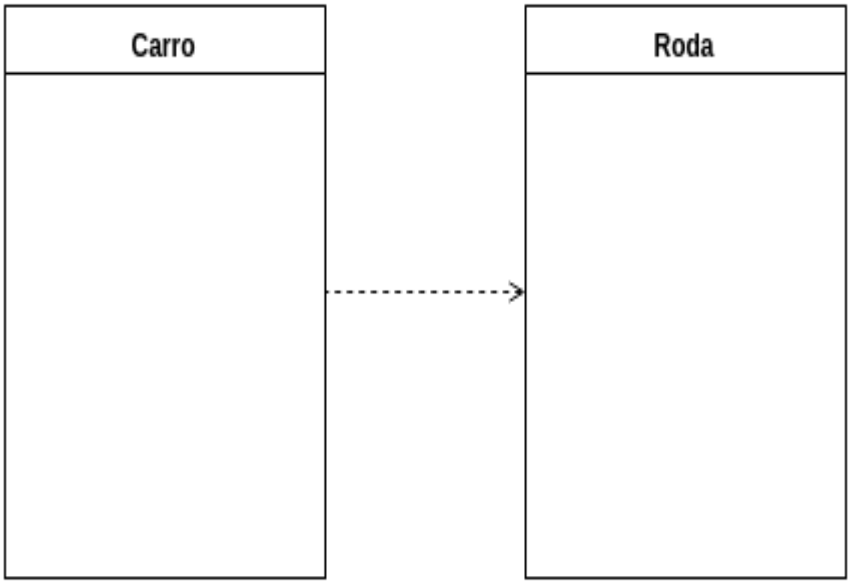

Figura 10 - Exemplo de dependência de classes
(Fonte: elaborado por Matheus Marabesi)

2.3.1.2. Associação

Com o diagrama de classe podemos representar quando uma classe está associada diretamente com outra classe, o que significa que a classe base irá conter uma referência para a classe associada, conforme mostra a figura 11.

“As associações são representadas por linhas ligando as classes envolvidas. Essas linhas podem ter nomes ou títulos para auxiliar a compreensão do tipo de vínculo estabelecido entre os objetos das classes envolvidas nas associações”. (Guedes, 2010, p.106)

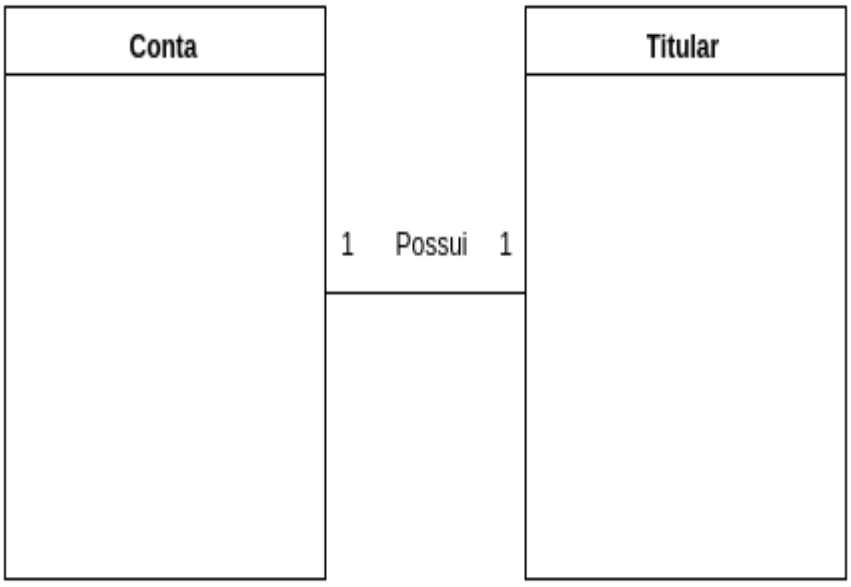

Figura 11 - Associação entre duas classes
(Fonte: elaborado por Matheus Marabesi)

2.3.1.3. Agregação

Com a agregação podemos representar classes relacionadas como um todo (figura 12).

“A agregação é um tipo especial de associação onde se tenta demonstrar que as informações de um objeto (chamado objeto-todo) precisam ser complementadas pelas informações contidas em um ou mais objetos de outra classe (chamados objetos-parte)”. (Guedes, 2010, p.111).

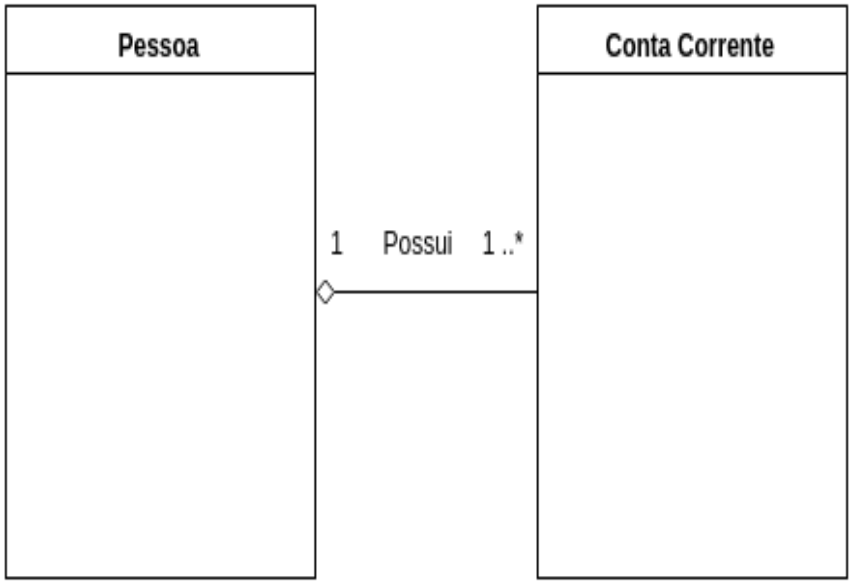

Figura 12 - Exemplo de uma classe com relação de agregação
(Fonte: elaborado por Matheus Marabesi)

2.3.1.4. Composição

Utilizando composição enfatizamos o relacionamento entre objeto-todo e objeto-parte. Em uma composição os objetos-parte não podem ser destruídos por um objeto diferente do objeto-todo”, conforme figura abaixo.

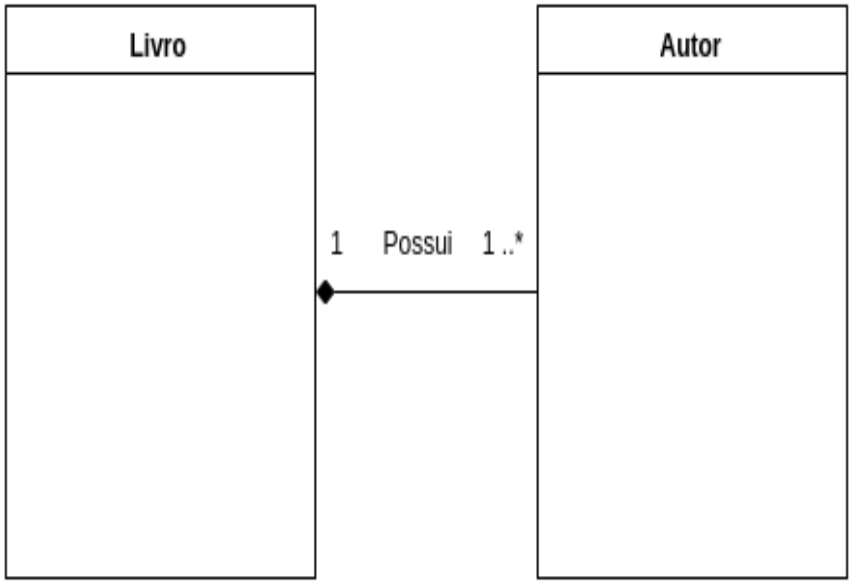

Figura 13 - Exemplo de uma classe com relacionamento de composição
(Fonte: elaborado por Matheus Marabesi)

2.3.1.5. Herança

Na programação orientada a objetos temos a associação herança (figura 14), assim como na UML, que pode representar que o tipo d uma classe é de outro tipo, ou podemos nos referenciar como generalização/especialização.

“O objetivo dessa associação é representar a ocorrência de herança entre as classes, identificando as classes-mãe (ou superclasses), chamadas gerais, e classes-filhas (ou sub-classes), chamadas especializadas”. (Guedes, 2010, p.113)

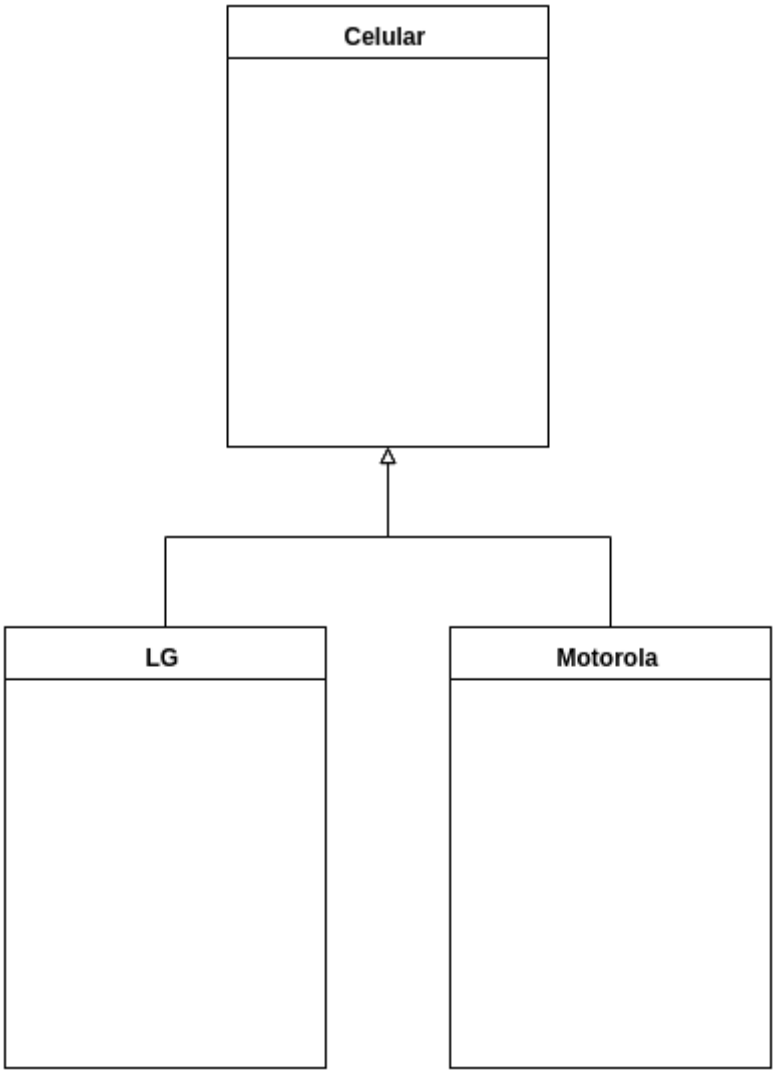

Figura 14 - Exemplo de associação de herança entre classes
(Fonte: elaborado por Matheus Marabesi)

2.3.2. Diagrama de Objetos

Diagrama estrutural que mostra um conjunto de objetos e seus relacionamentos, conforme ilustra a figura 15. Não mostram a evolução do sistema com o tempo. Estes diagramas possuem objetos e vínculos. O que distingue este diagrama dos demais é o seu conteúdo particular. Graficamente é representado por uma coleção de vértices e de arcos. Guedes ilustra com objetividade o que é o diagrama de objetos.

“O diagrama de objetos tem como objetivo fornece uma visão dos valores armazenados pelos objetos das classes, definidas no diagrama de classes, em um determinado momento do sistema”. (Guedes, 2010, p.183)

Para criar um diagrama de objetos geralmente utilizamos como base o diagrama de classe (figura 15), demonstrando assim o estado de uma determinada

classe em um dado momento do sistema. As semelhanças entre os diagramas são muitas, porém Guedes descreve com detalhes o diagrama de objeto.

“O nome do objeto, com todas as letras minúsculas, seguido do símbolo de dois pontos (:) e o nome da classe à qual o objeto pertence, com as letras iniciais maiúsculas”. (Guedes, 2010, p.183)

Figura 15 - Representação de um objeto
(Fonte: elaborado por Matheus Marabesi)

Podemos representar vínculos entre objetos, ou seja, podemos representar os vínculos de um objeto que estão associadas a uma classe (figura 16).

“Esses vínculos nada mais são do que instâncias das associações entre as classes representadas no diagrama de classes, assim como os objetos são instâncias da própria classe”. (Guedes, 2010, p.184)

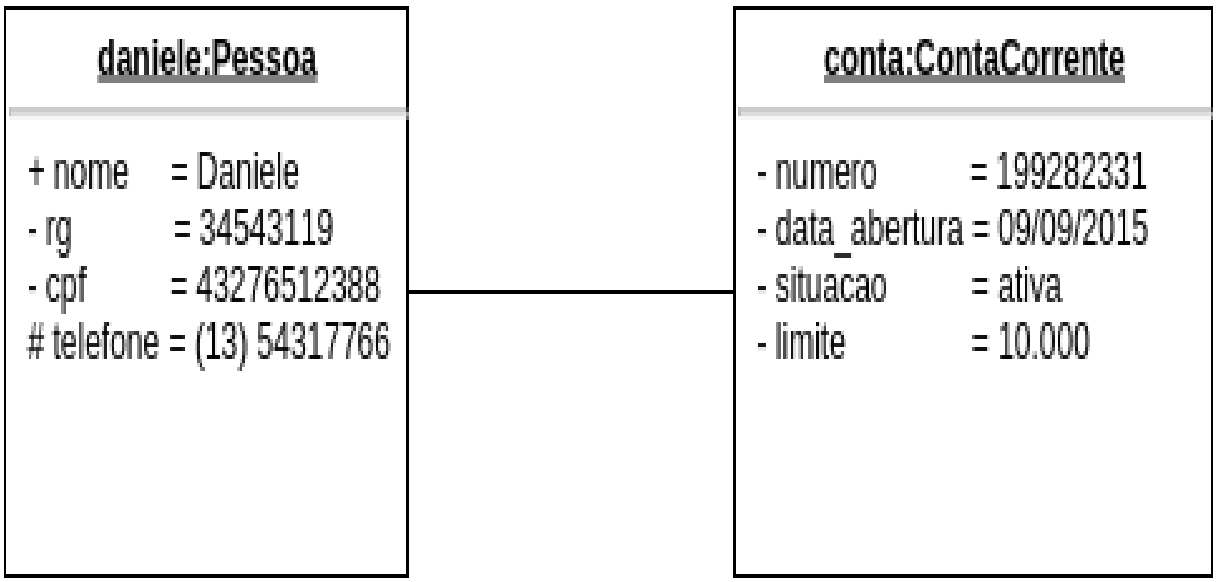

Figura 16 - Representação de vínculo entre objetos
(Fonte: elaborado por Matheus Marabesi)

2.3.3. Diagrama de Pacotes

Diagramas de pacote nos ajudam a manter a organização em nosso projeto, mesmo ele sendo pequeno, utilizando pacotes fornecemos uma forma de crescimento do projeto muito confortável, conforme mostra a figura 17.

> Se você está criando um pequeno programa (contendo apenas algumas classes), você não deve se preocupar em utilizar pacotes. Conforme o seu programa cresce e você adiciona desenvolvedores ao seu projeto, pacotes introduzem a estrutura e deixam você saber quem está trabalhando. (HAMILTON, 2006, p.289)

Diagrama estrutural que mostra a organização do modelo em pacotes. Entende-se por pacotes o agrupamento de um conjunto de elementos podendo ser estas classes, diagramas, interfaces ou até mesmo outros pacotes. Sua representação gráfica é uma pasta com uma guia.

“O diagrama de pacotes descreve como os elementos do modelo estão organizados em pacotes e demonstra as dependências entre eles. [...] Pode ser utilizado também para representar um conjunto de sistemas”. (Guedes, 2010, p.187)

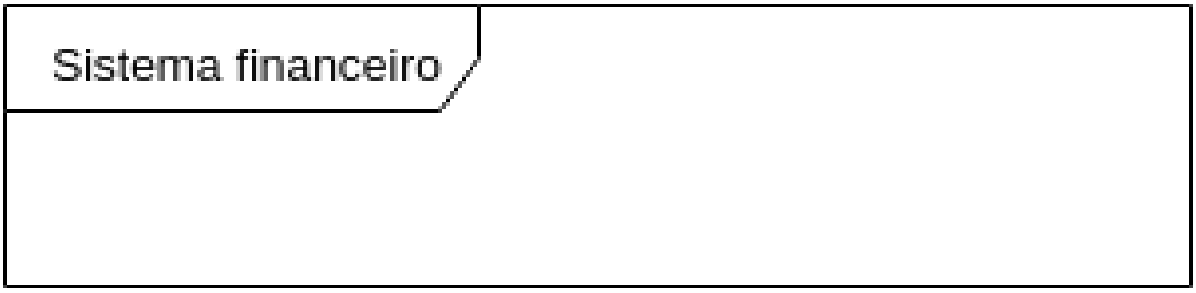

Figura 17 - Representação de um pacote
(Fonte: elaborado por Matheus Marabesi)

O propósito mais comumente utilizado é a organização dos elementos de modelagem em grupos que poderão ser nomeados e manipulados como um conjunto.

“Pacotes são utilizados para agrupar elementos e fornecer denominações para esses grupos”. (Guedes, 2010, p.187)

Com o diagrama de pacotes (figura 18) podemos visualizar de uma maneira genérica todo o sistema ou uma parte do mesmo. Podemos ainda entrar em um nível a mais de detalhes combinando o diagrama de pacotes com o diagrama de classes.

“O problema dessa abordagem é o grande espaço ocupado pelo pacote, sendo mais comum encontrar pacotes sem detalhamento de seu conteúdo”. (Guedes, 2010, p.188)

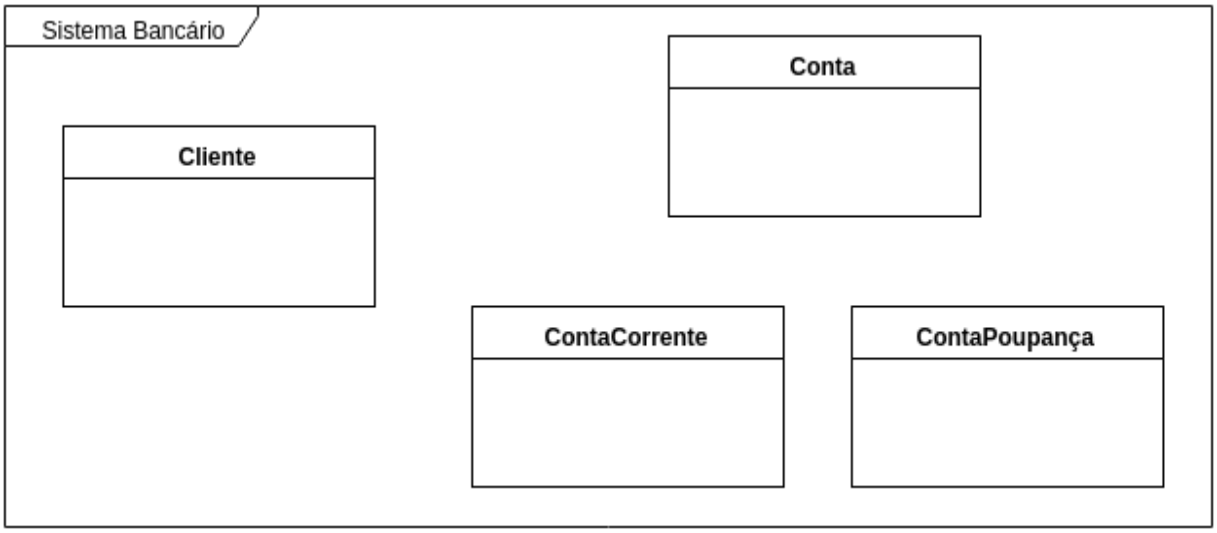

Figura 18 - Diagrama de pacotes.
(Fonte: elaborado por Matheus Marabesi)

Podemos utilizar diagramas de pacotes não somente com diagramas de classes, mas também outros tipos de diagramas da UML. Os pacotes são utilizados

para agrupar elementos da UML como classes e casos de uso. Diagrama de pacotes pode também conter dependência entre si.

“Se um elemento do pacote A usa um elemento do pacote B, então o pacote A depende do pacote B.” (Hamilton; Miles, 2006, p.289)

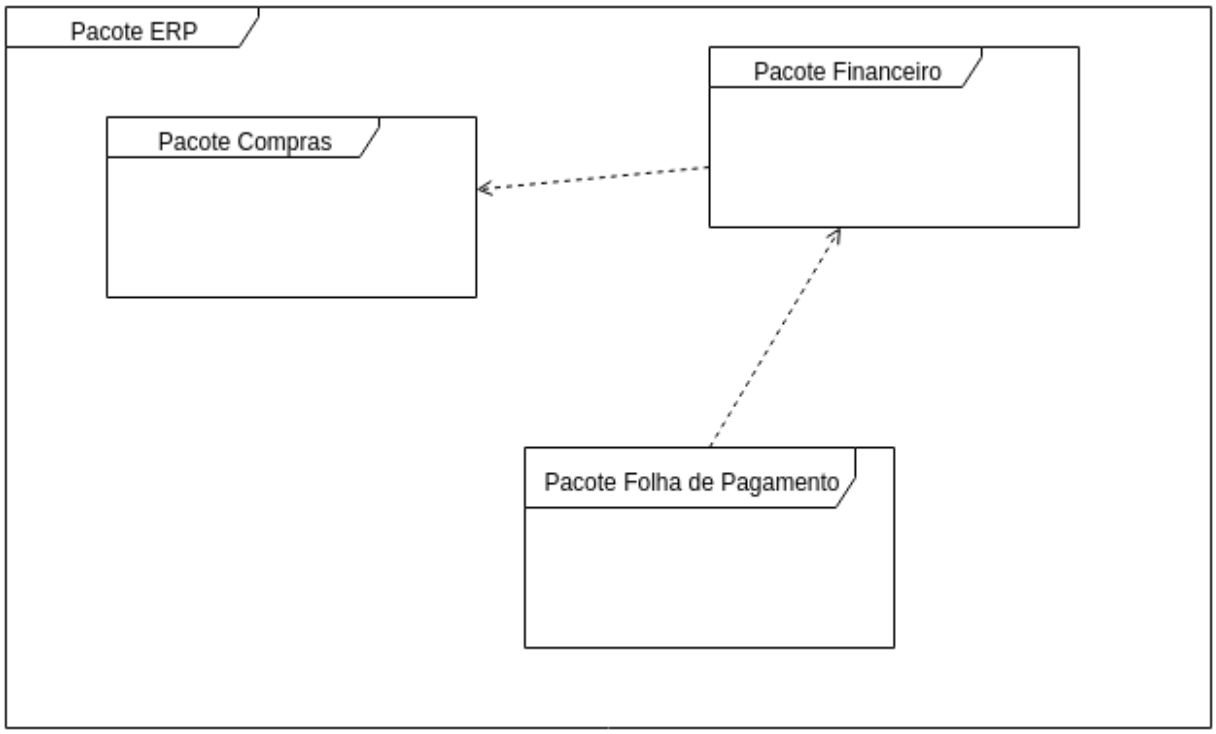

Figura 19 - Representação da dependência entre pacotes no diagrama de pacotes
(Fonte: UML 2 – Uma abordagem prática)

Como podemos observar, além da dependência entre pacotes, podemos também visualizar diagramas contendo pacotes dentro de pacotes. Na imagem 19 possuímos um pacote ERP que possui outros três pacotes, Pacote Compras, Pacote Financeiro e Pacote Folha de Pagamento.

“É bastante comum, principalmente na definição de arquiteturas de linguagens de modelagem, que um pacote se subdivide em diversos outros pacotes”. (Guedes, 2010, p.190)

2.3.4. Diagrama de Interação

Este diagrama atua como um termo genérico aplicado a junção de dois outros diagramas: sequência e colaboração. É utilizado para modelagem dos aspectos dinâmicos do sistema, mostrando a interação entre conjuntos de objetos e seus relacionamentos, inclusive as mensagens que poderão ser trocadas entre eles, conforme mostra a figura 20.

> Além de poder ser utilizado para expressar o comportamento de uma parte do sistema, quando é chamado de máquina de estado comportamental, também pode ser usado para expressar o protocolo de uso de parte de um sistema, quando identifica uma máquina de estado de protocolo. (Guedes, 2011, p.23)

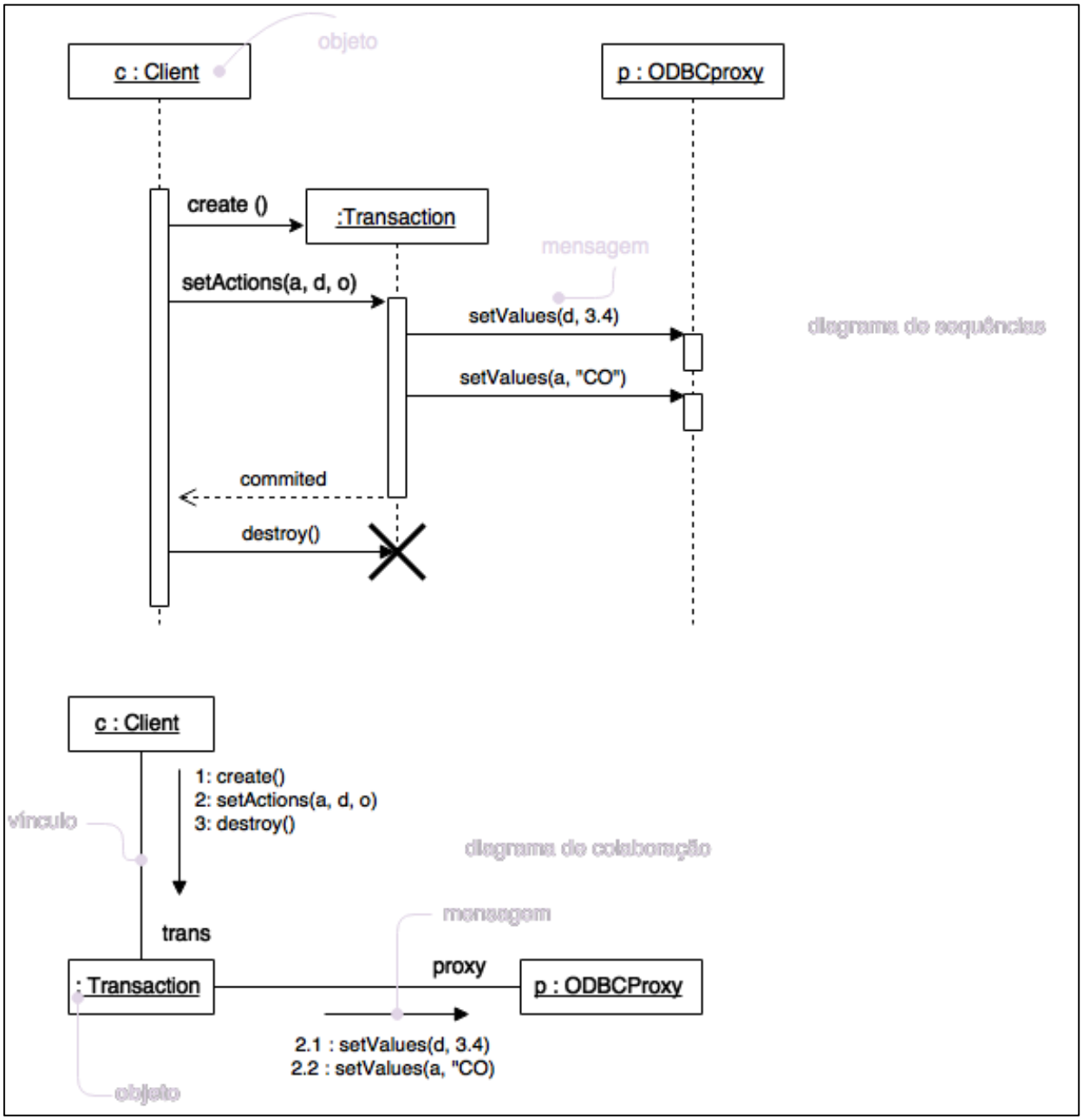

Figura 20 - Diagrama de interação
(Fonte: UML Guia do usuário)

2.3.5. Diagrama de Sequência

Diagrama comportamental que mostra uma interação (figura 21), dando ênfase à ordenação temporal das mensagens.

“Este é um diagrama comportamental que procura determinar a sequência de eventos que ocorrem em um determinado processo identificando quais mensagens devem ser disparadas entre os elementos envolvidos e em que ordem”. (Guedes, 2010, p.192)

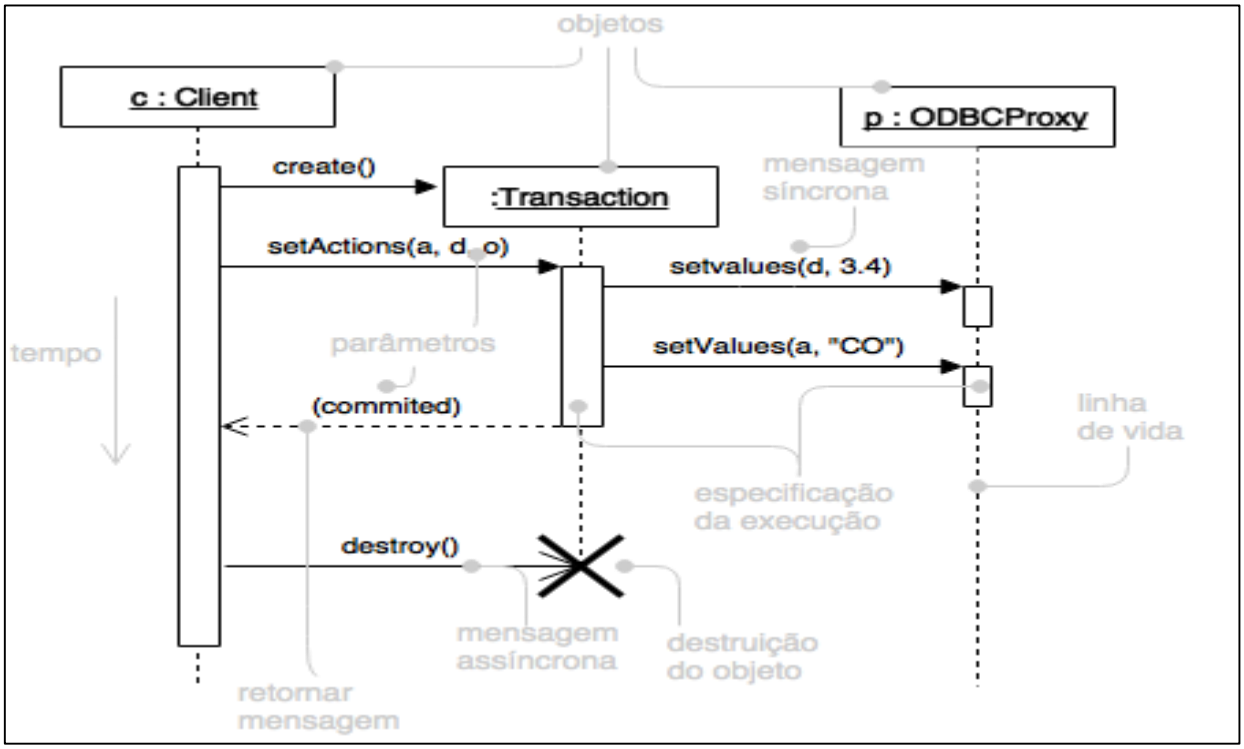

Figura 21 - Diagrama de sequência
(Fonte: UML Guia do usuário)

Diagramas de sequência possuem dois eixos: o eixo vertical, que mostra o tempo e o eixo horizontal, que mostra os objetos envolvidos na sequência de uma certa atividade. Eles também mostram as interações para um cenário específico de uma certa atividade do sistema.

No eixo horizontal estão os objetos envolvidos na sequência. Cada um é representado por um retângulo de objeto (similar ao diagrama de objetos) e uma linha vertical pontilhada chamada de linha de vida do objeto, indicando a execução do objeto durante a sequência, como exemplo citamos: mensagens recebidas ou enviadas e ativação de objetos.

A comunicação entre os objetos é representada como linha com setas horizontais simbolizando as mensagens entre as linhas de vida dos objetos. A seta especifica se a mensagem é síncrona, assíncrona ou simples. As mensagens podem

possuir também números sequenciais, eles são utilizados para tornar mais explícito a sequência no diagrama.

2.3.6. Diagrama de Comunicação

O diagrama de comunicação (figura 22) era conhecido como de colaboração até a versão 1.5 da UML, tendo seu nome modificado para diagrama de comunicação a partir da versão 2.0. Este diagrama é comportamental, pois mostra uma interação, dando ênfase à organização estrutural de objetos que enviam e recebem mensagens. Está amplamente associado ao diagrama de sequência: na verdade, um complementa o outro.

> As informações mostradas no diagrama de comunicação com frequência são praticamente as mesmas apresentadas no de sequência, porém com um enfoque distinto, visto que esse diagrama não se preocupa com a temporalidade do processo, concentrando-se em como os elementos do diagrama estão vinculados e quais mensagens trocam-se entre si durante o processo. (GUEDES, 2011 ,p. 231)

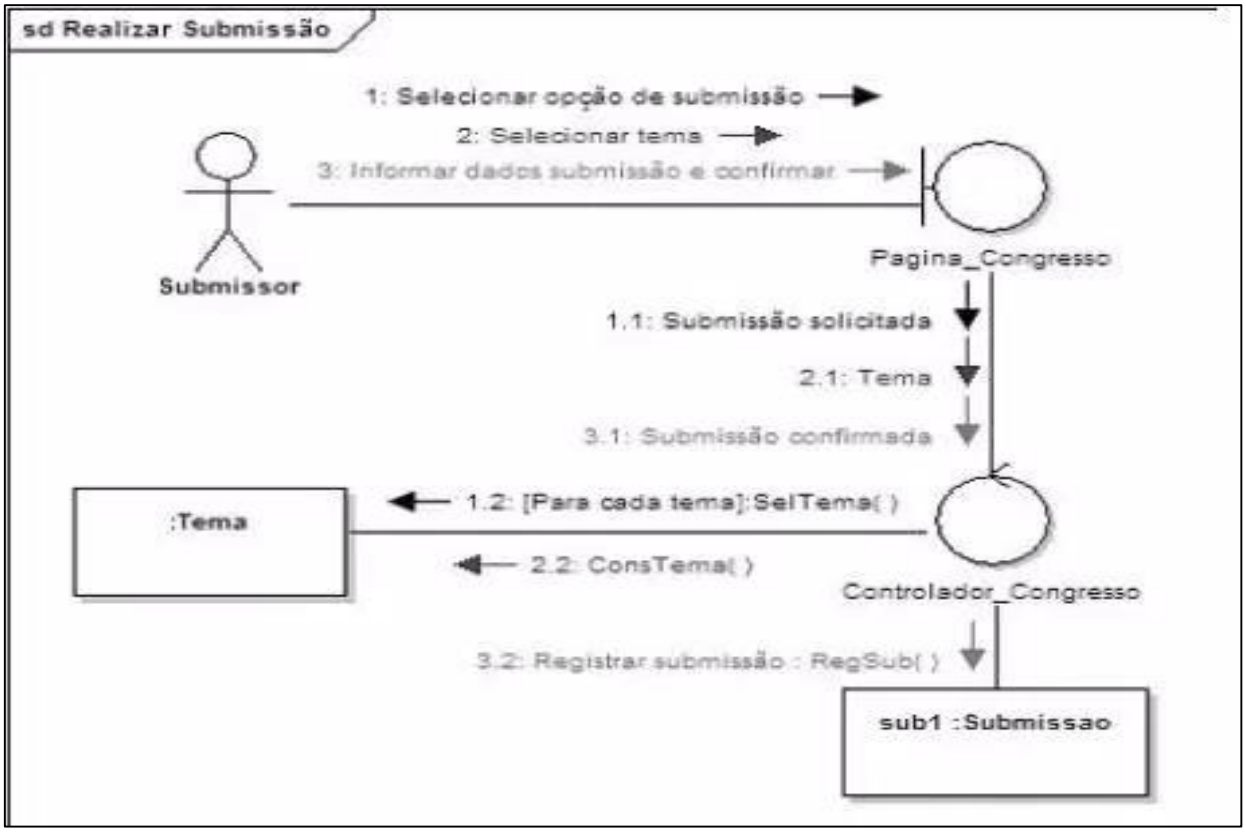

Figura 22 - Diagrama de comunicação
(Fonte: UML 2 – Guia Prático)

2.3.7. Diagrama de Máquina de Estados

Diagrama comportamental (figura 23) que mostra uma máquina de estados, dando ênfase ao comportamento ordenados por eventos de um objeto.

"Além de poder ser utilizado para expressar o comportamento de uma parte do sistema, [..] pode ser utilizado para expressar o protocolo de uso de parte de um sistema". (Guedes, 2010, p.242)

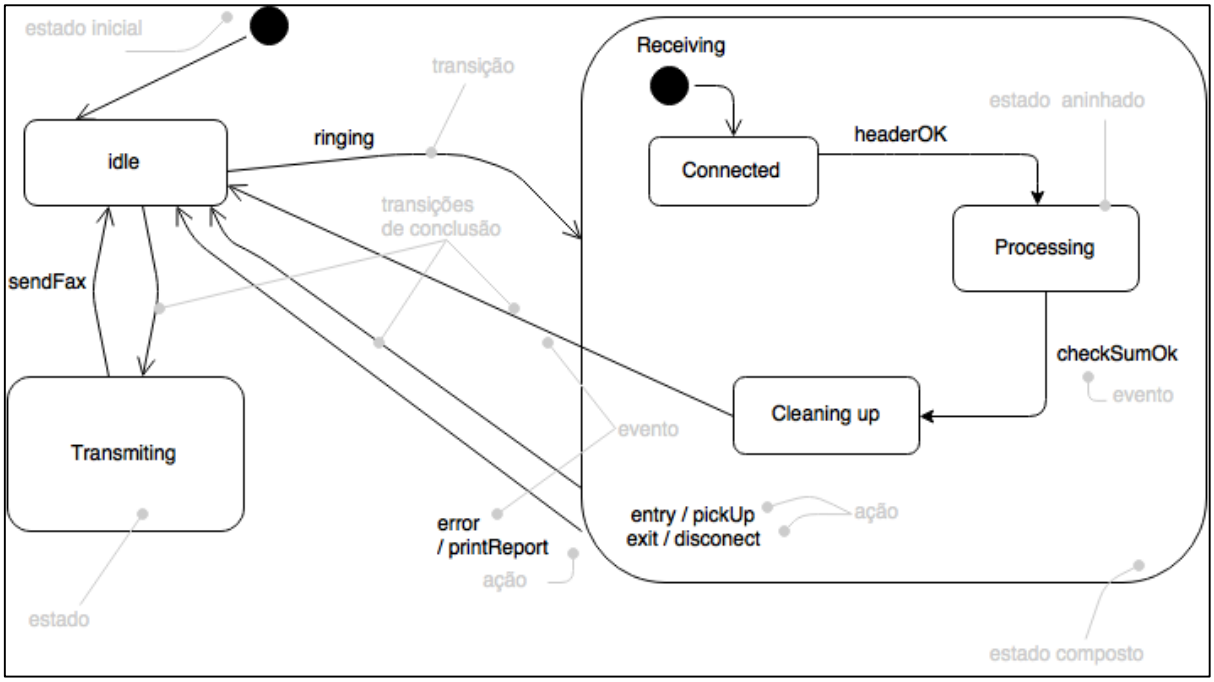

Figura 23 - Diagrama de máquina de estados
(Fonte: UML Guia do usuário)

Uma máquina de estados é um comportamento que especifica as sequencias de estados pelos quais um objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos.

Um estado é uma condição ou situação na vida de um objeto durante o qual ele satisfaz a alguma condição, realiza alguma atividade ou aguarda algum evento. Graficamente, um diagrama de gráficos de estados é uma coleção de vértices e arcos.

2.3.8. Diagrama de Atividades

Diagrama comportamental que mostra um processo computacional, dando ênfase ao fluxo de uma atividade para outra, conforme ilustrado na figura 24.

Este diagrama atua como um fluxograma e modela o tempo de vida de um objeto no sistema.

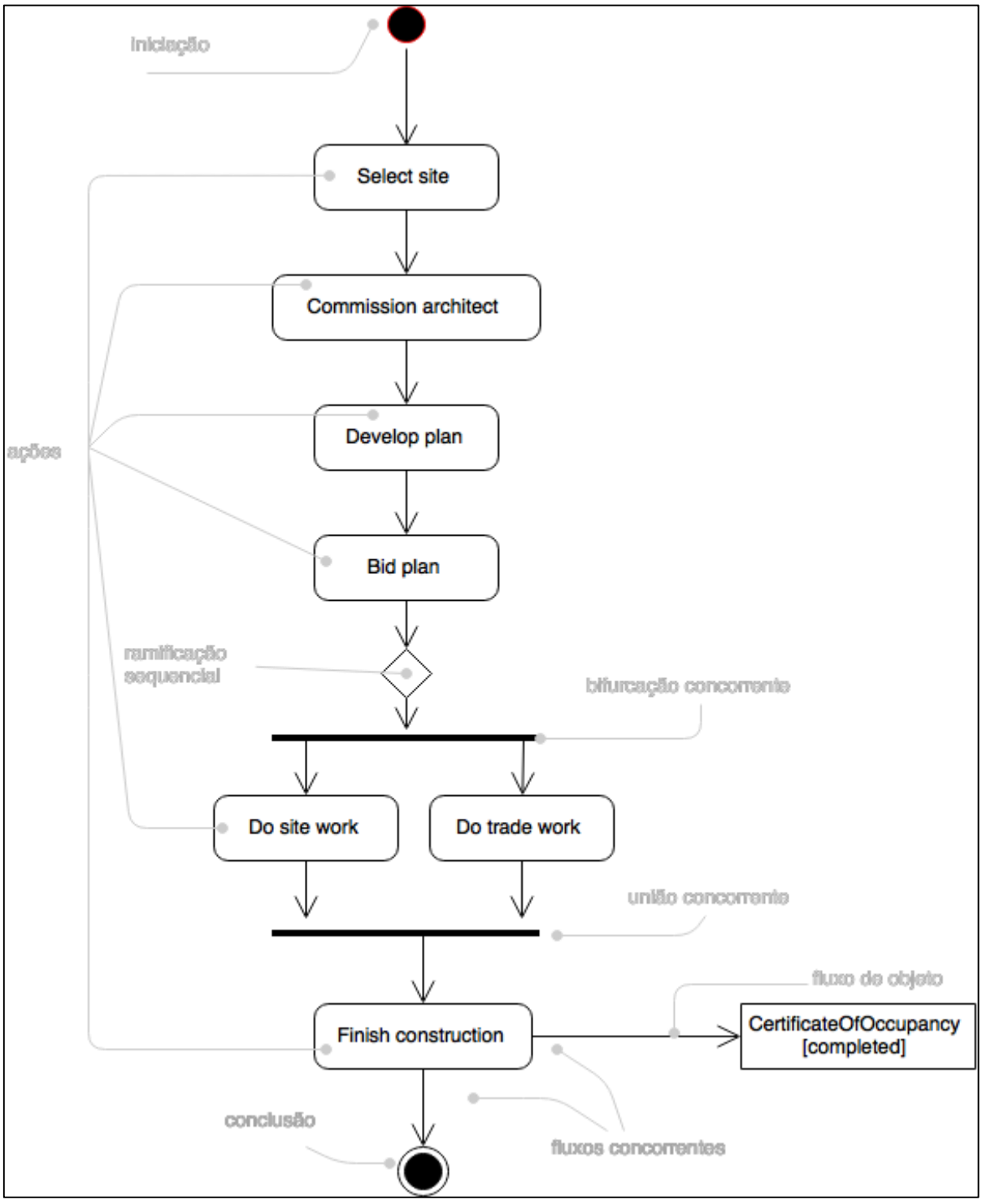

Figura 24 - Diagrama de atividades
(Fonte: UML Guia do usuário)

Uma atividade é caracterizada por um conjunto de ações em um determinado momento que ocasiona uma mudança de estado.

2.3.9. Diagrama de Visão Geral de Interação

Diagrama introduzido a partir da 2ª versão da UML. Este diagrama é comportamental, pois combina aspectos dos diagramas de atividades e dos diagramas de sequências, conforme ilustrado na figura 25.

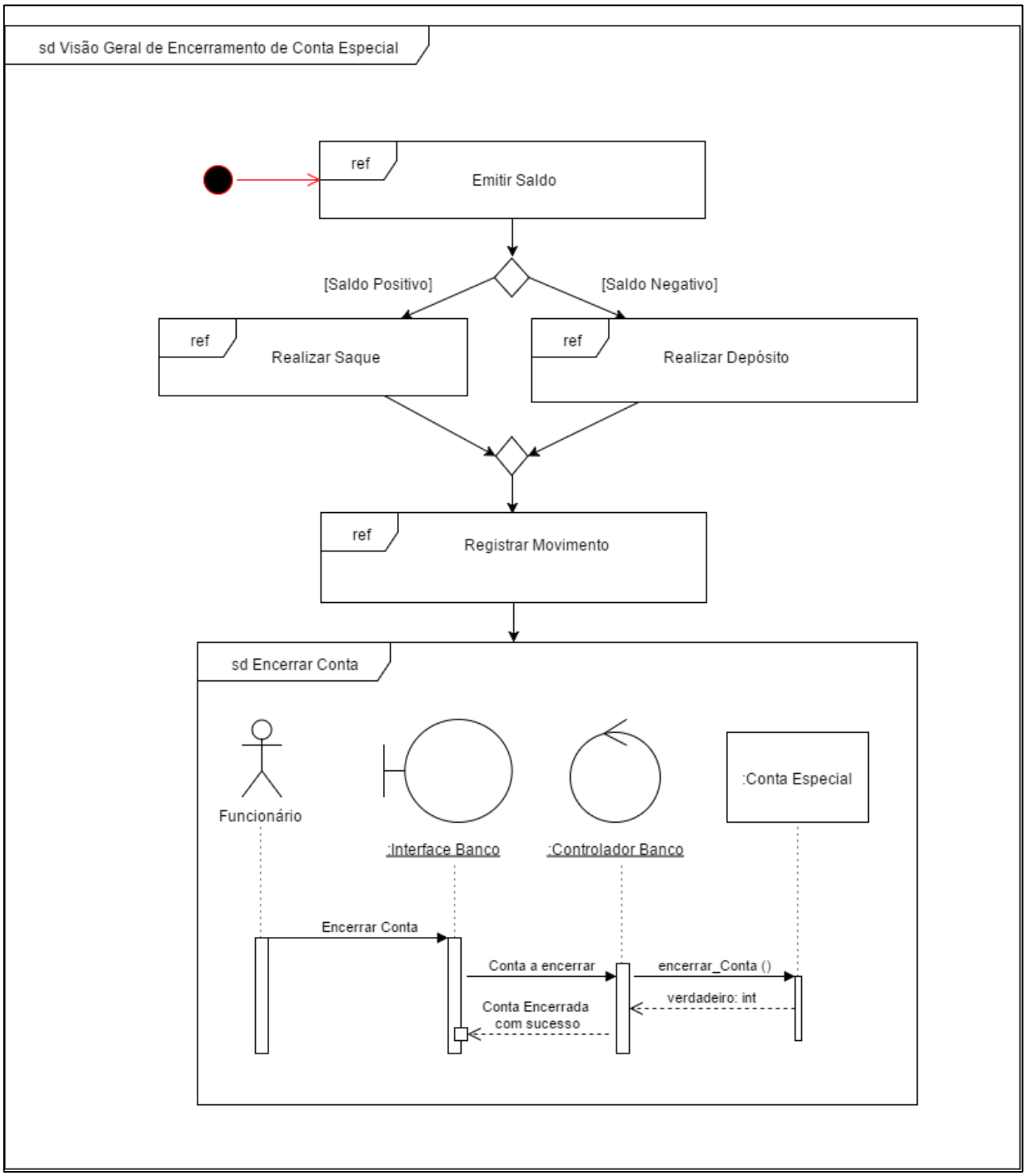

Figura 25 - Diagrama de visão geral de interação
(Fonte: UML 2, p. 314)

O diagrama de visão geral de interação não referência os casos de uso, mas as respectivas modelagens de interação.

2.3.10. Diagrama de Componentes

Diagrama estrutural que mostra as interfaces externas (figura 26), incluindo portas e a composição interna de um componente. Entende-se por componente uma parte lógica e substituível de um sistema ao qual se adapta.

A representação de um componente é um retângulo com um pequeno ícone de dois pinos no canto superior direito. O nome do componente aparece no retângulo.

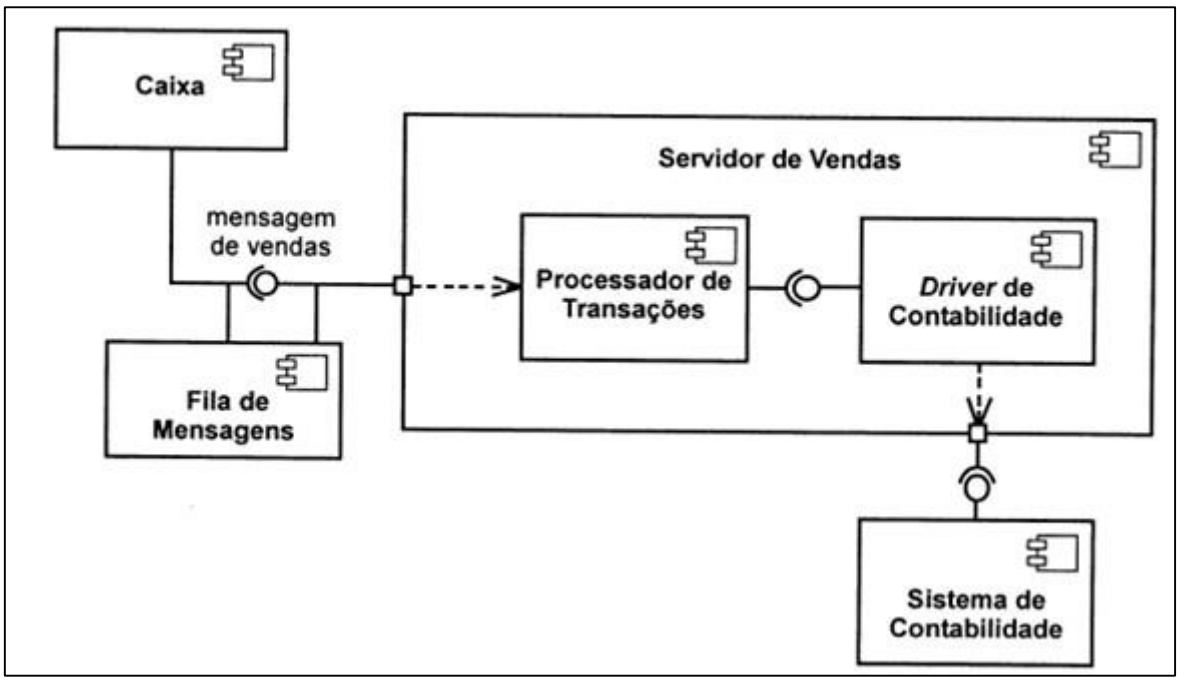

Figura 26 - Diagrama de componentes
(Fonte: UML Essencial)

Os componentes permitem o encapsulamento das partes do sistema, para reduzir as dependências, torná-las explícitas e melhorar a capacidade de substituição e flexibilização para possíveis alterações futuras.

2.3.11. Diagrama de Implantação

Diagrama estrutural que mostra os relacionamentos entre um conjunto de nós (figura 27), artefatos e classes manifestadas e componentes. É empregado na modelagem estática da implantação de um sistema.

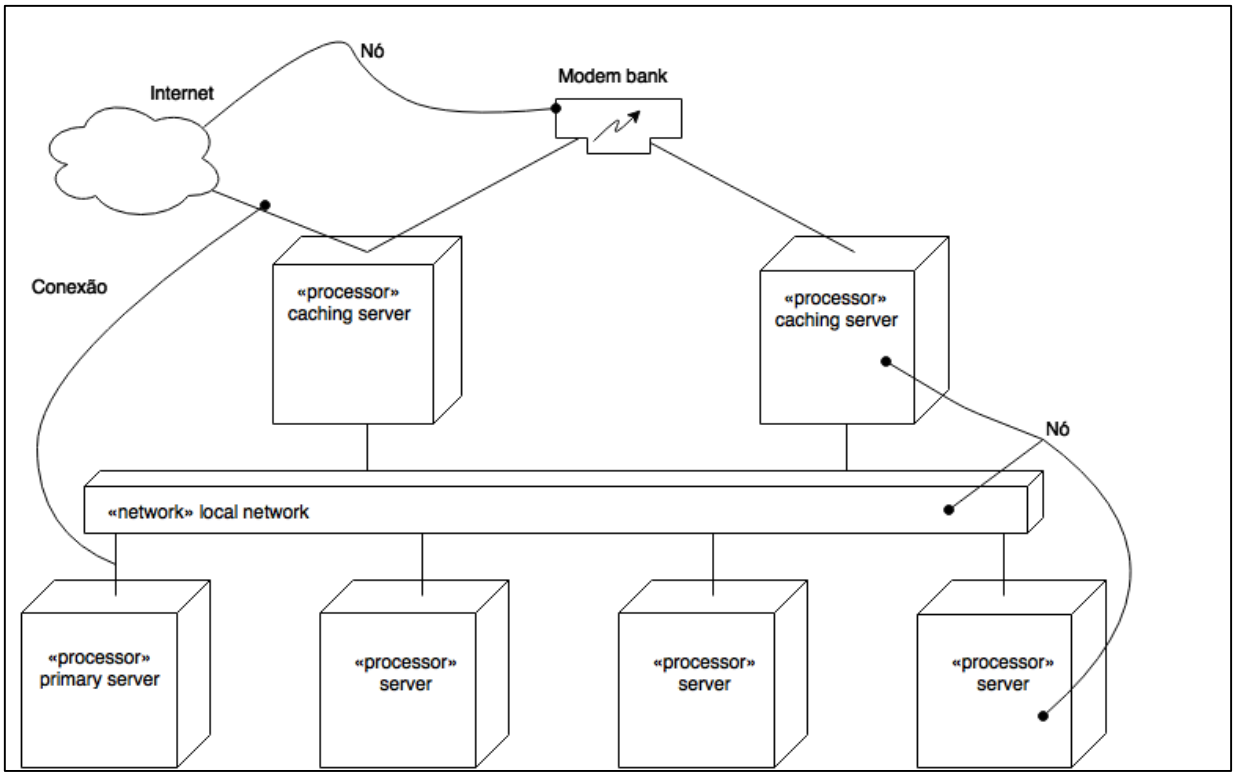

Figura 27 - Diagrama de implantação
(Fonte: UML Guia do usuário)

Sua finalidade não se restringe apenas para visualização, especificação e documentação de sistemas embutidos, mas como afirma Booch, serve também “para o gerenciamento de sistemas executáveis por meio de engenharia de produção e reversa” (Booch, 2006, pag.411).

2.3.12. Diagrama de Caso de Uso

Este é o diagrama mais conhecido e utilizado dentro da UML, sobretudo nas fases de levantamento e análises de requisitos do sistema. Sua característica é comportamental, pois representa relação entre use case e ator, caracterizando a funcionalidade e comportamento do sistema. Originalmente este diagrama foi criado por Ivar Jacobson, baseado em sua experiência no desenvolvimento de um sistema para a Ericsson com a utilização dos métodos OOSE e Objectory. Por meio desta modelagem pode-se ter uma noção de como será o funcionamento do sistema, sem preocupar-se com a implementação do mesmo. Podemos definir como a primeira abstração do sistema a ser desenvolvido.

O diagrama de casos de uso é composto por ‘atores’ e os ‘casos de uso’ propriamente ditos, conforme a figura 28.

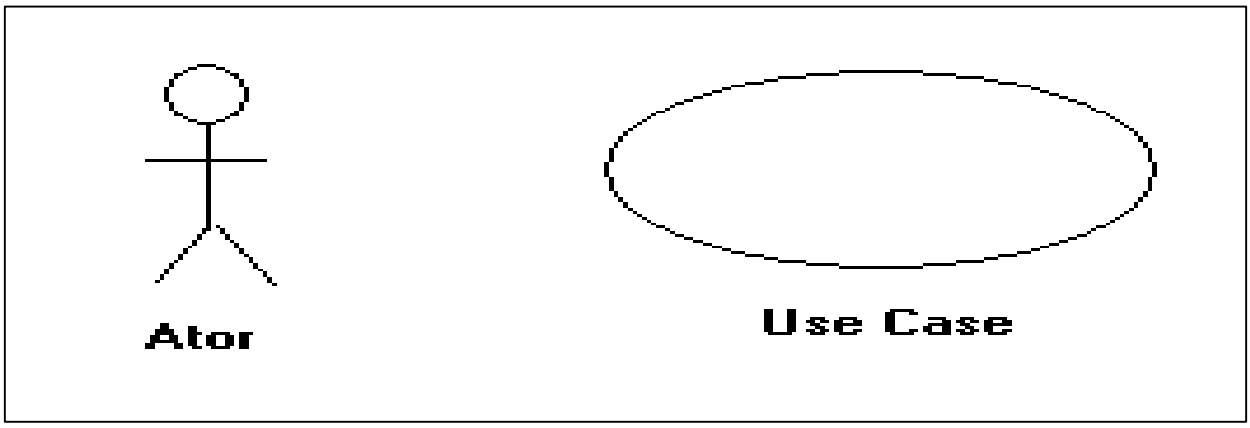

Figura 28 - Ator e caso de uso
(Fonte: UML Guia do usuário)

2.3.12.1. Atores

Um ator representa as interações que um ou mais usuários irão ter. O ator não diz somente a um ser humano, mas também dispositivos, sistemas que venham a interagir com a aplicação. Os atores comunicam-se com o sistema por meio de use-cases.

2.3.12.2. Casos de Uso

Um caso de uso é descrito como um conjunto de sequências de ações, inclusive variantes, que o sistema executa para produzir um resultado de valor observável por um ator. O caso de uso é representado por uma elipse.

2.3.12.2.1. Documentação de Casos de Uso

Para Bezerra (2006), a UML não define um formato engessado de caso de uso, porém, existem itens “realmente úteis” que precisam ser incluídos para que o caso de uso seja inteligível e compreensível para os usuário e programadores.

2.3.12.2.1.1. Nome

Representa o nome que aparece no diagrama de caso de uso, que deve possuir um nome único.

2.3.12.2.1.2. Identificador

Um número único que permite fazer uma referência cruzada entre diversos documentos relacionados com o modelo de caso de uso.

2.3.12.2.1.3. Importância

A definição da complexidade e relevância do caso de uso.

2.3.12.2.1.4. Descrição Resumida

Uma breve descrição do caso de uso e suas funcionalidades, devendo ter no máximo duas frases.

2.3.12.2.1.5. Ator Primário

Nome do ator que inicia o caso de uso.

2.3.12.2.1.6. Ator Secundário

Nome dos atores que possuem uma ação no caso de uso, caso exista.

2.3.12.2.1.7. Pré-condições

São situações que devem ser atendidas antes de iniciar o caso de uso como, por exemplo, o usuário deve estar autenticado no sistema para acessar a tela.

2.3.12.2.1.8. Fluxo Principal

Este item corresponde ao passo a passo que deve ser executado pelo ator e pelo sistema (no ponto de vista do usuário) para a execução da funcionalidade do caso de uso. O texto deve ser objetivo e claro, devendo também mencionar os fluxos alternativos e fluxos de exceção, caso existam.

2.3.12.2.1.9. Fluxo Alternativo

Este fluxo referência uma alternativa, diferente do fluxo principal, que pode ser realizada pelo ator. Ele tipo de fluxo também pode ser usado para situações de escolhas exclusivas entre si (em que existem diversas alternativas e somente uma deve ser realizada).

A figura abaixo ilustra a diferença de um fluxo alternativo independe e exclusivas entre si, onde as linhas tracejadas representam os casos de uso.

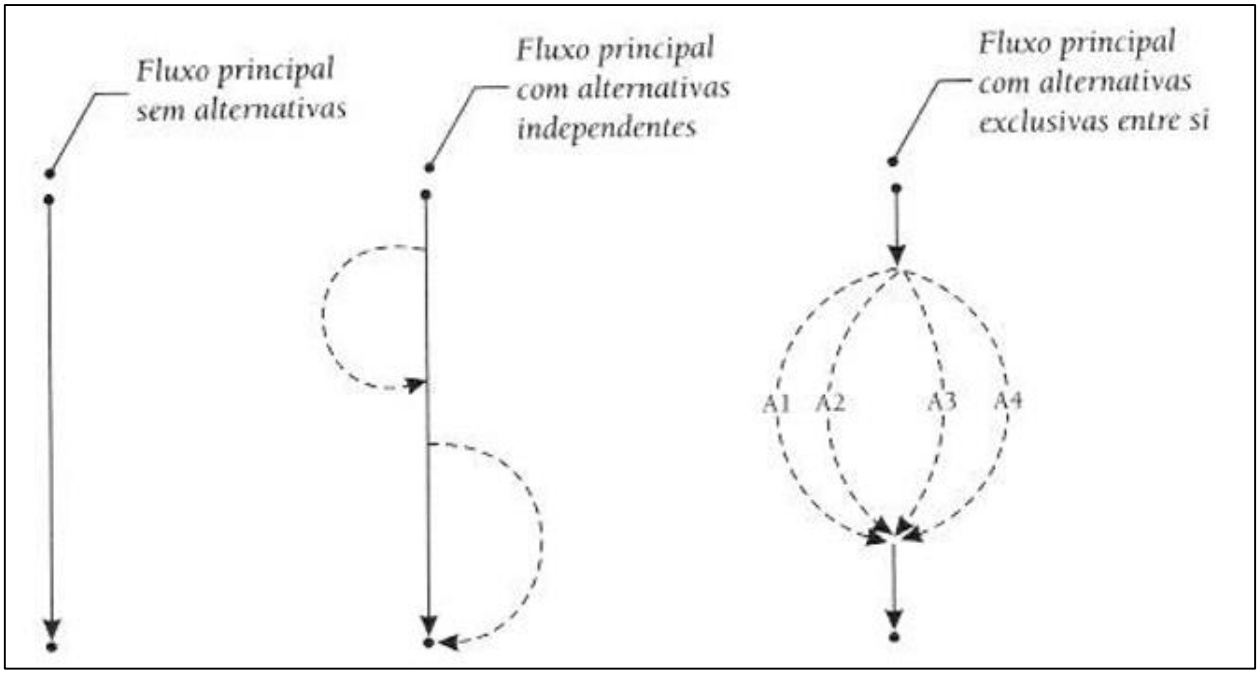

Figura 29 - Fluxos alternativos em um caso de uso
(Fonte: Princípios de Análise e Projetos de Sistemas com UML, p. 67)

2.3.12.2.1.10. Fluxo de Exceção

Descreve sobre as situações de exceção que podem ocorrer durante a execução do fluxo principal como, por exemplo, quando o usuário realiza uma ação inválida. Este fluxo possui algumas características importantes, tais como: podem representar possíveis erros durante a execução, deve sempre indicar o passo em que o caso de uso continuará ou se terminará e não fazem sentido fora do contexto do caso de uso.

Bons exemplos para o entendimento de fluxos de exceções seriam “se o usuário insere uma senha incorreta” e “se a loja não tem estoque suficiente”.

2.3.12.2.1.11. Regras de Negócios

São políticas, restrições e condições que devem ser respeitados. Elas geralmente recebem um identificador único durante as fases de levantamento de requisitos. Uma regra de negócios pode ser descrita informalmente ou através de uma estruturação de passos, sempre mencionando ações que influenciam no negócio, por

exemplo, “o número máximo de alunos por turma é de 30” ou “um professor só pode lecionar 5 disciplinas em cada escola”.

2.3.12.2.1.12. Pós-condições

Após a realização do caso de uso, o comportamento do sistema será alterado (por menor que seja) e deve ser descrito neste item.

2.3.12.2.1.13. Histórico

Este item é utilizado para descrever quem alterou, quando e o que foi modificado no caso de uso.

2.3.12.3. Relacionamentos

Segundo Bezerra (2006), casos de usos e atores não existem sozinhos. Um ator deve estar relacionado a um ou mais casos de uso, de forma que possa existir também relacionamentos entre casos de uso de um sistema. A UML define alguns tipos de relacionamentos no modelo de caso de uso, sendo eles, “comunicação”, “inclusão”, “extensão” e, por fim, “generalização”.

2.3.12.3.1. Relacionamento de Comunicação

Representa a informação de quais atores estão associados ao caso de uso, significando que o ator exerce algum ato (troca informações) com o sistema. Estes atores podem interagir com mais de um caso de uso.

2.3.12.3.2. Relacionamento de Inclusão

Este relacionamento é realizado apenas entre casos de uso e possui um conceito voltado para programação (rotina). Após a execução de uma sequência de instruções, a “rotina” é chamada de algum ponto do programa. Caso dois ou mais casos de uso incluam uma sequência de interações, essa sequência pode ser descrita em outro caso de uso, evitando redundâncias e deixando os casos de uso mais simples. A figura abaixo apresenta alguns exemplos de relacionamento de inclusão.

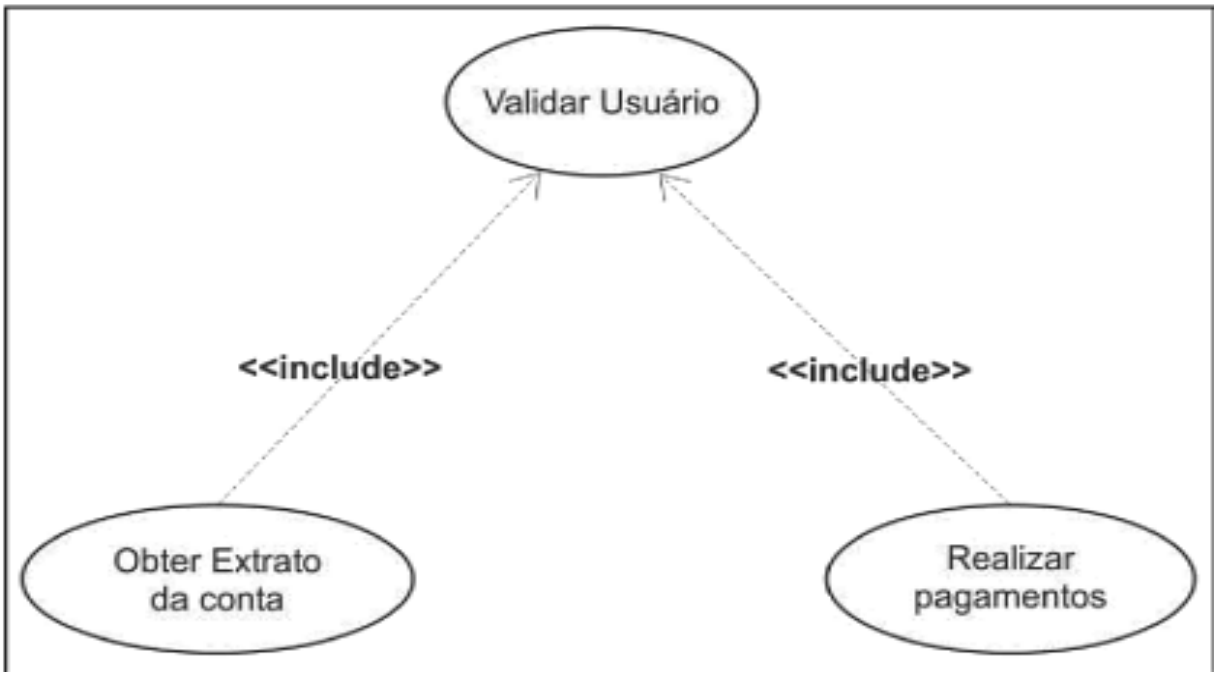

Figura 30 - Relação de Inclusão entre casos de uso
(Fonte: Treinamento Prático em UML)

2.3.12.3.3. Relacionamento de Extensão

Este relacionamento é utilizado para especificar situações em que diferentes sequências de interações podem ser inseridas em um caso de uso, chamado “estendido”. O comportamento dos casos de uso só ocorre sob certas condições, ou cuja realização depende diretamente do ator.

Existem duas formas de definir em que ponto no caso de uso estendido pode ocorrer o comportamento do extensor, sendo que a primeira é a definição na descrição textual e a segunda é através de um ou mais pontos de extensão (figura 31). Um ponto de extensão é a indicação de onde o extensor pode ser inserido. O recomendado é utilizar a primeira alternativa, uma vez que pontos de extensão “poluem” o caso de uso. Bons exemplos para a utilização de extensores seriam os casos de uso “Corrigir Ortografia” ou “Substituir Texto”, que podem ser definidos como extensões do caso de uso “Editar Documento”.

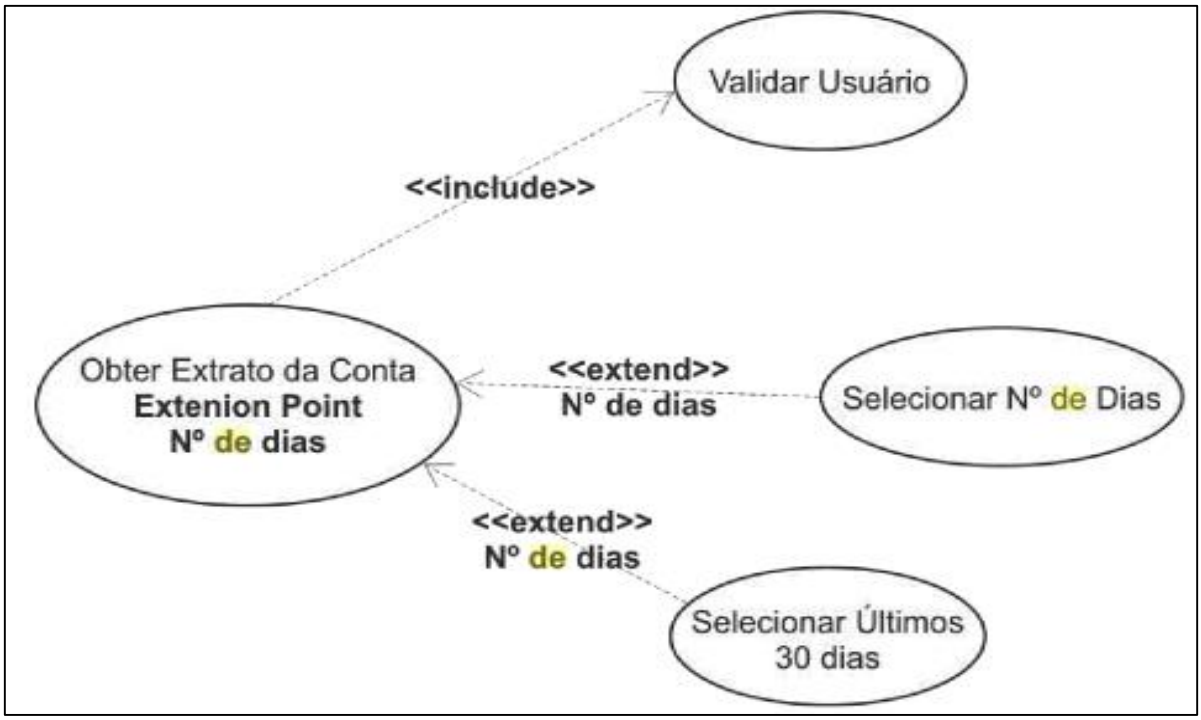

Figura 31 - Relação de extensão entre casos de uso
(Fonte: Treinamento Prático em UML)

2.3.12.3.4. Relacionamento de Generalização

Este relacionamento é utilizado quando fica evidente o reuso de ações, ou seja, permite que o caso de uso “herde” características de outro caso de uso mais genérico. Este tipo de caso de uso é conhecido como “base”. Este relacionamento pode existir entre casos de usos ou entre atores, e sua vantagem é não precisar reescrever funcionalidades.

Uma generalização entre atores quer dizer que o ator “herdeiro” possui o mesmo comportamento (em relação ao sistema) que o ator do qual ele herda, conforme mostra a figura abaixo.

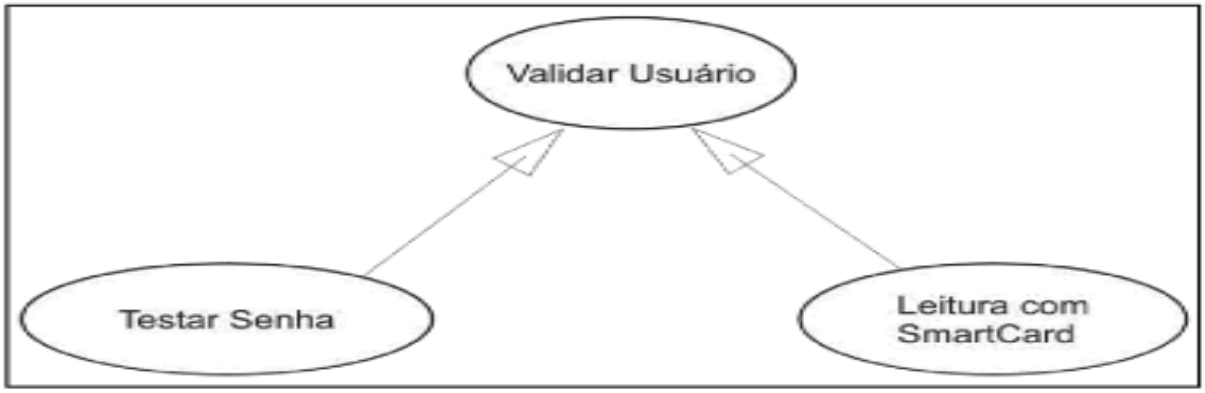

Figura 32 - Relação de Generalização entre casos de uso (Fonte: Treinamento Prático em UML)

A vantagem de utilizar conceitos de relacionamentos entre casos de usos é a simplificação. Porém, é necessário atentar-se que o excesso de utilização dos relacionamentos pode dificultar a compreensão do desenvolvedor.

3. SOLUÇÃO PROPOSTA

Neste capítulo apresentaremos a especificação da ferramenta que auxiliará no gerenciamento da documentação dos casos de uso. O protótipo deverá auxiliar na fase de documentação de casos de uso, permitindo o armazenamento e organização dos requisitos levantados pelo engenheiro de software.

3.1. Funcionalidade do Sistema

O sistema permitirá ao usuário documentar um ou mais casos de uso por meio de software de fácil interação. Este programa atenderá somente a parte do caso de uso, possibilitando o cadastro de um requisito, regras de negócio, atores envolvidos, fluxo básico, fluxo de exceção e alternativo no que diz respeito a um requisito funcional.

3.2. Técnicas e Ferramentas

Realizamos pesquisa de mercado para analisar quais ferramentas que são comercializadas hoje, quais vantagens possuem e que funcionalidades seriam essenciais para o desenvolvimento de nossa própria aplicação.

3.2.1. Análise de Ferramentas de mercado

Para desenvolvimento da ferramenta, analisamos o que hoje é oferecido pelo mercado de TI. Em nosso levantamento, destacamos os softwares pagos Dimensions RM, da Serena e Rational Requisite Pro da IBM. Serena Dimensions RM é uma aplicação web para melhorar a definição e gestão dos requisitos em todo o ciclo de vida do produto para assegurar a visibilidade de ideias iniciais para entregas, bem como a entrega de baixo custo no tempo.

Poderíamos dividir a aplicação em duas grandes categorias: definição de requisitos (extração, analise, especificação e validação dos requisitos) e gerenciamento de requisitos (gestão e acompanhamento de requisitos através do desenvolvimento). A ferramenta Requisite Pro permite a visualização da informação

nas fases de projeto: demanda inicial, estudo preliminar, projeto lógico, projeto físico e construção, integradas por versão.

Neste software é criado um projeto onde existirá uma hierarquia de documentação e padronização para diferente níveis de requisitos de um produto. A arquitetura do Requisite Pro é composta de documentos, atributos de requisitos, repositórios de requisitos e rastreabilidade.

3.2.2. Modelo de Dados da Solução

Abaixo o modelo de dados da solução desenvolvida:

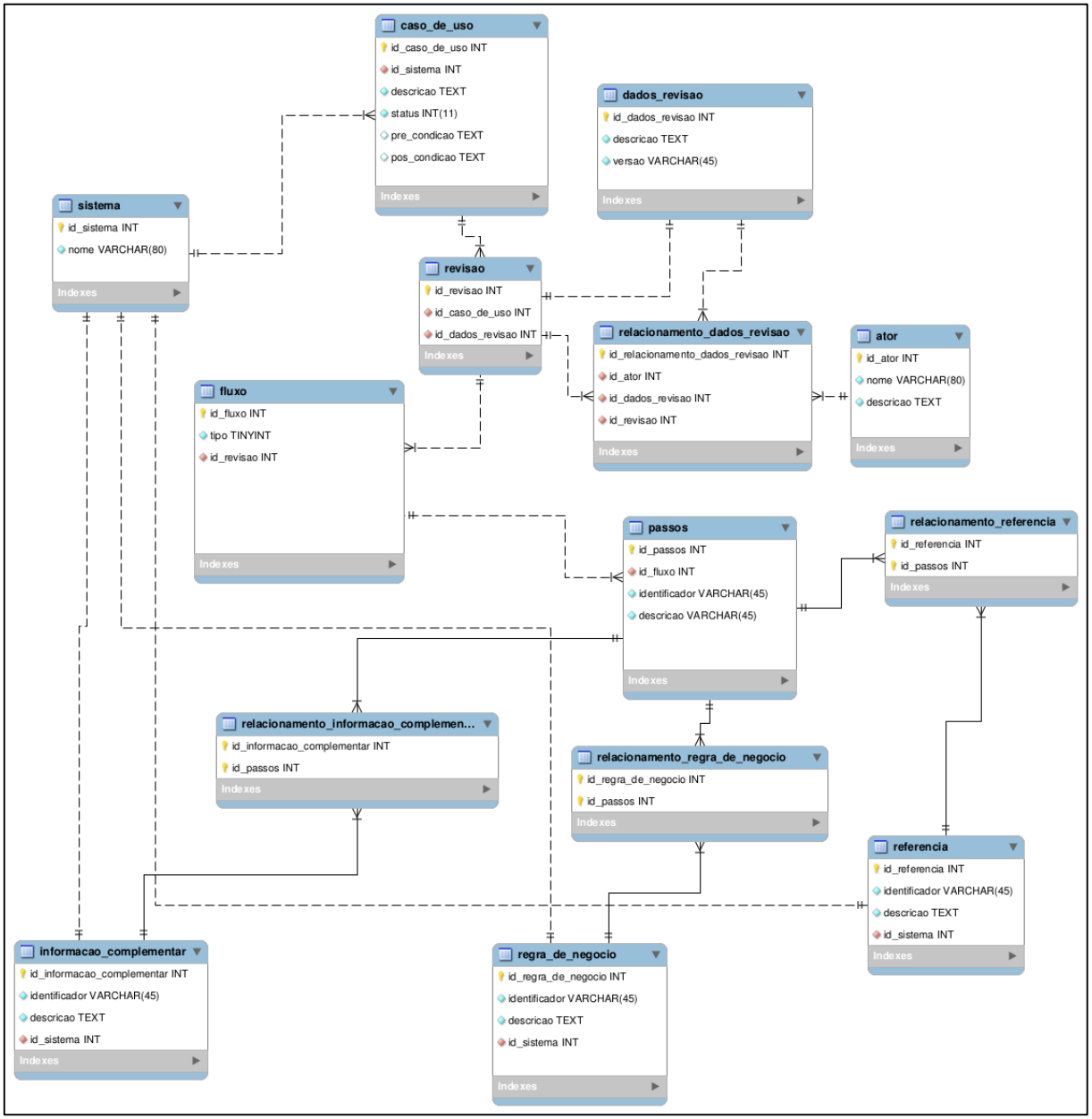

Figura 33 - Modelo de Entidade Relacionamento
(Fonte: elaborado por Matheus Marabesi)

3.3. Caso de Uso

Para validação do sistema, documentamos os casos de uso referente a cada funcionalidade que a aplicação terá. Eles foram listados abaixo:

3.3.1. Caso de uso: Sistema

Caso de Uso	Cadastrar Sistema
Ator	Analista de Requisitos
Descrição	Cadastrar o nome da aplicação para qual serão cadastrados os casos de uso
Pré-condição	Usuário deverá ter permissão para cadastrar.
Cenário Principal	Este caso de uso inicia quando o sistema apresenta a opção de cadastros de sistema. 1. O sistema apresenta o formulário de cadastros de sistemas; 2. O Analista de Requisitos preenche as informações necessárias e confirma a operação; [Fluxo alternativo 01: Exclusão de um sistema] [Fluxo alternativo 02: Edição de um sistema] 3. O sistema exibe em uma tabela o nome do sistema criado; 4. O caso de uso termina.
Cenário Alternativo	Fluxo alternativo 01: Exclusão de um sistema Este fluxo alternativo inicia quando o Analista de Requisitos seleciona a opção de exclusão de um sistema previamente cadastrado. [Exceção 01: Dependência entre casos de uso] 1. O sistema apresenta mensagem de confirmação de exclusão; 2. O Analista de Requisitos confirma a ação; 3. O sistema exclui as informações do sistema; 4. O caso de uso termina. Fluxo alternativo 02: Edição de um sistema Este fluxo alternativo inicia quando o Analista de Requisitos seleciona um sistema e a ação de edição.

	1. O sistema apresenta o formulário com as informações preenchidas; 2. O Analista de Requisitos edita as informações; 3. O sistema armazena as alterações; 4. O caso de uso termina. Exceção 01: Dependência entre casos de uso 1. O sistema identifica que o sistema possui dependência com um caso de uso e não permite a exclusão; 2. O caso de uso termina.
Pós-condição	Sistema cadastrado com sucesso.

3.3.2. Caso de uso: Caso de Uso

Caso de Uso	Cadastrar Caso de Uso
Ator	Analista de Requisitos
Descrição	Cadastrar informações referente ao caso de uso
Pré-condição	Um sistema já deve ter sido cadastrado.
Cenário Principal	Este caso de uso inicia quando o sistema apresenta a opção de cadastros de casos de uso. 1. O Analista de Requisitos seleciona o sistema ao qual quer associar o caso de uso; 2. O Analista de Requisitos preenche a descrição para o caso de uso, seleciona em uma lista o status do caso de uso e confirma a operação; [Fluxo alternativo 01: Exclusão de um caso de uso] [Fluxo alternativo 02: Edição de um caso de uso] 3. O sistema armazena as informações e exibe uma lista com os casos de uso cadastrados; 4. Encerrar caso de uso.
Cenário Alternativo	Fluxo alternativo 01: Exclusão de um sistema Este fluxo alternativo inicia quando o Analista de Requisitos seleciona a opção de exclusão de um caso de uso. [Exceção 01: Dependência entre passos] 1. O sistema apresenta mensagem de confirmação de exclusão; 2. O Analista de Requisitos confirma a ação; 3. O sistema exclui as informações do caso de uso;

	4. O caso de uso termina. Fluxo alternativo 02: Edição de um caso de uso Este fluxo alternativo inicia quando o Analista de Requisitos seleciona um caso de uso e a ação de edição. 1. O sistema apresenta o formulário com as informações preenchidas e o campo “Sistema” desabilitado para edição; 2. O Analista de Requisitos edita as informações; 3. O sistema armazena as alterações; 4. O caso de uso termina. Exceção 01: Dependência entre passos 1. O sistema identifica que o caso de uso possui dependência com um passo e não permite a exclusão; 2. O caso de uso termina.
Pós-condição	O caso de uso é cadastrado com sucesso.

3.3.3. Caso de uso: Versionamento

Caso de Uso	Cadastrar Versionamento
Ator	Analista de Requisitos
Descrição	Cadastrar número da versão na qual ficarão associados os casos de uso do sistema.
Pré-condição	Um caso de uso já deverá ter sido cadastrado.
Cenário Principal	Este caso de uso inicia quando o sistema apresenta a opção de cadastros de versionamento. 1. O Analista de Requisitos informa o número, uma descrição para a versão e confirma a operação; 2. O sistema armazena as informações; [Fluxo alternativo 01: Exclusão de um versionamento] [Fluxo alternativo 02: Edição de um versionamento] 3. O sistema armazena as informações; 4. O caso de uso termina.
Cenário Alternativo	Fluxo alternativo 01: Exclusão de um versionamento Este fluxo alternativo inicia quando o Analista de Requisitos seleciona a opção de exclusão de um versionamento

	previamente cadastrado. [Exceção 01: Dependência entre casos de uso] 1. O sistema apresenta mensagem de confirmação de exclusão; 2. O Analista de Requisitos confirma a ação; 3. O sistema exclui as informações do versionamento; 4. O caso de uso termina. Fluxo alternativo 02: Edição de um versionamento Este fluxo alternativo inicia quando o Analista de Requisitos seleciona um versionamento e a ação de edição. 1. O sistema apresenta o formulário com as informações preenchidas; 2. O Analista de Requisitos edita as informações; 3. O sistema armazena as alterações; 4. O caso de uso termina. Exceção 01: Dependência entre casos de uso 1. O sistema identifica que o sistema possui dependência com um caso de uso e não permite a exclusão; 2. O caso de uso termina.
Pós-condição	O versionamento é cadastrado com sucesso.

3.3.4. Caso de uso: Ator

Caso de Uso	Cadastrar Ator
Ator	Analista de Requisitos
Descrição	Cadastrar tipo de ator que estará associado a um caso de uso.
Pré-condição	Não se aplica
Cenário Principal	Este caso de uso inicia quando o sistema apresenta a opção de cadastros de atores. 1. O sistema apresenta o formulário de cadastros de atores; 2. O Analista de Requisitos preenche as informações necessárias e confirma a operação; [Fluxo alternativo 01: Exclusão de um ator] 3. O sistema exibe em uma tabela o nome do ator criado;

	4. O caso de uso termina.
Cenário Alternativo	Fluxo alternativo 01: Exclusão de um ator Este fluxo alternativo inicia quando o Analista de Requisitos seleciona a opção de exclusão de um ator previamente cadastrado. [Exceção 01: Dependência entre casos de uso] 1. O sistema apresenta mensagem de confirmação de exclusão; 2. O Analista de Requisitos confirma a ação; 3. O sistema exclui as informações do ator; 4. O caso de uso termina. Exceção 01: Dependência entre casos de uso 1. O sistema identifica que o ator possui dependência com um caso de uso e não permite a exclusão; 2. O caso de uso termina.
Pós-condição	O ator é cadastrado com sucesso.

3.3.5. Caso de uso: Passos

Caso de Uso	Cadastrar passo a passo
Ator	Analista de Requisitos
Descrição	Cadastrar a sequência lógica das informações que remetem ao caso de uso em questão.
Pré-condição	Sistema e caso de uso previamente cadastrados.
Cenário Principal	Este caso de uso inicia quando o sistema apresenta a opção de cadastro de passos. 1. O sistema recupera as informações de sistemas; 2. O Analista de Requisitos seleciona um sistema; 3. O sistema recupera as informações de caso de uso; 4. O Analista de Requisitos seleciona um caso de uso; 5. O Analista de Requisitos seleciona uma opção tipo de fluxo e preenche o ID e descrição; 6. O sistema recupera as informações complementares; [Fluxo alternativo 01: Inclusão de Informação Complementar] 7. O Analista de Requisitos seleciona uma informação complementar;

	8. O sistema recupera as regras de negócios; [Fluxo alternativo 02: Inclusão de Regra de Negócios] 9. O Analista de Requisitos seleciona uma regra de negócios; 10. O sistema recupera as referências; [Fluxo alternativo 03: Inclusão de Referências] 11. O Analista de Requisitos seleciona uma referência; 12. O Analista de Requisitos confirma a ação; 13. O sistema armazena as informações e apresenta um grid contemplando todos os passos cadastradas; [Fluxo alternativo 04: Edição de um passo] [Fluxo alternativo 05: Exclusão de um passo] 14. O caso de uso termina.
Cenário Alternativo	Fluxo alternativo 01: Inclusão de Informação Complementar. Este fluxo alternativo inicia quando o Analista de Requisitos seleciona a opção de cadastro de Informação Complementar. 1. O sistema apresenta o formulário para preenchimento para edição 2. O caso de uso retorna ao passo 2 do Fluxo Principal. Fluxo alternativo 02: Inclusão de Regra de Negócios Este fluxo alternativo inicia quando o Analista de Requisitos seleciona a opção de cadastro de regras de negócios. 1. O sistema apresenta o formulário para preenchimento; 2. O Analista de Requisitos informa o ID e descrição e confirma a ação; 3. O sistema armazena as informações; 4. O caso de uso termina. Fluxo alternativo 03: Inclusão de Referências Este fluxo alternativo inicia quando o Analista de Requisitos seleciona a opção de cadastro de referências. 1. O sistema apresenta o formulário para preenchimento; 2. O Analista de Requisitos informa o ID e descrição e confirma a ação; 3. O sistema armazena as informações; 4. O caso de uso termina. Fluxo alternativo 04: Edição de um passo

	Este fluxo alternativo inicia quando o Analista de Requisitos seleciona um passo na grid e a ação de edição. 5. O sistema apresenta o formulário com as informações preenchidas; 6. O Analista de Requisitos edita as informações; 7. O sistema armazena as alterações; 8. O caso de uso termina. <u>Fluxo alternativo 05:</u> Exclusão de um passo Este fluxo alternativo inicia quando o Analista de Requisitos seleciona um passo na grid e a ação de exclusão. 1. O sistema apresenta mensagem de confirmação de exclusão; 2. O Analista de Requisitos confirma a ação; 3. O sistema exclui as informações do passo; 4. O caso de uso termina.
Pós-condição	O passo é cadastrado com sucesso.

3.4. Vantagens da Aplicação

A aplicação apresenta um design intuitivo e de fácil uso, não requer instalação em *desktops* de usuários finais, é gratuita (*open source*) e pode ser utilizada inclusive para fins acadêmicos, como objeto de pesquisa.

3.5. Como Utilizar

3.5.1. Menu de acesso e dashboard

A aplicação possui opção de idiomas (inglês e português) localizado no canto superior direito, que pode ser escolhido pelo próprio usuário. Na lateral esquerda, segue o menu de navegação da aplicação.

A primeira funcionalidade é o painel (figura 34) onde o sistema exibe uma representação ilustrada dos casos de uso cadastrados no sistema, dividindo-os em ativo (cor azul) e inativo (cor vermelha).

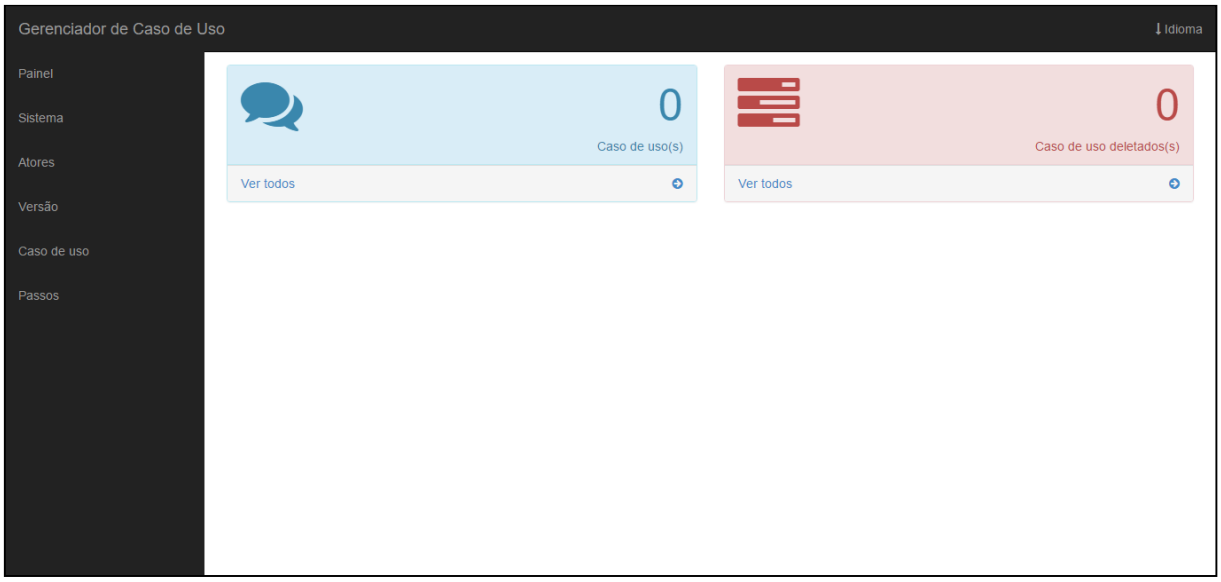

Figura 34 - Dashboard do sistema
(Fonte: *printscreen* do sistema)

3.5.2. Cadastro de sistema

No menu, ao escolher a opção sistema, será exibida a tela onde o usuário irá cadastrar o nome do sistema, conforme figura 35.

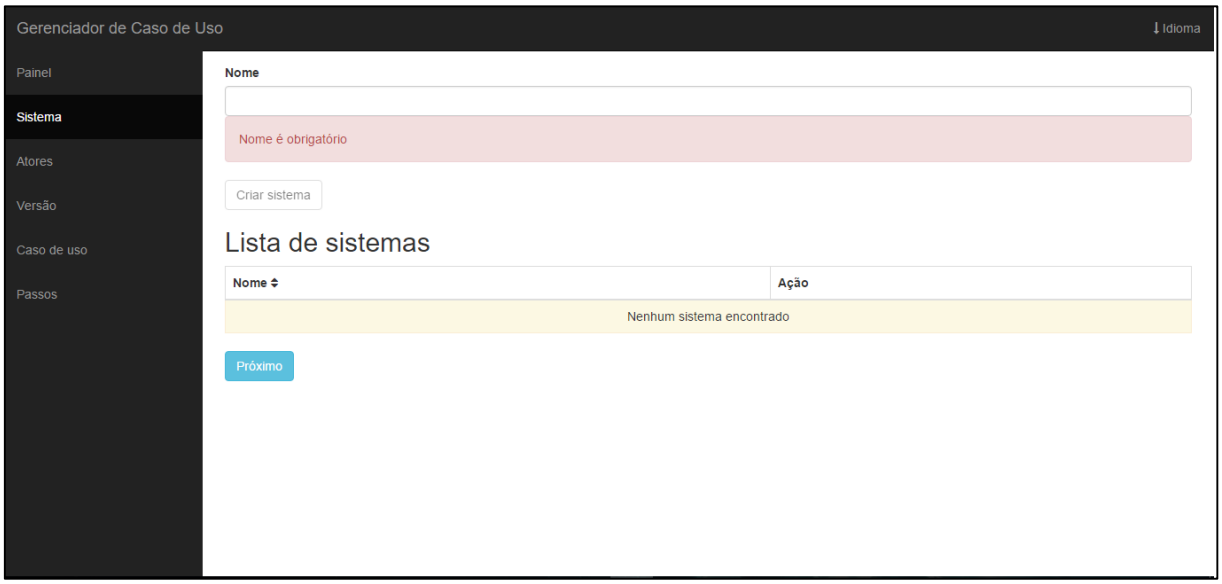

Figura 35 - Tela de cadastro de sistema
(Fonte: *printscreen* do sistema)

Em “Nome” deverá ser inserido o a nomenclatura que será dada a aplicação. Após o cadastro do nome, o sistema exibe um grid contendo todos os nomes já cadastrados.

3.5.3. Ator

Tela onde serão cadastrados os atores envolvidos no caso de uso. Entende-se por autor, algo que interaja com o sistema, mas sobre o qual não se tem controle (figura 36).

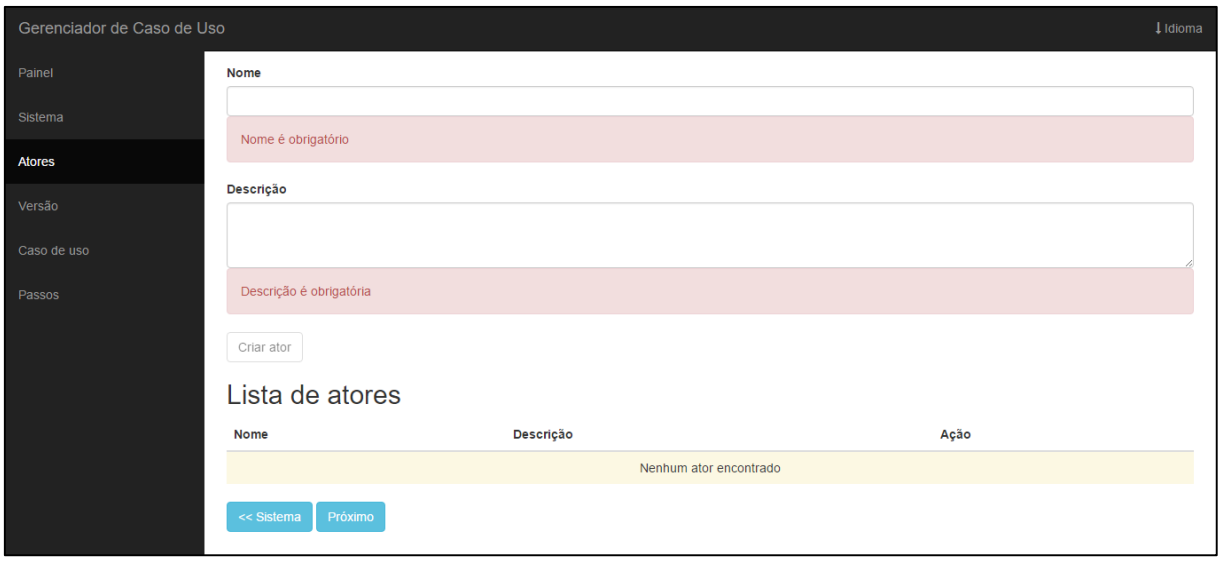

Figura 36 - Tela de cadastro de atores
(Fonte: *printscreen* do sistema)

No campo “Nome” deve ser informado um nome, seja um próprio ou mesmo cargo, para o ator. Já no campo “Descrição”, descrever quais serão os acessos desse ator. Por exemplo, o ator X irá interagir com os requisitos A,B,C; desta forma a compreensão sobre os limites de acesso ficarão mais claras.

3.5.4. Versão

Nesta tela será cadastrado a versão da aplicação (figura 37), bem como a descrição de quais alterações a compõem.

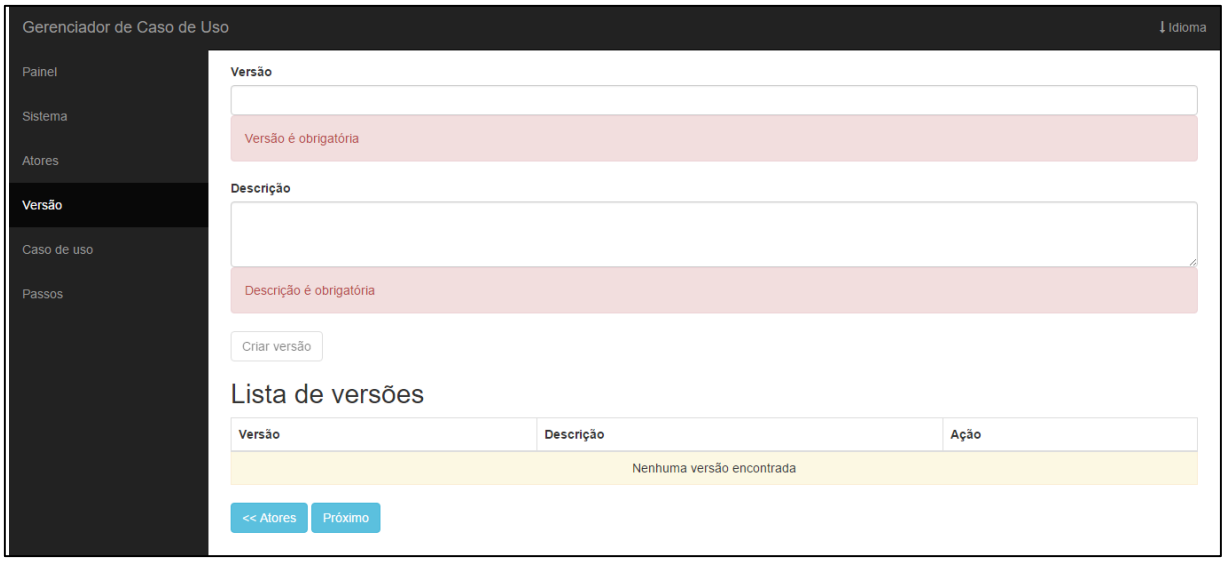

Figura 37 - Tela de cadastro da versão
(Fonte: *printscreen* do sistema)

No campo “Versão” deverá ser informado o número da versão que o sistema se encontra. Já no campo “Descrição” deverão ser informadas as alterações realizadas nesta versão. Ex.: Versão: 2.5

Descrição: Caso de uso Fornecedor atualizado segundo as novas regras de negócio.

Após o cadastro das informações, o sistema exibirá os resultados em uma grid abaixo da descrição.

3.5.5. Caso de uso

Nesta tela será permitido o cadastro de um caso de uso, associando-o a um sistema já incluído, conforme figura 38.

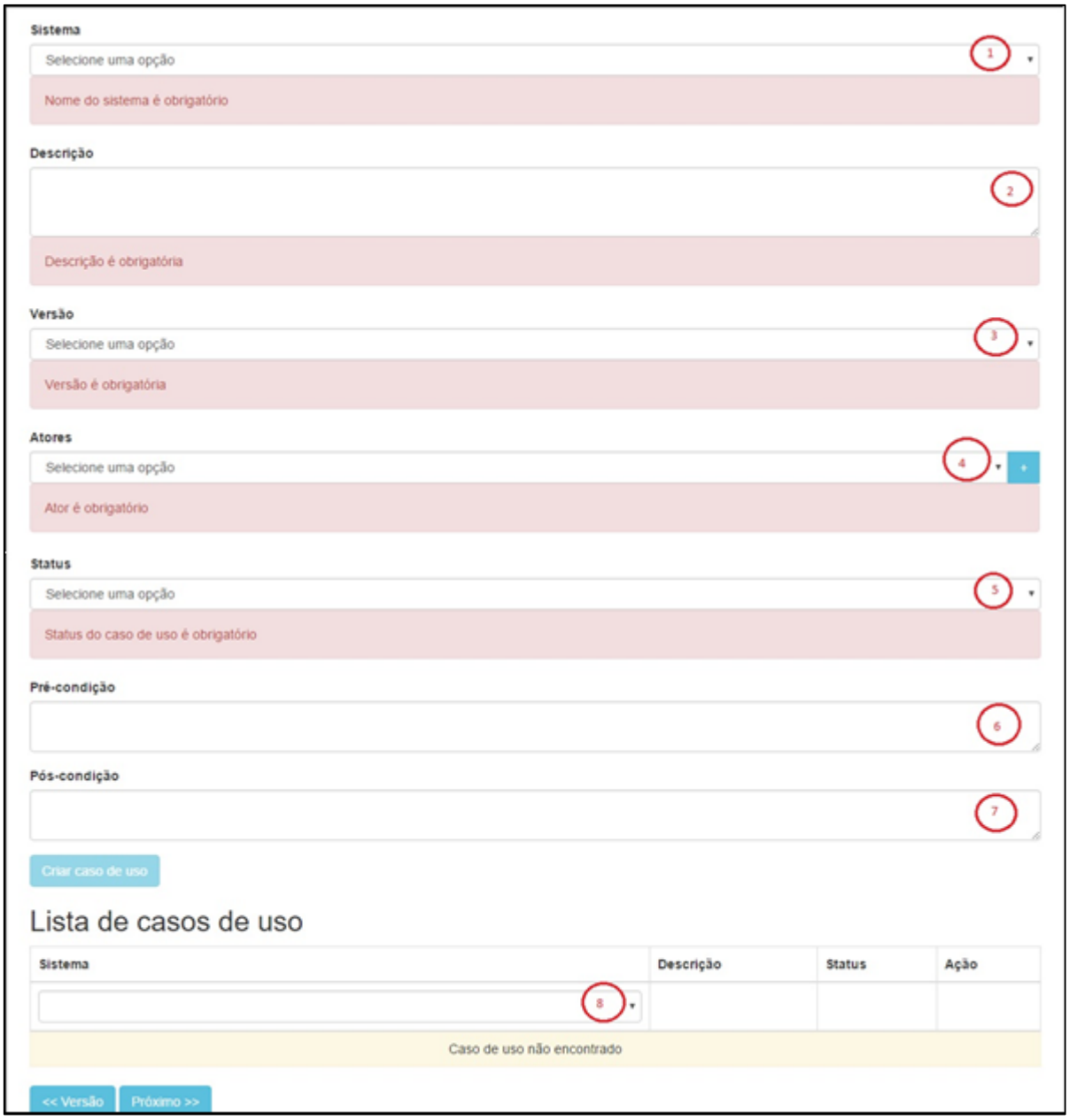

Figura 38 - Tela de cadastro de Caso de Uso
(Fonte: *printscreen* do sistema)

1 - No campo “Sistema” o usuário precisa selecionar, em uma combo box, um sistema que tenha sido cadastrado anteriormente.

2 - No campo “Descrição”, deverá ser descrita a finalidade do caso de uso. Lembrando que quanto maior o detalhamento, melhor será a compreensão da equipe que fará uso da informação cadastrada.

3 - No campo “versão” deverá ser escolhida por meio de combo box, uma versão já cadastrada anteriormente.

4 - No campo Atores, deverá ser escolhido um ator já cadastrado anteriormente. O sistema exibe as opções por meio de uma lista suspensa. Pode-se cadastrar mais

de um usuário ao clicar no quadrado azul com sinal de + ou excluir clicando no quadrado vermelho com sinal -.

5 - No campo “Status”, o sistema exibe uma lista contendo as seguintes opções:

a. Concluído: Caso de uso foi cadastrado e suas informações validadas;

b. Excluído: Caso de uso não será mais usado e será desconsiderado do sistema. O Sistema contabiliza e exibe o resultado na tela Painel como caso de uso deletado;

c. Desenvolvimento: Caso de Uso foi cadastrado, porém ainda não foi concluído;

d. Suspenso: Aguardando alguma análise mais detalhada para poder classifica-lo em outra categoria.

e. Inativo: Caso de uso não será usado naquele momento.

6 – No campo “Pré-condição”, deve-se listar as condições, premissas, que devem ser verificadas antes da execução deste caso de uso.

7 – No campo “Pós-condição”, descrever o estado do sistema após a execução do caso de uso.

8 – Após o preenchimento de todas as informações, ao clicar no botão Criar caso de uso, será possível consultar a informação na lista de casos de uso, mediante escolha do sistema que se deseja consultar.

3.5.6. Passos

Nesta tela será permitido o cadastro dos passos que compõem um caso de uso, conforme figura 39.

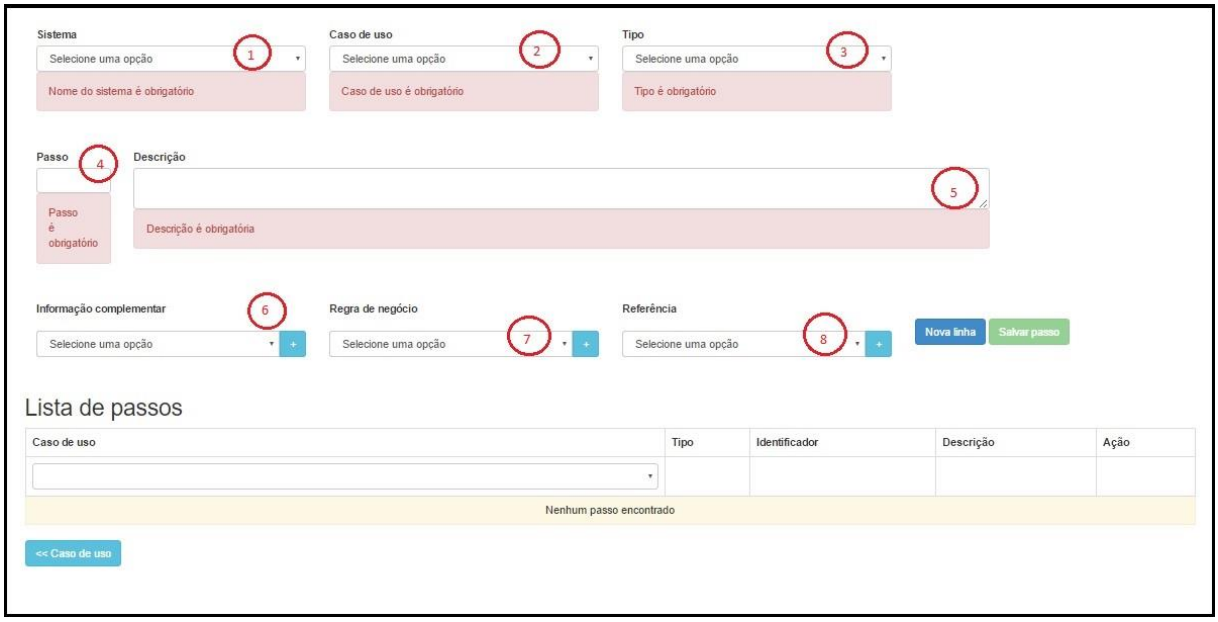

Figura 39 - Tela de cadastro de passos.
(Fonte: printscreen do sistema)

1 – Primeiro seleciona-se por meio de uma combo box um sistema cadastrado anteriormente;

2- Seleciona-se o caso de uso, por meio de uma combo, ao qual os passos serão associados;

3 - Seleciona-se o tipo de passo que será cadastrado entre as seguintes opções: básico, alternativo e exceção;

4 - No campo “Passo” deverá ser informado o código do passo;

5 - No campo “Descrição” deverá ser informado o nome do passo cadastrado;

6 - Ao clicar no botão “+” o sistema abrirá uma nova janela que permitirá o cadastro do identificador e sua descrição. Após salvar, será possível escolher a opção cadastrada por meio da caixa de seleção.

7 - No campo “Regras de negócio” o sistema permitirá o cadastramento de uma nova regra ao clicar no sinal de “+”. Após o cadastro, a informação pode ser selecionada por meio da caixa de seleção.

8 - O campo “Referência” pode ser referente a um requisito da regra de negócio ou mesmo um complemento.

3.6. Público Alvo

Fabrica funcional, equipes de projetos de qualquer segmento, além do que poderá servir de embasamento ou mesmo consulta acadêmica.

3.7. Mercado

Comparamos com duas ferramentas pagas, sendo elas das empresas Serena e IBM. A proposta é oferecer esta ferramenta como Open Source, sem custo para os utilizadores, disponibilizando assim uma opção para que empresas com baixo orçamento na área de TI possam documentar e gerenciar seus casos de uso e requisitos funcionais.

4. CONCLUSÃO

4.1. Considerações Iniciais

O trabalho aqui apresentado foi desenvolvido especialmente para a Engenharia de Requisitos, uma das áreas que compõem a engenharia de software. Optamos pelo enfoque na análise/obtenção de requisitos.

Observando as opções presentes no mercado e analisando a situação de algumas empresas, desenvolvemos uma ferramenta que pretende auxiliar no gerenciamento de casos de uso, sejam eles voltados para empresas do ramo de TI ou não.

4.2. Contribuições Obtidas

Por meio da documentação utilizada e os conhecimentos oriundos do curso, foi possível um aprofundamento na linguagem de modelagem universal, UML, no que se refere a coleta de requisitos para gerar essa ferramenta, que nos serviu de inspiração para desenvolvê-la de forma *open source*, ou seja, livre para a sua modificação e/ou utilização. Com isso certamente fomentaremos de uma maneira positiva a gerencia de requisitos contribuindo com os profissionais de TI e todos que busquem o gerenciamento de seus casos de uso.

4.3. Proposta para evolução de pesquisa

Esta primeira versão compreende necessidades básicas para atendimento inicial do gerenciamento de caso de uso.

Para a expansão da ferramenta é possível realizar: integração com outras aplicações, criação de um FAQ explicando o detalhamento de cada tela, novos filtros de busca, melhoria no designer gráfico da ferramenta, controle de acesso por meio de *login* e senha, além de desenvolver um gerador de diagrama de caso de uso a partir de dados existentes no sistema, sendo este um diferencial e atrativo para um número maior de adesões ao uso desta ferramenta.

4.4. Considerações Finais

O objetivo principal desse trabalho foi desenvolver uma ferramenta para auxiliar a gestão de requisitos, especificamente os casos de uso. Desenvolver tal ferramenta não foi tarefa fácil pois esse é um mercado dominado por ferramentas pagas que grandes corporações possuem.

O grupo teve que trazer todo o embasamento teórico existente na UML para chegar no resultado final que é a ferramenta. Exploramos diversos diagramas como o diagrama de atividade, comunicação, pacote entre outros.

Foi aplicado aqui o conhecimento adquirido ao longo das disciplinas desta pós-graduação, tendo como referencial o embasamento bibliográfico.

REFERÊNCIAS

BEZERRA, Eduardo. **Princípios de Análise e Projetos de Sistemas com UML**: um guia prático para modelagem de sistemas oa objetos através da Linguagem de Modelagem Unificada. 2.ed. Rio de Janeiro: Elsevier, 2006. 65-67 p.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar.**UML Guia do Usuário.** 2. ed Rio de Janeiro: Editora Campus, 2006. 411-412 p.

FERNANDES, Aguinaldo Aragon; ABREU, Vladimir Ferraz. **Implantado a Governança de TI da Estratégia à Gestão dos Processos e Serviços.** 3. ed. São Paulo: Brasport, 2012. 57; 476 p.

GUEDES, Gilleanes. **U.M.L 2 uma abordagem prática**. 2. ed. São Paulo: Novatec, 2010. 18; 45-47; 101-103; 111-116; 183-192; 242; 343 p.

GUEDES, Gilleanes. **U.M.L 2 uma abordagem prática**. 2. ed. São Paulo: Novatec, 2011. 230, 231; 270 p.

HAMILTON, Kim; MILES, Russ. **Learning UML 2.0**. 2. ed. O´Reilly, 2006. 15; 122; 289; 307 p.

INTERNATIONAL INSTITUTE OF BUSINESS (IIBA). **A Guia Corpo de Conhecimento de Análise de Negócios (Guia BABOK)**. 2. Ed. 2011. 52-60; 190-191 p.

PRESSMAN, Roger. **Software Engineering a practitioner's approach**. 7. ed. Boston: Mc Graw Hill, 2010. 57; 119-121 p.

SOMMERVILLE, Ian. **Engenharia de Software**. 6. ed. São Paulo: Addison Wesley, 2004.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. ed. São Paulo: Addison Wesley, 2008. 82-88 p.