UFES - Universidade Federal do Espírito Santo

Engenharia de Requisitos

Notas de Aula

Ricardo de Almeida Falbo

E-mail: falbo@inf.ufes.br

2017

Sumário

Capítulo 1 – Introdução

Sistemas de software são reconhecidamente importantes ativos estratégicos para diversas organizações. Uma vez que tais sistemas, em especial os sistemas de informação, têm um papel vital no apoio aos processos de negócio das organizações, é fundamental que os sistemas funcionem de acordo com os requisitos estabelecidos. Neste contexto, uma importante tarefa no desenvolvimento de software é a identificação e o entendimento dos requisitos dos negócios que os sistemas vão apoiar (AURUM; WOHLIN, 2005).

A Engenharia de Requisitos é o processo pelo qual os requisitos de um produto de software são coletados, analisados, documentados e gerenciados ao longo de todo o ciclo de vida do software (AURUM; WOHLIN, 2005). Este texto aborda o processo de Engenharia de Requisitos, concentrando-se nas atividades de levantamento de requisitos e modelagem conceitual.

Este capítulo apresenta o tema e como o mesmo é tratado neste texto. A Seção 1.1 discute a relação entre a Engenharia de Requisitos e o processo de software. A Seção 1.2 apresenta a organização deste texto.

1.1 – Desenvolvimento de Software e Engenharia de Requisitos

Um processo de software envolve diversas atividades que podem ser classificadas quanto ao seu propósito em:

- Atividades de Desenvolvimento (ou Técnicas): são as atividades diretamente relacionadas ao processo de desenvolvimento do software, ou seja, que contribuem diretamente para o desenvolvimento do produto de software a ser entregue ao cliente. São exemplos de atividades de desenvolvimento: levantamento e análise de requisitos, projeto e implementação.
- Atividades de Gerência: envolvem atividades relacionadas ao gerenciamento do projeto de maneira abrangente. Incluem, dentre outras: atividades de planejamento e acompanhamento gerencial do projeto (processo de Gerência de Projetos), tais como realização de estimativas, elaboração de cronogramas, análise dos riscos do projeto etc.; atividades relacionadas à gerência da evolução dos diversos artefatos produzidos nos projetos de software (processo de Gerência de Configuração); atividades relacionadas à gerência de ativos reutilizáveis de uma organização (processo de Gerência de Reutilização) etc.
- Atividades de Controle da Qualidade: são aquelas relacionadas com a avaliação da qualidade do produto em desenvolvimento e do processo de software utilizado. Incluem atividades de verificação, validação e garantia da qualidade.

As atividades de desenvolvimento formam a espinha dorsal do desenvolvimento e são realizadas segundo uma ordem estabelecida no planejamento. As atividades de gerência e de

controle da qualidade são, muitas vezes, ditas atividades de apoio, pois não estão ligadas diretamente à construção do produto final, ou seja, o software a ser entregue para o cliente, incluindo toda a documentação necessária. Essas atividades, normalmente, são realizadas ao longo de todo o ciclo de vida, sempre que necessário ou em pontos pré-estabelecidos durante o planejamento, ditos marcos ou pontos de controle. A Figura 1.1 mostra a relação entre esses tipos de atividades.

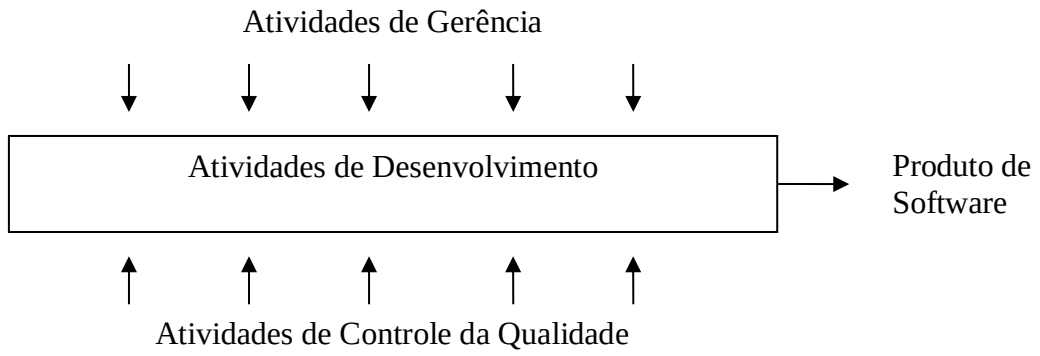


Figura 1.1 – Atividades do Processo de Software

No que concerne às atividades técnicas, tipicamente o processo de software inicia-se com o Levantamento de Requisitos, quando os requisitos do sistema a ser desenvolvido são preliminarmente capturados e organizados. Uma vez capturados, os requisitos devem ser modelados, avaliados e documentados. Uma parte essencial dessa fase é a elaboração de modelos descrevendo *o quê* o software tem de fazer (e não *como* fazê-lo), dita Modelagem Conceitual. Até este momento, a ênfase está sobre o domínio do problema e não se deve pensar na solução técnica, computacional a ser adotada.

Com os requisitos pelo menos parcialmente capturados e especificados na forma de modelos, pode-se começar a trabalhar no domínio da solução. Muitas soluções são possíveis para o mesmo conjunto de requisitos e elas são intrinsecamente ligadas a uma dada plataforma de implementação (linguagem de programação, mecanismo de persistência a ser adotado etc.). A fase de projeto tem por objetivo definir e especificar uma solução a ser implementada. É uma fase de tomada de decisão, tendo em vista que muitas soluções são possíveis.

Uma vez projetado o sistema, pode dar-se início à implementação, quando as unidades de software do projeto são implementadas e testadas individualmente. Gradativamente, os elementos vão sendo integrados e testados (teste de integração), até se obter o sistema, quando o todo deve ser testado (teste de sistema). Por fim, uma vez testado no ambiente de desenvolvimento, o software pode ser colocado em produção. Usuários devem ser treinados, o ambiente de produção deve ser configurado e o sistema deve ser instalado e testado, agora pelos usuários no ambiente de produção (testes de homologação ou aceitação). Caso o software demonstre prover as capacidades requeridas, ele pode ser aceito e a operação iniciada.

Requisitos têm um papel central no desenvolvimento de software, uma vez que uma a principal medida do sucesso de um software é o grau no qual ele atende aos objetivos e requisitos para os quais foi construído. Requisitos são a base para estimativas, modelagem, projeto, implementação, testes e até mesmo para a manutenção. Portanto, estão presentes ao longo de todo o ciclo de vida de um software.

Nos estágios iniciais de um projeto, requisitos têm de ser levantados, entendidos e documentados (atividades de Levantamento, Análise e Documentação de Requisitos). Dada a

importância dos requisitos para o sucesso de um projeto, atividades de controle da qualidade devem ser realizadas para verificar, validar e garantir a qualidade dos requisitos, uma vez que os custos serão bem maiores se defeitos em requisitos forem identificados tardiamente. Mesmo quando coletados de forma sistemática, requisitos mudam. Os negócios são dinâmicos e não há como garantir que os requisitos não sofrerão alterações. Assim, é fundamental gerenciar a evolução dos requisitos, bem como manter a rastreabilidade entre os requisitos e os demais artefatos produzidos no projeto (atividade de Gerência de Requisitos).

Como se pode observar, o tratamento de requisitos envolve atividades de desenvolvimento (Levantamento, Análise e Documentação de Requisitos), gerência (Gerência de Requisitos) e controle da qualidade (Verificação, Validação e Garantia da Qualidade de Requisitos). Ao conjunto de atividades relacionadas a requisitos, dá-se o nome de Processo de Engenharia de Requisitos.

1.2 - A Organização deste Texto

Este texto procura oferecer uma visão geral da Engenharia de Requisitos de Software, discutindo as principais atividades desse processo e como realizá-las. Ênfase especial é dada à aplicação das técnicas de modelagem e especificação de Sistemas de Informação, i.e., sistemas desenvolvidos para apoiar processos de negócio das organizações.

Nos capítulos que se seguem, os seguintes temas são abordados:

- Capítulo 2 – *Engenharia de Requisitos de Software*: inicia discutindo o que são requisitos e tipos e níveis de requisitos. A seguir, apresenta o processo de engenharia de requisitos considerado neste texto, o qual contém as seguintes atividades: Levantamento de Requisitos, Análise de Requisitos, Documentação de Requisitos, Verificação e Validação de Requisitos e Gerência de Requisitos. Discute-se, também, como as principais normas e modelos de qualidade de processos de software tratam a questão dos requisitos.
- Capítulo 3 – *Levantamento de Requisitos*: provê uma visão geral do processo de levantamento de requisitos e aborda técnicas para levantar requisitos, dentre elas entrevistas, questionários, observação, investigação de documentos e prototipagem. Aborda-se, também, a modelagem de processos de negócio e como ela pode apoiar o levantamento de requisitos. Finalmente, discute-se como escrever e documentar requisitos de cliente.
- Capítulo 4 – *Análise de Requisitos*: trata da análise de requisitos, discutindo a análise e especificação de requisitos funcionais (modelagem conceitual) e não funcionais. O capítulo inicia provendo uma introdução à modelagem conceitual. Na sequência é apresentada brevemente a Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML), amplamente usada na Análise de Requisitos e os conceitos da orientação a objetos, paradigma adotado neste texto. Um método de análise de requisitos funcionais é apresentado. A seguir, discute-se a especificação de requisitos não funcionais. Por fim, a documentação da atividade de análise de requisitos é abordada.
- Capítulo 5 – *Modelagem de Objetivos e de Casos de Uso*: discute a análise de objetivos como uma ferramenta para apoiar a identificação de requisitos e as razões que os justificam, e o papel da modelagem de casos de uso na especificação de

requisitos funcionais de sistema. Apresenta os elementos centrais da modelagem de casos de uso, o diagrama de casos de uso da UML e discute como descrever casos de uso textualmente.

- Capítulo 6 – *Modelagem Estrutural*: trata da modelagem dos principais conceitos do domínio, suas relações e propriedades, permitindo representar o conhecimento do domínio relevante para o sistema em desenvolvimento. A elaboração de diagramas de classes da UML é o foco deste capítulo.
- Capítulo 7 – *Modelagem Dinâmica*: aborda os modelos usados para especificar as mudanças válidas no estado dos objetos de domínio, bem como para modelar o comportamento esperado do sistema. O foco deste capítulo é a elaboração de diagramas de estados e diagramas de atividades.
- Capítulo 8 – *Qualidade e Agilidade em Requisitos*: explora técnicas de leitura de modelos, visando apoiar a verificação da consistência entre os diversos artefatos produzidos durante a análise de requisitos; apresenta o enfoque da modelagem ágil; e discute abordagens para reutilização na Engenharia de Requisitos, dentre elas Engenharia de Domínio, Ontologias e Padrões de Análise.

Referências do Capítulo

AURUM, A., WOHLIN, C., *Engineering and Managing Software Requirements*, Springer-Verlag, 2005.

Capítulo 2 – Engenharia de Requisitos de Software

Requisitos têm um papel central no processo de software, sendo considerados um fator determinante para o sucesso ou fracasso de um projeto de software. O processo de levantar, analisar, documentar, gerenciar e controlar a qualidade dos requisitos é chamado de Engenharia de Requisitos.

Este capítulo dá uma visão geral da Engenharia de Requisitos. A Seção 2.1 apresenta diferentes definições do conceito de requisito e trata de tipos e níveis de requisitos. A Seção 2.2 aborda o processo de engenharia de requisitos, discutindo cada uma de suas atividades. Finalmente, a Seção 2.3 discute como as principais normas e modelos de qualidade de processos de software tratam a questão dos requisitos.

2.1 – Requisitos

Existem diversas definições para requisito de software na literatura, dentre elas:

- Requisitos são descrições dos serviços que devem ser providos pelo sistema e de suas restrições operacionais (SOMMERVILLE, 2007).
- Um requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir seus objetivos (PFLEEGER, 2004).
- Um requisito é alguma coisa que o produto tem de fazer ou uma qualidade que ele precisa apresentar (ROBERTSON; ROBERTSON, 2006).

Com base nessas e em outras definições, pode-se dizer que os requisitos de um sistema incluem especificações dos serviços que o sistema deve prover, restrições sob as quais ele deve operar, propriedades gerais do sistema e restrições que devem ser satisfeitas no seu processo de desenvolvimento.

As várias definições acima apresentadas apontam para a existência de diferentes tipos de requisitos. Uma classificação amplamente aceita quanto ao tipo de informação documentada por um requisito faz a distinção entre requisitos funcionais e requisitos não funcionais.

- **Requisitos Funcionais**: são declarações de serviços que o sistema deve prover, descrevendo o que o sistema deve fazer (SOMMERVILLE, 2007). Um requisito funcional descreve uma interação entre o sistema e o seu ambiente (PFLEEGER, 2004), podendo descrever, ainda, como o sistema deve reagir a entradas específicas, como o sistema deve se comportar em situações específicas e o que o sistema não deve fazer (SOMMERVILLE, 2007).
- **Requisitos Não Funcionais**: descrevem restrições sobre os serviços ou funções oferecidos pelo sistema (SOMMERVILLE, 2007), as quais limitam as opções para criar uma solução para o problema (PFLEEGER, 2004). Neste sentido, os requisitos não funcionais são muito importantes para a fase de projeto (design), servindo como base para a tomada de decisões nessa fase.

Os requisitos não funcionais têm origem nas necessidades dos usuários, em restrições de orçamento, em políticas organizacionais, em necessidades de interoperabilidade com outros sistemas de software ou hardware ou em fatores externos como regulamentos e legislações (SOMMERVILLE, 2007). Assim, os requisitos não funcionais podem ser classificados quanto à sua origem. Existem diversas classificações de requisitos não funcionais. Sommerville (2007), por exemplo, classifica-os em:

- **Requisitos de produto**: especificam o comportamento do produto (sistema). Referem-se a atributos de qualidade que o sistema deve apresentar, tais como confiabilidade, usabilidade, eficiência, portabilidade, manutenibilidade e segurança.
- **Requisitos organizacionais**: são derivados de metas, políticas e procedimentos das organizações do cliente e do desenvolvedor. Incluem requisitos de processo (padrões de processo e modelos de documentos que devem ser usados), requisitos de implementação (tal como a linguagem de programação a ser adotada), restrições de entrega (tempo para chegar ao mercado - *time to market*, restrições de cronograma etc.), restrições orçamentárias (custo, custo-benefício) etc.
- **Requisitos externos**: referem-se a todos os requisitos derivados de fatores externos ao sistema e seu processo de desenvolvimento. Podem incluir requisitos de interoperabilidade com sistemas de outras organizações, requisitos legais (tais como requisitos de privacidade) e requisitos éticos.

No que se refere aos RNFs de produto, eles podem estar relacionados a propriedades emergentes do sistema como um todo, ou seja, propriedades que não podem ser atribuídas a uma parte específica do sistema, mas que, ao contrário, só aparecem após a integração de seus componentes, tal como confiabilidade (SOMMERVILLE, 2007). Contudo, algumas vezes, essas características podem estar associadas a uma função específica ou a um conjunto de funções. Por exemplo, uma certa função pode ter restrições severas de desempenho, enquanto outras funções do mesmo sistema podem não apresentar tal restrição.

Os requisitos devem ser redigidos de modo a serem passíveis de entendimento pelos diversos interessados (*stakeholders*). Clientes[1], usuários finais e desenvolvedores são todos interessados em requisitos, mas têm expectativas diferentes. Enquanto desenvolvedores e usuários finais têm interesse em detalhes técnicos, clientes requerem descrições mais abstratas. Assim, é útil apresentar requisitos em diferentes níveis de descrição. Sommerville (2007) sugere dois níveis de descrição de requisitos:

- **Requisitos de Cliente ou de Usuário:** são declarações em linguagem natural acompanhadas de diagramas intuitivos de quais serviços são esperados do sistema e das restrições sob as quais ele deve operar. Devem estar em um nível de abstração mais alto, de modo que sejam compreensíveis pelos clientes e usuários do sistema que não possuem conhecimento técnico.

[1] É importante notar a distinção que se faz aqui entre clientes e usuários finais. Consideram-se clientes aqueles que contratam o desenvolvimento do sistema e que, muitas vezes, não usarão diretamente o sistema. Eles estão mais interessados nos resultados da utilização do sistema pelos usuários do que no sistema em si. Usuários, por outro lado, são as pessoas que utilizarão o sistema em seu dia a dia. Ou seja, os usuários são as pessoas que vão operar ou interagir diretamente com o sistema.

- **Requisitos de Sistema:** definem detalhadamente as funções, serviços e restrições do sistema. São versões expandidas dos requisitos de cliente usados pelos desenvolvedores para projetar, implementar e testar o sistema. Como requisitos de sistema são mais detalhados, as especificações em linguagem natural são insuficientes e para especificá-los, notações mais especializadas devem ser utilizadas.

Vale destacar que esses níveis de descrição de requisitos são aplicados em momentos diferentes e com propósitos distintos. Requisitos de cliente são elaborados nos estágios iniciais do desenvolvimento (levantamento preliminar de requisitos) e servem de base para um entendimento entre clientes e desenvolvedores acerca do que o sistema deve contemplar. Esses requisitos são, normalmente, usados como base para a contratação e o planejamento do projeto. Requisitos de sistema, por sua vez, são elaborados como parte dos esforços diretos para o desenvolvimento do sistema, capturando detalhes importantes para as fases técnicas posteriores do processo de desenvolvimento, a saber: projeto, implementação e testes.

Entretanto, não se deve perder de vista que requisitos de sistema são derivados dos requisitos de cliente. Os requisitos de sistema acrescentam detalhes, explicando os serviços e funções a serem providos pelo sistema em desenvolvimento. Os interessados nos requisitos de sistema necessitam conhecer mais precisamente o que o sistema fará, pois eles estão preocupados com o modo como o sistema apoiará os processos de negócio ou porque estão envolvidos na sua construção (SOMMERVILLE, 2007).

Uma vez que requisitos de cliente e de sistema têm propósitos e público alvo diferentes, é útil descrevê-los em documentos diferentes. Pfleeger (2004) sugere que dois tipos de documentos de requisitos sejam elaborados:

- **Documento de Definição de Requisitos**: deve ser escrito de maneira que o cliente possa entender, i.e., na forma de uma listagem do quê o cliente espera que o sistema proposto faça. Ele representa um consenso entre o cliente e o desenvolvedor sobre o quê o cliente quer.
- **Documento de Especificação de Requisitos**: refina os requisitos de cliente em termos mais técnicos, apropriados para o desenvolvimento de software, sendo produzido por analistas de requisitos.

Vale ressaltar que deve haver uma correspondência direta entre cada requisito de usuário listado no documento de requisitos e os requisitos de sistema tratados no documento de especificação de requisitos.

2.2 – O Processo de Engenharia de Requisitos

A Engenharia de Requisitos (ER) é o ramo da Engenharia de Software que envolve as atividades relacionadas com a definição dos requisitos de software de um sistema, desenvolvidas ao longo do ciclo de vida de software (KOTONYA; SOMMERVILLE, 1998). O processo de ER envolve criatividade, interação entre pessoas, conhecimento e experiência para transformar informações diversas (sobre a organização, sobre leis, sobre o sistema a ser construído etc.) em documentos e modelos que direcionem o desenvolvimento de software (KOTONYA; SOMMERVILLE, 1998).

A Engenharia de Requisitos é fundamental, pois possibilita, dentre outros, estimar custo e tempo de maneira mais precisas e melhor gerenciar mudanças em requisitos. Dentre os problemas de um processo de engenharia de requisitos ineficiente, podem-se citar (KOTONYA; SOMMERVILLE, 1998): (i) requisitos inconsistentes, (ii) produto final com custo maior do que o esperado, (iii) software instável e com altos custos de manutenção e (iv) clientes insatisfeitos.

A Engenharia de Requisitos pode ser descrita como um processo, ou seja, um conjunto organizado de atividades que deve ser seguido para derivar, avaliar e manter os requisitos e artefatos relacionados. Uma descrição de um processo, de forma geral, deve incluir, além das atividades a serem seguidas, a estrutura ou sequência dessas atividades, quem é responsável por cada atividade, suas entradas e saídas, as ferramentas usadas para apoiar as atividades e os métodos, técnicas e diretrizes a serem seguidos na sua realização.

Processos de ER podem variar muito de uma organização para outra, ou até mesmo dentro de uma organização específica, em função de características dos projetos. A definição de um processo apropriado à organização traz muitos benefícios, pois uma boa descrição do mesmo fornecerá orientações e reduzirá a probabilidade de esquecimento ou de uma execução superficial. No entanto, não faz sentido falar em processo ideal ou definir algum e impô-lo a uma organização. Ao invés disto, as organizações devem iniciar com um processo genérico e adaptá-lo para um processo mais detalhado, que seja apropriado às suas reais necessidades (SOMMERVILLE; SAWYER, 1997). Assim, a implantação de processos em uma organização deve ser feita segundo as necessidades da mesma, ou seja, o processo deve ser definido de acordo com as características da organização. Existem, portanto, fatores que contribuem para a variabilidade do processo de ER, dentre eles a maturidade técnica, o envolvimento disciplinar, a cultura organizacional e os domínios de aplicação nos quais a organização atua (KOTONYA; SOMMERVILLE, 1998).

Wiegers (2003) destaca alguns benefícios que um processo de ER de qualidade pode trazer, dentre elas: menor quantidade de defeitos nos requisitos, redução de retrabalho, desenvolvimento de menos características desnecessárias, diminuição de custos, desenvolvimento mais rápido, menos problemas de comunicação, alterações de escopo reduzidas, estimativas mais confiáveis e maior satisfação de clientes e desenvolvedores.

Ainda que diferentes projetos requeiram processos com características específicas para contemplar suas peculiaridades, é possível estabelecer um conjunto de atividades básicas que deve ser considerado na definição de um processo de ER. Tomando por base o processo proposto por Kotonya e Sommerville (1998), neste texto considera-se que um processo de ER deve contemplar, tipicamente, as atividades mostradas na Figura 2.1: levantamento de requisitos, análise de requisitos, documentação de requisitos, verificação e validação de requisitos e gerência de requisitos.

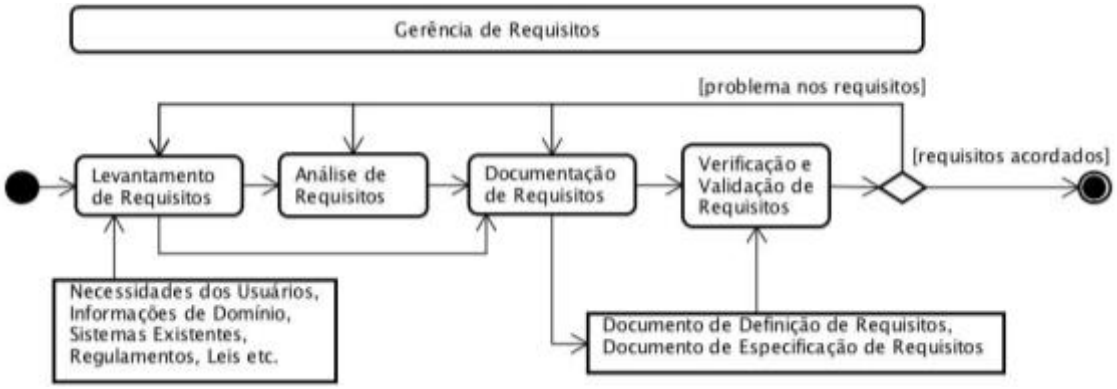


Figura 2.1 – Processo de Engenharia de Requisitos (adaptado de (KOTONYA; SOMMERVILLE, 1998))

O processo começa pelo levantamento de requisitos, que deve levar em conta necessidades dos usuários e clientes, informações de domínio, sistemas existentes, regulamentos, leis etc. Uma vez identificados requisitos, é possível iniciar a atividade de análise, quando os requisitos levantados são usados como base para a modelagem do sistema. Tanto no levantamento quanto na análise de requisitos, é importante documentar requisitos e modelos. Conforme discutido anteriormente, para documentar requisitos, dois documentos são normalmente utilizados: o Documento de Definição de Requisitos, contendo uma lista dos requisitos de cliente identificados, e o Documento de Especificação de Requisitos, que registra os requisitos de sistema e os vários diagramas resultantes do trabalho de análise. Os documentos produzidos são, então, verificados e validados. Adicionalmente, um esforço de garantia da qualidade deve ser realizado, visando garantir conformidade em relação a padrões e ao processo estabelecidos pela organização. Caso clientes, usuários e desenvolvedores estejam de acordo com os requisitos, o processo de desenvolvimento pode avançar; caso contrário, deve-se retornar à atividade correspondente para resolver os problemas identificados. Em paralelo a todas as atividades anteriormente mencionadas, há a gerência de requisitos, que se ocupa em gerenciar mudanças nos requisitos.

Vale destacar que não há limites bem definidos entre as atividades acima citadas. Na prática, elas são intercaladas e existe um alto grau de iteração e feedback entre elas. Idealmente, deve-se começar com um levantamento preliminar de requisitos que considere apenas requisitos de cliente. Esses requisitos devem ser documentados em um Documento de Definição de Requisitos e avaliados (verificação e validação). Esse documento deve ser usado como base para a contratação do projeto. Caso haja acordo em relação aos requisitos de cliente, pode-se iniciar um ciclo de levantamento detalhado de requisitos e análise. O processo é executado até que todos os usuários estejam satisfeitos e concordem com os requisitos ou até que a pressão do cronograma precipite o início da fase de projeto, o que é indesejável (KOTONYA; SOMMERVILLE, 1998).

Além disso, ao se adotar um modelo de ciclo de vida iterativo, essas atividades podem ser realizadas muitas vezes. Neste caso, uma vez contratado o projeto e, portanto, obtido um acordo em relação aos requisitos de cliente iniciais, pode-se iniciar um ciclo de levantamento detalhado de requisitos e análise, produzindo uma versão do Documento de Especificação de Requisitos. Havendo acordo em relação aos requisitos e modelos contidos nessa primeira versão do documento, o desenvolvimento pode prosseguir para a

porção tratada nessa iteração (projeto, implementação e testes), enquanto um novo ciclo de levantamento e análise se inicia para tratar outros requisitos de cliente ainda não contemplados.

A seguir, as atividades do processo de ER proposto são discutidas com um pouco mais de detalhes.

2.2.1 – Levantamento de Requisitos

O levantamento de requisitos corresponde à fase inicial do processo de ER e envolve atividades de descoberta dos requisitos. Nessa fase, um esforço conjunto de clientes, usuários e especialistas de domínio é necessário, com o objetivo de entender a organização, seus processos, necessidades, deficiências dos sistemas de software atuais, possibilidades de melhorias, bem como restrições existentes. Trata-se de uma atividade complexa que não se resume somente a perguntar às pessoas o que elas desejam, mas sim analisar cuidadosamente a organização, o domínio da aplicação e os processos de negócio no qual o sistema será utilizado (KOTONYA; SOMMERVILLE, 1998).

Para levantar quais são os requisitos de um sistema, devem-se obter informações dos interessados (*stakeholders*), consultar documentos, obter conhecimentos do domínio e estudar o negócio da organização. Neste contexto, quatro dimensões devem ser consideradas, como ilustra a Figura 2.2 (KOTONYA; SOMMERVILLE, 1998):

Figura 2.2 - Dimensões do levantamento de requisitos

- **Entendimento do domínio da aplicação:** entendimento geral da área na qual o software a ser desenvolvido está inserido;
- **Entendimento do problema:** entendimento dos detalhes do problema específico a ser resolvido com o auxílio do sistema a ser desenvolvido;
- **Entendimento do negócio:** entender como o sistema afetará a organização e como contribuirá para que os objetivos do negócio e os objetivos gerais da organização sejam atingidos;

- **Entendimento das necessidades e das restrições dos interessados:** entender as demandas de apoio para a realização do trabalho de cada um dos interessados no sistema, entender os processos de trabalho a serem apoiados pelo sistema e o papel de eventuais sistemas existentes na execução e condução dos processos de trabalho. Consideram-se interessados no sistema, todas as pessoas que são afetadas pelo sistema de alguma maneira, dentre elas clientes, usuários finais e gerentes de departamentos onde o sistema será instalado.

A atividade de levantamento de requisitos é dominada por fatores humanos, sociais e organizacionais e envolve pessoas com diferentes conhecimentos e objetivos, o que a torna complexa. Christel e Kang (apud PRESSMAN, 2006) citam alguns problemas que tornam o levantamento de requisitos uma tarefa difícil:

- Problemas de escopo: as fronteiras do sistema são mal definidas ou os clientes/usuários especificam detalhes técnicos desnecessários que podem confundir, em vez de esclarecer, os objetivos globais do sistema.
- Problemas de entendimento: Os clientes/usuários não estão completamente certos do que é necessário, têm pouca compreensão das capacidades e limitações de um sistema computacional, não têm pleno entendimento do domínio do problema, têm dificuldade de comunicar suas necessidades, omitem informação que acreditam ser óbvia, especificam requisitos que conflitam com as necessidades de outros clientes/usuários ou especificam requisitos que são ambíguos ou impossíveis de testar.
- Problemas de volatilidade: Os requisitos mudam ao longo do tempo.

Kotonya e Sommerville (1998) destacam outras dificuldades que complementam e reforçam os problemas apontados por Christel e Kang, a saber:

- Pode ser difícil compreender e coletar informações quando existem muitos termos desconhecidos, manuais técnicos etc.
- Pessoas que entendem o problema a ser resolvido podem ser muito ocupadas e não ter muito tempo para, juntamente como analista, levantar os requisitos e entender o sistema.
- Políticas organizacionais podem influenciar nos requisitos de um sistema.
- Os interessados não sabem muito bem o que querem do sistema e não conhecem muitos termos.

Diversas técnicas podem ser utilizadas no levantamento de requisitos, as quais podem possuir diferentes objetos de investigação ou podem ter foco em tipos diferentes de requisitos. Assim, é útil empregar várias dessas técnicas concomitantemente, de modo a se ter um levantamento de requisitos mais eficaz. Dentre as várias técnicas, podem ser citadas (KENDALL; KENDALL, 2010; KOTONYA; SOMMERVILLE, 1998; AURUM; WOHLIN, 2005):

- Entrevistas: técnica amplamente utilizada, que consiste em conversas direcionadas com um propósito específico e com formato “pergunta-resposta”. Seu objetivo é descobrir problemas a serem tratados, levantar procedimentos importantes e saber a opinião e as expectativas do entrevistado sobre o sistema.

- Questionários: o uso de questionários possibilita ao analista obter informações como postura, crenças, comportamentos e características de várias pessoas que serão afetas pelo sistema.
- Observação: consiste em observar o comportamento e o ambiente dos indivíduos de vários níveis organizacionais. Utilizando-se essa técnica, é possível capturar o que realmente é feito e qual tipo de suporte computacional é realmente necessário. Ajuda a confirmar ou refutar informações obtidas com outras técnicas e ajuda a identificar tarefas que podem ser automatizadas e que não foram identificadas pelos interessados.
- Análise de documentos: pela análise de documentos existentes na organização, analistas capturam informações e detalhes difíceis de conseguir por entrevista e observação. Documentos revelam um histórico da organização e sua direção.
- Cenários: com o uso desta técnica, um cenário de interação entre o usuário final e o sistema é montado e o usuário simula sua interação com o sistema nesse cenário, explicando ao analista o que ele está fazendo e de que informações ele precisa para realizar a tarefa descrita no cenário. O uso de cenários ajuda a entender requisitos, a expor o leque de possíveis interações e a revelar facilidades requeridas.
- Prototipagem: um protótipo é uma versão preliminar do sistema, muitas vezes não operacional e descartável, que é apresentada ao usuário para capturar informações específicas sobre seus requisitos de informação, observar reações iniciais e obter sugestões, inovações e informações para estabelecer prioridades e redirecionar planos.
- Dinâmicas de Grupo: há várias técnicas de levantamento de requisitos que procuram explorar dinâmicas de grupo para a descoberta e o desenvolvimento de requisitos, tais como *Brainstorming* e JAD (*Joint Application Development*). Na primeira, representantes de diferentes grupos de interessados engajam-se em uma discussão informal para rapidamente gerarem o maior número possível de ideias. Na segunda, interessados e analistas se reúnem para discutir problemas a serem solucionados e soluções possíveis. Com as diversas partes envolvidas representadas, decisões podem ser tomadas e questões podem ser resolvidas mais rapidamente. A principal diferença entre JAD e *Brainstorming* é que, em JAD, tipicamente os objetivos do sistema já foram estabelecidos antes dos interessados participarem. Além disso, sessões JAD são normalmente bem estruturadas, com passos, ações e papéis de participantes definidos.

2.2.2 – Análise de Requisitos

Uma vez preliminarmente identificados os requisitos, é possível iniciar a atividade de análise, quando os requisitos levantados devem ser refinados. Neste contexto, modelos são essenciais e diversos tipos de modelos podem ser utilizados. Esses modelos são representações gráficas que descrevem objetivos e processos de negócio, o problema a ser resolvido e o sistema a ser desenvolvido.

De maneira simples, um modelo é uma simplificação da realidade enfocando certos aspectos considerados relevantes segundo a perspectiva do modelo, e omitindo os

demais. Modelos são construídos para se obter uma melhor compreensão da porção da realidade sendo modelada.

Modelos são úteis para a Engenharia de Requisitos (e, de maneira mais geral, para a Engenharia de Software) por vários motivos, dentre eles (LAMSWEERDE, 2009) (PRESSMAN, 2006):

- Modelos permitem enfocar os aspectos chave, em detrimento de detalhes irrelevantes.
- Ajudam o engenheiro de requisitos a entender a informação, a função e o comportamento do sistema, tornando a tarefa de análise de requisitos mais fácil e sistemática.
- Proveem apoio para o entendimento e explanação para os envolvidos.
- Tornam-se o ponto focal para a revisão e, portanto, a chave para a determinação da consistência e correção da especificação. Deste modo, apoiam a análise, detecção e reparo de erros de forma precoce.
- Servem de base para a tomada de decisão.
- Fornecem uma estrutura para as atividades da ER, provendo um alvo para o que deve ser obtido, avaliado, especificado, consolidado e modificado, sendo a base para a geração do Documento de Especificação de Requisitos.

Em essência, a fase de análise é uma atividade de modelagem. A modelagem nesta fase é dita conceitual, pois ela se preocupa com o domínio do problema e não com soluções técnicas para o mesmo. Os modelos de análise são elaborados para se obter uma compreensão maior acerca do sistema a ser desenvolvido e para especificá-lo. Diferentes modelos podem ser construídos para representar diferentes perspectivas. Classicamente, há duas principais perspectivas que são consideradas na fase de análise:

- Perspectiva estrutural: busca modelar os conceitos, propriedades e relações do domínio que são relevantes para o sistema em desenvolvimento. Provê uma visão estática das informações que o sistema necessita tratar e, portanto, refere-se às representações que o sistema terá de prover para abstrair entidades do mundo real. Em outras palavras, esta perspectiva foca em "sobre o quê?" (quais informações?) o sistema opera. Diagramas de classes e modelos de entidades e relacionamentos são usados para modelar esta perspectiva.
- Perspectiva comportamental: visa modelar o comportamento geral do sistema, de suas funcionalidades ou de uma entidade específica ao longo do tempo. Provê uma visão do comportamento do sistema ou de uma parcela do sistema. De maneira bem simples, o foco dessa perspectiva é no "quê" o sistema deve fazer (que funções ou serviços ele deve prover e como ele deve se comportar). Diagramas de casos de uso, diagramas de atividades, diagramas de estados e diagramas de interação são usados para modelar essa visão.

Contudo, outras perspectivas podem ser alvo de modelos. A abordagem de Engenharia de Requisitos Baseada em Objetivos (*Goal-Oriented Requirements Engineering* - GORE), p.ex., assume que objetivos são uma perspectiva fundamental, pois estabelecem o "porquê" do sistema (e, portanto, dos elementos identificados em outras perspectivas). Na abordagem GORE, as razões para um novo sistema (ou uma nova

versão de um sistema) precisam ser explicitadas em termos de objetivos a serem satisfeitos por ele (LAMSWEERDE, 2009) e, para tal, modelos de objetivos devem ser desenvolvidos.

A análise de requisitos é uma atividade extremamente vinculada ao levantamento de requisitos. Durante o levantamento de requisitos, alguns problemas são identificados e tratados. Entretanto, determinados problemas somente são identificados por meio de uma análise mais detalhada. A análise de requisitos ajuda a entender e detalhar os requisitos levantados, a descobrir problemas nesses requisitos e a obter a concordância sobre as alterações, de modo a satisfazer a todos os envolvidos. Seu objetivo é estabelecer um conjunto acordado de requisitos completos, consistentes e sem ambiguidades, que possa ser usado como base para as demais atividades do processo de desenvolvimento de software (KOTONYA; SOMMERVILLE, 1998).

De forma resumida, pode-se dizer que a análise atende a dois propósitos principais: (i) prover uma base para o entendimento e concordância entre clientes e desenvolvedores sobre o que o sistema deve fazer e (ii) prover uma especificação que guie os desenvolvedores na demais etapas do desenvolvimento, sobretudo no projeto, implementação e testes do sistema (PFLEEGER, 2004).

O processo de ER é dominado por fatores humanos, sociais e organizacionais. Ele envolve pessoas de diferentes áreas de conhecimento e com objetivos individuais e organizacionais diferentes. Dessa forma, é comum que cada indivíduo tente influenciar os requisitos para que seu objetivo seja alcançado, sem necessariamente alcançar os objetivos dos demais (KOTONYA; SOMMERVILLE, 1998).

Problemas e conflitos encontrados nos requisitos devem ser listados. Usuários, clientes, especialistas de domínio e engenheiros de requisitos devem discutir os requisitos que apresentam problemas, negociar e chegar a um acordo sobre as modificações a serem feitas. Idealmente, as discussões devem ser governadas pelas necessidades da organização, incluindo o orçamento e o cronograma disponíveis. No entanto, muitas vezes, as negociações são influenciadas por considerações políticas e os requisitos são definidos em função da posição e da personalidade dos indivíduos e não em função de argumentos e razões (KOTONYA; SOMMERVILLE, 1998).

A maior parte do tempo da negociação é utilizada para resolver conflitos de requisitos. Quando discussões informais entre analistas, especialistas de domínio e usuários não forem suficientes para resolver os problemas, é necessária a realização de reuniões de negociação, que envolvem (KOTONYA; SOMMERVILLE, 1998):

- Discussão: os requisitos que apresentam problemas são discutidos e os interessados presentes opinam sobre eles.
- Priorização: requisitos são priorizados para identificar requisitos críticos e ajudar nas decisões e planejamento.
- Concordância: soluções para os problemas são identificadas, mudanças são feitas e um acordo sobre o conjunto de requisitos é acertado.

2.2.3 – Documentação de Requisitos

Os requisitos e modelos capturados nas etapas anteriores devem ser descritos e apresentados em documentos. A documentação é, portanto, uma atividade de registro e oficialização dos resultados da Engenharia de Requisitos. Como resultado, um ou mais documentos devem ser produzidos.

Uma boa documentação fornece muitos benefícios, tais como (IEEE, 1998; NUSEIBEH; EASTERBROOK, 2000): (i) facilita a comunicação dos requisitos; (ii) reduz o esforço de desenvolvimento, pois sua preparação força usuários e clientes a considerar os requisitos atentamente, evitando retrabalho nas fases posteriores; (iii) fornece uma base realística para estimativas; (iv) fornece uma base para verificação e validação; (v) serve como base para futuras manutenções ou incremento de novas funcionalidades.

A documentação dos requisitos tem um conjunto diversificado de interessados, dentre eles (SOMMERVILLE, 2007; KOTONYA; SOMMERVILLE, 1998):

- Clientes, Usuários e Especialistas de Domínio: são interessados na documentação de requisitos, uma vez que atuam na especificação, avaliação e alteração de requisitos.
- Gerentes de Cliente: utilizam o documento de requisitos para planejar um pedido de proposta para o desenvolvimento de um sistema, contratar um fornecedor e para acompanhar o desenvolvimento do sistema.
- Gerentes de Fornecedor: utilizam a documentação dos requisitos para planejar uma proposta para o sistema e para planejar e acompanhar o processo de desenvolvimento.
- Desenvolvedores (analistas, projetistas, programadores e mantenedores): utilizam a documentação dos requisitos para compreender o sistema e as relações entre suas partes.
- Testadores: utilizam a documentação dos requisitos para projetar casos de teste, sobretudo testes de validação do sistema.

Diferentes interessados têm propósitos diferentes. Assim, pode ser útil ter mais do que um documento para registrar os resultados da engenharia de requisitos. Conforme discutido anteriormente, Pfleeger (2004) sugere que dois tipos de documentos de requisitos sejam elaborados: um Documento de Definição de Requisitos e um Documento de Especificação de Requisitos.

O Documento de Definição de Requisitos deve conter uma descrição do propósito do sistema, uma breve descrição do domínio do problema tratado pelo sistema e listas de requisitos funcionais e não funcionais, descritos em linguagem natural (requisitos de cliente). Para facilitar a identificação e rastreamento dos requisitos, devem-se utilizar identificadores únicos para cada um dos requisitos listados. O público-alvo desse documento são clientes, usuários, gerentes (de cliente e de fornecedor) e desenvolvedores.

O Documento de Especificação de Requisitos deve conter os requisitos escritos a partir da perspectiva do desenvolvedor, devendo haver uma correspondência direta com os requisitos no Documento de Definição de Requisitos, de modo a se ter requisitos rastreáveis. Os vários modelos produzidos na fase de análise devem ser apresentados no

Documento de Especificação de Requisitos, bem como glossários de termos usados e outras informações julgadas relevantes.

Deve-se observar que não há um padrão definido quanto à quantidade e ao nome dos documentos de requisitos. Há organizações que optam por ter apenas um documento de requisitos, contendo diversas seções, algumas tratando de requisitos de cliente, outras tratando de requisitos de sistema. Outras organizações, por sua vez, fazem uso de vários documentos distintos, capturando em documentos separados, por exemplo, requisitos funcionais, requisitos não funcionais, modelos de caso de uso e modelos estruturais e comportamentais. De fato, cabe a cada organização definir a quantidade, o nome e o conteúdo de cada documento. Para estruturar o conteúdo, é necessário que a organização defina seus modelos de documentos de requisitos.

Várias diretrizes têm sido propostas com objetivo de melhorar a estrutura e organização da documentação de requisitos, dentre elas (SOMMERVILLE; SAWYER, 1997; KOTONYA; SOMMERVILLE, 1998; PRESSMAN, 2006; WIEGERS, 2003): (i) definir um modelo de documento (*template*) para cada tipo de documento a ser considerado, definindo um padrão de estrutura para o documento; (ii) explicar como cada classe de leitores deve usar os diferentes tipos de documentos; (iii) definir termos especializados em um glossário; (iv) organizar o layout do documento para facilitar a leitura; (v) auxiliar os leitores a encontrar a informação, incluindo recursos tais como listas de conteúdo e índices, e organizando os requisitos em capítulos, seções e subseções identificadas; (vi) identificar as fontes dos requisitos de modo a manter dados da origem do requisito, de modo que, quando alguma mudança for solicitada, seja possível saber com quem essa mudança deve ser discutida e avaliada; e (vii) criar um identificador único para cada requisito, de modo a facilitar a rastreabilidade e o controle de mudanças.

2.2.4 – Verificação e Validação de Requisitos

As atividades de Verificação & Validação (V&V) devem ser iniciadas o quanto antes no processo de desenvolvimento de software, pois quanto mais tarde os defeitos são encontrados, maiores os custos associados à sua correção (ROCHA; MALDONADO; WEBER, 2001). Uma vez que os requisitos são a base para o desenvolvimento, é fundamental que eles sejam cuidadosamente avaliados. Assim, os documentos produzidos durante a atividade de documentação de requisitos devem ser submetidos à verificação e à validação.

É importante realçar a diferença entre verificação e validação. O objetivo da verificação é assegurar que o software esteja sendo construído de forma correta. Deve-se verificar se os artefatos produzidos atendem aos requisitos estabelecidos e se os padrões organizacionais (de produto e processo) foram consistentemente aplicados. Por outro lado, o objetivo da validação é assegurar que o software que está sendo desenvolvido é o software correto, ou seja, assegurar que os requisitos, e o software deles derivado, atendem ao uso proposto (ROCHA; MALDONADO; WEBER, 2001).

No caso de requisitos, a verificação é feita, sobretudo, em relação à consistência entre requisitos e modelos e à conformidade com padrões organizacionais de documentação de requisitos. Já a validação tem de envolver a participação de usuários e clientes, pois somente eles são capazes de dizer se os requisitos atendem aos propósitos do sistema.

Nas atividades de V&V de requisitos, examinam-se os documentos de requisitos para assegurar que (PRESSMAN, 2006; KOTONYA; SOMMERVILLE, 1998; WIEGERS, 2003): (i) todos os requisitos do sistema tenham sido declarados de modo não-ambíguo, (ii) as inconsistências, conflitos, omissões e erros tenham sido detectados e corrigidos, (iii) os documentos estão em conformidade com os padrões estabelecidos e (iv) os requisitos realmente satisfazem às necessidades dos clientes e usuários. Em outras palavras, idealmente, um requisito, seja ele funcional ou não funcional, deve ser (WIEGERS, 2003; PFLEEGER, 2004):

- Completo: o requisito deve descrever completamente a funcionalidade a ser entregue (no caso de requisito funcional) ou a restrição a ser considerada (no caso de requisito não funcional). Ele deve conter as informações necessárias para que o desenvolvedor possa projetar, implementar e testar essa funcionalidade ou restrição.
- Correto: cada requisito deve descrever exatamente a funcionalidade ou restrição a ser incorporada ao sistema.
- Consistente: o requisito não deve ser ambíguo ou conflitar com outro requisito.
- Realista: deve ser possível implementar o requisito com a capacidade e com as limitações do sistema e do ambiente de desenvolvimento.
- Necessário: o requisito deve descrever algo que o cliente realmente precisa ou que é requerido por algum fator externo ou padrão da organização.
- Passível de ser priorizado: os requisitos devem ter ordem de prioridade para facilitar o gerenciamento durante o desenvolvimento do sistema.
- Verificável e passível de confirmação: deve ser possível desenvolver testes para verificar se o requisito foi realmente implementado.
- Rastreável: deve ser possível identificar quais requisitos foram tratados em um determinado artefato, bem como identificar que produtos foram originados a partir de um requisito.

Neste contexto é útil (WIEGERS, 2003):

- Realizar revisões dos documentos de requisitos, procurando por problemas (conflitos, omissões, inconsistências, desvios dos padrões etc) e discutindo soluções.
- Definir casos de teste para os requisitos especificados.
- Definir critérios de aceitação de requisitos, i.e., os usuários devem descrever como vão determinar se o produto atende às suas necessidades e se é adequado para uso.

De maneira geral, i.e., sem estarem restritas a requisitos, as atividades de V&V envolvem análises estáticas e dinâmicas. A análise dinâmica (ou testes) objetiva detectar defeitos ou erros no software por meio da execução do produto. Já a análise estática não envolve a execução do produto, sendo feita por meio de revisões dos artefatos a serem avaliados (ROCHA; MALDONADO; WEBER, 2001). No caso de requisitos, podem-se realizar revisões dos documentos de requisitos para avaliar requisitos e modelos (análise

estática), bem como é possível utilizar prototipagem para validar requisitos (análise dinâmica).

A análise dinâmica, neste caso, se justifica, pois, muitas vezes, as pessoas encontram dificuldades em visualizar como os requisitos serão traduzidos em um sistema. Essa dificuldade pode ser amenizada por meio de protótipos, que auxiliam os usuários na identificação de problemas e na sugestão de melhorias dos requisitos. Dessa forma, a prototipagem pode ser utilizada no processo de validação de requisitos. Entretanto, sua utilização nessa fase tem uma relação custo-benefício mais efetiva quando ela tiver sido empregada também na fase de levantamento de requisitos (KOTONYA; SOMMERVILLE, 1998).

Das atividades de verificação e validação, a atividade de teste é considerada um elemento crítico para a garantia da qualidade dos artefatos produzidos ao longo do desenvolvimento e, por conseguinte, do produto de software final (ROCHA; MALDONADO; WEBER, 2001). Contudo, exceto por meio de protótipos ou especificações de requisitos executáveis (estas um recurso muito pouco utilizado na prática), não é possível testar requisitos. Entretanto, uma das características de qualidade de um requisito bem elaborado é ser testável e, portanto, uma boa maneira de identificar problemas nos requisitos é definir casos de teste para os mesmos. Se um requisito está incompleto, inconsistente ou ambíguo, pode ser difícil definir casos de teste para ele (KOTONYA; SOMMERVILLE, 1998).

Definir casos de teste para requisitos e avaliar protótipos são importantes meios de se verificar e validar requisitos. Contudo, é imprescindível, ainda, realizar revisões dos documentos de requisitos. De maneira geral, em uma revisão, processos, documentos e outros artefatos são revisados por um grupo de pessoas, com o objetivo de avaliar se os mesmos estão em conformidade com os padrões organizacionais estabelecidos e se o propósito de cada um deles está sendo atingido. Assim, o objetivo de uma revisão é detectar erros e inconsistências em artefatos e processos, sejam eles relacionados à forma, sejam eles relacionados ao conteúdo, e apontá-los aos responsáveis pela sua elaboração (ROCHA; MALDONADO; WEBER, 2001).

Em um formato de revisão técnica formal, o processo de revisão começa com o planejamento da revisão, quando uma equipe de revisão é formada, tendo à frente um líder. A equipe de revisão deve incluir membros da equipe que possam ser efetivamente úteis para atingir o objetivo da revisão. Muitas vezes, a pessoa responsável pela elaboração do artefato a ser revisado integra a equipe de revisão (ROCHA; MALDONADO; WEBER, 2001).

O propósito da revisão deve ser previamente informado e o material a ser revisado deve ser entregue com antecedência para que cada membro da equipe de revisão possa avaliá-lo. Uma vez que todos estejam preparados, uma reunião é convocada pelo líder. Dando início à reunião de revisão, normalmente, o autor do artefato apresenta o mesmo e descreve a perspectiva utilizada para a sua construção. O líder orientará o processo de revisão, passando por todos os aspectos relevantes a serem revistos. Todas as considerações dos vários membros da equipe de revisão devem ser discutidas e as decisões registradas, dando origem a uma ata de reunião de revisão, contendo uma lista de defeitos encontrados. Essa reunião deve ser relativamente breve (duas horas, no máximo), uma vez que todos já devem estar preparados para a mesma (ROCHA; MALDONADO; WEBER, 2001).

No que se refere à revisão de requisitos, diversas técnicas de leitura podem ser usadas. A mais simples é a leitura *ad-hoc*, na qual os revisores aplicam seus próprios conhecimentos na revisão dos documentos de requisitos. Diferentemente de uma abordagem *ad-hoc*, outras técnicas buscam aumentar a eficiência dos revisores, direcionando os esforços para as melhores práticas de detecção de defeitos. Técnicas de leitura baseada em listas de verificação (*checklists*), leitura baseada em perspectivas e leitura de modelos orientados a objetos são bastante usadas na verificação e validação de documentos de requisitos.

Checklists definem uma lista de aspectos que devem ser verificados pelos revisores, guiando-os no trabalho de revisão. Podem ser usados em conjunto com outras técnicas, tais como as técnicas de leitura baseada em perspectiva e leitura de modelos orientados a objetos. A Figura 2.3 mostra um exemplo de *checklist* de requisitos.

Checklist para Avaliação de Documentos de Requisitos			
Projeto:	<<nome do projeto>>		
Documento:	<<nome do documento>>	**Versão:**	<<número>>
Item a ser avaliado		**Resultado**	**Problemas Detectados**
Aderência à Estrutura do Modelo de Documento de Requisitos			
1. O documento segue os padrões de fonte e espaçamento?			
2. Todas as seções obrigatórias estão no documento?			
3. As seções do documento estão consistentes com as orientações do *template* para a elaboração das mesmas?			
Conteúdo do Documento			
1. A descrição do propósito está bem colocada, i.e., é sucinta (um único parágrafo) e descreve o objetivo do sistema?			
2. A descrição do minimundo permite uma compreensão básica do domínio, do negócio e do problema sendo tratado pelo sistema?			
3. Os requisitos listados estão compatíveis com a descrição do minimundo?			
4. Há omissões, inconsistência, informação estranha ou ambiguidade na descrição do minimundo e nos requisitos?			
5. Os requisitos estão sendo descritos em formato apropriado para compreensão por clientes e usuários?			
6. Identificadores de requisitos são únicos?			
7. As descrições dos requisitos seguem os padrões definidos?			
8. As dependências entre requisitos estão adequadamente registradas?			
9. As prioridades dos requisitos estão consistentes?			

Figura 2.3 – Exemplo de Checklist de Requisitos

A técnica de leitura baseada em perspectiva foi desenvolvida especificamente para a verificação e validação de requisitos. Ela explora a observação de quais informações de requisitos são mais ou menos importantes para as diferentes formas de utilização do documento de requisitos. Cada revisor realiza a revisão segundo uma perspectiva diferente. Algumas perspectivas tipicamente consideradas são as perspectivas de clientes, desenvolvedores e testadores. *Checklists* para cada uma dessas perspectivas podem ser providos (ROCHA; MALDONADO; WEBER, 2001).

A leitura de modelos orientados a objetos propõe um conjunto de técnicas de leitura para revisão dos diferentes diagramas utilizados durante um projeto orientado a objetos. Cada uma das técnicas é definida para um conjunto de diagramas a serem

analisados em conjunto. O processo de leitura é realizado de duas maneiras: a leitura horizontal diz respeito à consistência entre artefatos elaborados em uma mesma fase, procurando verificar se esses artefatos estão descrevendo consistentemente diferentes aspectos de um mesmo sistema, no nível de abstração relacionado à fase em questão; a leitura vertical refere-se à consistência entre artefatos elaborados em diferentes fases (análise e projeto) (ROCHA; MALDONADO; WEBER, 2001).

2.2.5 – Gerência de Requisitos

Mudanças nos requisitos ocorrem ao longo de todo o processo de software, desde o levantamento e análise de requisitos até durante a operação do sistema. Elas são decorrentes de diversos fatores, tais como descoberta de erros, omissões, conflitos e inconsistências nos requisitos, melhor entendimento por parte dos usuários de suas necessidades, problemas técnicos, de cronograma ou de custo, mudança nas prioridades do cliente, mudanças no negócio, aparecimento de novos competidores, mudanças econômicas, mudanças na equipe, mudanças no ambiente onde o software será instalado e mudanças organizacionais ou legais. Para minimizar as dificuldades impostas por essas mudanças, é necessário gerenciar requisitos.

O processo de gerência de requisitos envolve as atividades que ajudam a equipe de desenvolvimento a identificar, controlar e rastrear requisitos e gerenciar mudanças de requisitos em qualquer momento ao longo do ciclo de vida do software (KOTONYA; SOMMERVILLE, 1998; PRESSMAN, 2006). Os principais objetivos desse processo são (KOTONYA; SOMMERVILLE, 1998):

- Gerenciar alterações nos requisitos acordados.
- Gerenciar relacionamentos entre requisitos.
- Gerenciar dependências entre requisitos e outros documentos produzidos durante o processo de software.

Para tal, o processo de gerência de requisitos deve incluir as seguintes atividades, ilustradas na Figura 2.4 (WIEGERS, 2003): controle de mudanças, controle de versão, acompanhamento do estado dos requisitos e rastreamento de requisitos.

Figura 2.4 - Atividades da Gerência de Requisitos (WIEGERS, 2003)

O controle de mudança define os procedimentos e padrões que devem ser usados para gerenciar alterações em requisitos, assegurando que qualquer proposta de mudança seja analisada conforme os critérios estabelecidos pela organização (KOTONYA; SOMMERVILLE, 1998). Mudanças podem ser necessárias em diferentes momentos e por diferentes razões. De maneira geral, o controle de mudanças envolve atividades para (KOTONYA; SOMMERVILLE, 1998; WIEGERS, 2003):

- Verificar se uma mudança é válida.
- Descobrir quais os requisitos e artefatos afetados pela mudança, o que envolve rastrear informações.
- Estimar o impacto e o custo das mudanças.
- Negociar as mudanças com os clientes.
- Alterar requisitos e documentos associados.

Para garantir uma abordagem consistente, recomenda-se que as organizações definam um conjunto de políticas de gerência de mudança, contemplando (KOTONYA; SOMMERVILLE, 1998):

- o processo de solicitação de mudança e as informações necessárias para processar cada solicitação;
- o processo de análise de impacto e de custos da mudança, além das informações de rastreabilidade associadas;
- os membros que formalmente avaliarão as mudanças;
- as ferramentas que auxiliarão o processo.

Se as mudanças não forem controladas, alterações com baixa prioridade podem ser implementadas antes de outras mais importantes e modificações com custo alto que não são realmente necessárias podem ser aprovadas.

Ao se detectar a necessidade de alteração em um ou mais requisitos, deve-se registrar uma solicitação de mudança, a qual deve ser avaliada por algum membro da equipe do projeto de software. Nessa avaliação, o impacto da alteração deve ser determinado e valores de custo, esforço, tempo e viabilidade devem ser repassados ao solicitante da mudança. Assim, uma parte crítica do controle de mudanças é a avaliação do impacto de uma mudança no restante do software. Para que seja possível efetuar essa avaliação, cada requisito deve estar identificado unicamente e deve ser possível, por exemplo, saber quais são os requisitos dependentes desse requisito e em quais artefatos do processo de software esse requisito é tratado. Portanto, é necessário estabelecer uma rede de ligações de modo que um requisito e os elementos ligados a ele possam ser rastreados. Surge, então, o conceito de rastreabilidade.

A rastreabilidade pode ser definida como a habilidade de se acompanhar a vida de um requisito em ambas as direções do processo de software e durante todo o ciclo de vida. Ela fornece uma base para o desenvolvimento de uma trilha de auditoria para todo o projeto, possibilitando encontrar outros requisitos e artefatos que podem ser afetados pelas mudanças solicitadas (PALMER, 1997). Para tal, é necessário haver ligações entre requisitos e entre requisitos e outros elementos do processo de software. Assim, a identificação de dependências entre requisitos, de requisitos conflitantes, da origem dos requisitos e de seus interessados, além da identificação de em quais artefatos produzidos durante o desenvolvimento de software um requisito é tratado, é de fundamental

importância para que a rastreabilidade possa ser implementada (WIEGERS, 2003; ROBERTSON; ROBERTSON, 2006; KOTONYA; SOMMERVILLE, 1998).

Para apoiar a gerência de requisitos, matrizes de rastreabilidade podem ser produzidas. Elas relacionam os requisitos identificados a um ou mais aspectos do sistema ou do seu ambiente, de modo que elas possam ser procuradas rapidamente para entender como uma modificação em um requisito vai afetar diferentes aspectos do sistema. Dentre as muitas possíveis matrizes de rastreabilidade, há as seguintes:

- Matriz de rastreabilidade de dependência entre requisitos: indica como os requisitos estão relacionados uns com os outros.
- Matriz de rastreabilidade requisitos ←→ fontes: indica a fonte de cada requisito.
- Matriz de rastreabilidade requisitos ←→ subsistemas: indica os subsistemas que tratam os requisitos.
- Matriz de rastreabilidade requisitos ←→ casos de uso: indica os casos de uso que detalham um requisito funcional ou tratam um requisito não funcional.

Além das matrizes de rastreabilidade de requisitos, outras matrizes de rastreabilidade entre outros artefatos do processo podem ser construídas, de modo a apoiar a gerência de requisitos. Por exemplo, ao se estabelecer uma matriz de rastreabilidade casos de uso ←→ classes, indicando que classes de um modelo de análise são necessárias para se tratar um caso de uso, é possível, em conjunto com uma matriz de rastreabilidade requisitos ←→ casos de uso, saber que classes são importantes no tratamento de um requisito.

Davis (apud KOTONYA; SOMMERVILLE, 1998) aponta quatro tipos de rastreabilidade interessantes para que uma análise de impacto de mudanças possa ser feita mais facilmente, como ilustra a Figura 2.5:

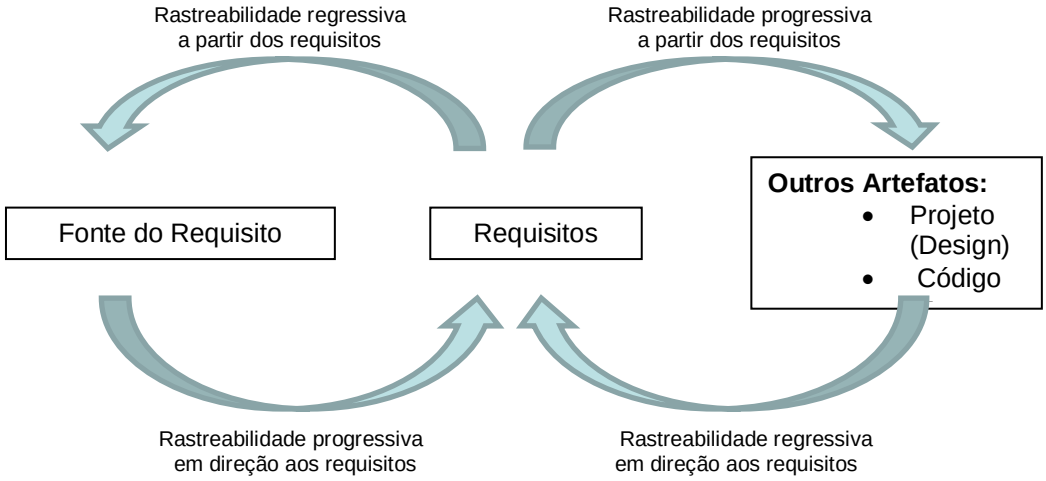


Figura 2.5 – Tipos de Rastreabilidade

- Regressiva a partir dos requisitos (*backward-from traceability*): relaciona requisitos com suas origens/fontes (outros documentos ou pessoas);
- Progressiva a partir dos requisitos (*forward-from traceability*): relaciona requisitos a artefatos do projeto (modelos de análise e projeto, código etc.);

- Regressiva em direção aos requisitos (*backward-to traceability*): relaciona artefatos do projeto aos requisitos;
- Progressiva em direção aos requisitos (*forward-to traceability*): relaciona as fontes aos requisitos relevantes.

Embora a rastreabilidade de requisitos não possa ser completamente automatizada, porque o conhecimento das ligações se origina na mente dos membros da equipe de desenvolvimento, uma vez identificadas essas ligações, ferramentas de apoio são importantíssimas para ajudar a gerenciar a grande quantidade de informações de rastreabilidade (WIEGERS, 2003).

Requisitos guiam várias atividades do processo de software, dentre elas planejamento, projeto (*design*), codificação e testes, e, portanto, esses artefatos devem ser rastreáveis para e a partir dos requisitos. A seguir são citadas algumas situações, em diversas etapas, nas quais os requisitos são importantes (WIEGERS, 2003):

- Planejamento de Projeto: requisitos são úteis para dimensionar o projeto, sendo utilizados para estimar o tamanho do produto. Planos devem ser atualizados caso haja mudanças em requisitos e requisitos prioritários são usados para direcionar iterações, quando modelos de ciclo de vida iterativos são adotados.
- Projeto e Codificação: requisitos, com destaque para os não funcionais, são usados para direcionar o projeto da arquitetura do sistema e são alocados a componentes. Mudanças em requisitos devem ser rastreadas para o projeto e o código gerado, de modo a guiar as alterações.
- Testes: requisitos são a base para diversos tipos de testes, dentre eles testes de sistema e de aceitação.

2.3 – Engenharia de Requisitos e Normas e Modelos de Qualidade

Visando à qualidade no processo de software, modelos e normas de qualidade de processo de software têm sido propostos, dentre eles o CMMI (*Capability Maturity Model Integration*) (SEI, 2010) e o Modelo de Referência MPS para Software (MR-MPS-SW) (SOFTEX, 2016).

O CMMI é um modelo de qualidade de processo desenvolvido pelo Instituto de Engenharia de Software (*Software Engineering Institute* – SEI) da Universidade de Carnegie Mellon. O CMMI para Desenvolvimento (CMMI-Dev) (SEI, 2010) enfoca o processo de software e tem como objetivo fornecer diretrizes para a definição e melhoria de processos de software de uma organização. O CMMI, em sua representação em estágio, possui cinco níveis de maturidade: 1- Inicial, 2- Gerenciado, 3- Definido, 4- Gerenciado Quantitativamente e 5- Em Otimização.

O CMMI-Dev trata de requisitos tanto no nível 2, através da área de processo Gestão de Requisitos, quanto no nível 3, por meio da área de processo Desenvolvimento de Requisitos. Na Gestão de Requisitos o objetivo é gerenciar os requisitos dos produtos e componentes de produto do projeto e garantir alinhamento entre esses requisitos e os planos e produtos de trabalho do projeto (SEI, 2010). Para isso, o CMMI sugere as seguintes práticas:

SG 1 Gerenciar Requisitos
SP 1.1 Entender os requisitos.
SP 1.2 Obter comprometimento com os requisitos.
SP 1.3 Gerenciar mudanças de requisitos.
SP 1.4 Manter rastreabilidade bidirecional dos requisitos.
SP 1.5 Garantir alinhamento entre o trabalho de projeto (planos de projeto e produtos de trabalho) e requisitos.

O Desenvolvimento de Requisitos, por sua vez, visa levantar, analisar e estabelecer requisitos de cliente, do produto e de componentes do produto (SEI, 2010). Neste caso, o CMMI sugere as seguintes práticas:

SG 1 Desenvolver Requisitos de Cliente
SP 1.1 Levantar necessidades.
SP 1.2 Transformar necessidades dos interessados em requisitos de cliente.

SG 2 Desenvolver Requisitos do Produto
SP 2.1 Estabelecer os requisitos do produto e de componentes do produto.
SP 2.2 Alocar requisitos de componentes do produto.
SP 2.3 Identificar requisitos de interface.

SG 3 Analisar e Validar Requisitos
SP 3.1 Estabelecer conceitos e cenários operacionais.
SP 3.2 Estabelecer uma definição da funcionalidade e dos atributos de qualidade requeridos.
SP 3.3 Analisar requisitos.
SP 3.4 Analisar requisitos para balancear.
SP 3.5 Validar requisitos.

O Programa MPS.BR (Melhoria de Processo do Software Brasileiro) (SOFTEX, 2016) tem como objetivo a melhoria de processo do software brasileiro. Uma das metas do programa MPS.BR é definir e aprimorar um modelo de melhoria e avaliação de processo de software, visando preferencialmente às micro, pequenas e médias empresas, de forma a atender as suas necessidades de negócio e ser reconhecido nacional e internacionalmente como um modelo aplicável à indústria de software. O Modelo de Referência MPS para Software (MR-MPS-SW) é desenvolvido no contexto do MPS.BR e sua base técnica é composta pelas normas ISO/IEC 12207 e ISO/IEC 15504-2, além buscar garantir conformidade com o CMMI. Essa abordagem visa garantir que o MR-MPS-SW está em conformidade com padrões internacionais.

Assim como o CMMI, o MR-MPS-SW é organizado em níveis de maturidade. No caso do MR-MPS-SW, contudo, são sete níveis, a saber: G- Parcialmente Gerenciado, F- Gerenciado, E- Parcialmente Definido, D- Largamente Definido, C- Definido, B- Gerenciado Quantitativamente, A- Em Otimização. O MR-MPS-SW procura manter uma correspondência entre seus níveis de maturidade e os níveis de maturidade do CMMI. Assim, o nível 2 do CMMI corresponde ao nível F do MPS, o nível 3 do CMMI ao nível C do MPS, o nível 4 do CMMI ao nível B do MPS e finalmente o nível 5 do CMMI corresponde ao nível A do MPS.

O MR-MPS-SW define, em seu nível G, o processo de Gerência de Requisitos e no nível D o processo de Desenvolvimento de Requisitos. O propósito do processo de

Gerência de Requisitos é "*gerenciar os requisitos do produto e dos componentes do produto do projeto e identificar inconsistências entre os requisitos, os planos do projeto e os produtos de trabalho do projeto*". Esse processo tem como resultados esperados:

> GRE 1. O entendimento dos requisitos é obtido junto aos fornecedores de requisitos;
>
> GRE 2. Os requisitos são avaliados com base em critérios objetivos e um comprometimento da equipe técnica com esses requisitos é obtido;
>
> GRE 3. A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho é estabelecida e mantida;
>
> GRE 4. Revisões em planos e produtos de trabalho do projeto são realizadas visando identificar e corrigir inconsistências em relação aos requisitos;
>
> GRE 5. Mudanças nos requisitos são gerenciadas ao longo do projeto.

O Desenvolvimento de Requisitos, por sua vez, tem por objetivo "*definir os requisitos do cliente, do produto e dos componentes do produto*". Esse processo tem como resultados esperados:

> DRE 1. As necessidades, expectativas e restrições do cliente, tanto do produto quanto de suas interfaces, são identificadas;
>
> DRE 2. Um conjunto definido de requisitos do cliente é especificado e priorizado a partir das necessidades, expectativas e restrições identificadas;
>
> DRE 3. Um conjunto de requisitos funcionais e não-funcionais, do produto e dos componentes do produto que descrevem a solução do problema a ser resolvido, é definido e mantido a partir dos requisitos do cliente;
>
> DRE 4. Os requisitos funcionais e não-funcionais de cada componente do produto são refinados, elaborados e alocados. Interfaces internas e externas do produto e de cada componente do produto são definidas;
>
> DRE 5. Conceitos operacionais e cenários são desenvolvidos;
>
> DRE 6. Os requisitos são analisados, usando critérios definidos, para balancear as necessidades dos interessados com as restrições existentes;
>
> DRE 7. Os requisitos são validados.

De uma maneira geral, pode-se constatar que a Engenharia de Requisitos, pela sua importância no contexto do desenvolvimento de software, é cuidadosamente tratada pelos principais modelos de qualidade de processo de software.

Referências do Capítulo

AURUM, A., WOHLIN, C., *Engineering and Managing Software Requirements*, Springer-Verlag, 2005.

BLAHA, M., RUMBAUGH, J., *Modelagem e Projetos Baseados em Objetos com UML 2*, Elsevier, 2006.

IEEE, IEEE Recommended Practice for Software Requirements Specifications: IEEE Std 830-1998. New York: IEEE, 1998.

KENDALL, K.E., KENDALL, J.E.; *Systems Analysis and Design*, Prentice Hall, 8th Edition, 2010.

van LAMSWEERDE, A., *Requirements Engineering: From System Goals to UML Models to Software Specifications*, Willey, 2009.

KOTONYA, G., SOMMERVILLE, I., *Requirements engineering: processes and techniques*. Chichester, England: John Wiley, 1998.

NUSEIBEH, B., EASTERBROOK, S., "Requirements engineering: a roadmap". In: Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland, 2000.

PALMER, J.D., "Traceability". In: THAYER, H. R.; DORFMAN, M. (Org.) *Software requeriments engineering*. 2nd edition, Los Alamitos, California: IEEE Computer Society, p.364-374, 1997.

PFLEEGER, S.L., *Engenharia de Software: Teoria e Prática*, São Paulo: Prentice Hall, 2ª edição, 2004.

PRESSMAN, R.S., *Engenharia de Software*, McGraw-Hill, 6ª edição, 2006.

ROBERTSON, S., ROBERTSON, J. *Mastering the Requirements Process***.** 2nd Edition. Addison Wesley, 2006.

ROCHA, A.R.C., MALDONADO, J.C., WEBER, K.C., *Qualidade de Software: Teoria e Prática*. São Paulo: Prentice Hall, 2001.

SEI, CMMI for Development, Version, 1.3, CMMI-Dev V1.3, CMU/SEI-2010-TR-033, 2010.

SOFTEX, MPS.BR – Melhoria de Processo do Software Brasileiro – Guia Geral MPS de Software: 2016, Janeiro 2016.

SOMMERVILLE, I., *Engenharia de Software*, 8ª Edição. São Paulo: Pearson – Addison Wesley, 2007.

SOMMERVILLE, I., SAWYER, P., *Requirements engineering: a good practice guide*. Chichester, England: John Wiley, 1997.

WIEGERS, K.E., *Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle*. 2nd Edition, Microsoft Press, Redmond, Washington, 2003.

Capítulo 3 – Levantamento de Requisitos

Analistas e desenvolvedores trabalham com clientes e usuários para saber mais sobre o problema a ser resolvido, os serviços a serem providos pelo sistema, restrições etc. Isso não significa apenas perguntar o que eles desejam do sistema. Ao contrário, requer uma análise criteriosa da organização, do domínio do problema e dos processos de negócio que serão apoiados pelo sistema. O quadro fica ainda mais complicado por causa de diversos fatores, dentre eles: raramente os clientes têm uma visão clara de seus requisitos; diferentes pessoas em uma organização têm diferentes requisitos, às vezes conflitantes; e há limitações financeiras, tecnológicas e de prazos. Assim, levantar requisitos não é uma tarefa simples (KOTONYA; SOMMERVILLE, 1998).

Este capítulo enfoca o levantamento de requisitos. A Seção 3.1 procura dar uma visão geral dessa atividade, discutindo problemas típicos e tarefas a serem realizadas no levantamento de requisitos. A Seção 3.2 apresenta algumas técnicas para apoiar o levantamento de requisitos. A Seção 3.3 discute como a modelagem de processos de negócio pode ajudar na captura de requisitos de software. Finalmente, a Seção 3.4 discute formas de se escrever e documentar requisitos.

3.1 – Visão Geral do Levantamento de Requisitos

O levantamento de requisitos preocupa-se com o aprendizado e entendimento das necessidades dos usuários e patrocinadores do projeto, com o objetivo final de comunicar essas necessidades para os desenvolvedores do sistema. Uma parte substancial do levantamento de requisitos é dedicada a descobrir, extrair e aparar arestas dos desejos de potenciais interessados (AURUM; WOHLIN, 2005).

A fase de levantamento de requisitos envolve buscar, junto aos usuários, clientes e outros interessados, seus sistemas e documentos, todas as informações possíveis sobre as funções que o sistema deve executar (requisitos funcionais) e as restrições sob as quais ele deve operar (requisitos não funcionais). O produto principal dessa fase é o Documento de Definição de Requisitos.

Entretanto, conforme citado anteriormente, levantar requisitos não é uma tarefa fácil. Esta é uma tarefa de natureza multidimensional e os analistas (ou engenheiros de requisitos) se veem diante de vários desafios, dentre eles (KOTONYA; SOMMERVILLE, 1998):

- O conhecimento acerca do domínio de aplicação encontra-se disperso em uma variedade de fontes, tais como livros, manuais e, sobretudo, nas cabeças das pessoas que trabalham na área. Além disso, muitas vezes, envolve uma terminologia especializada que não é imediatamente compreensível pelo analista.
- As pessoas que entendem o problema a ser resolvido frequentemente estão muito ocupadas para despender tempo ajudando os analistas a entender os requisitos para

um novo sistema. Elas podem, inclusive, não estar convencidas da necessidade do novo sistema e, por conseguinte, não quererem se envolver no processo de ER.

- Fatores políticos e questões organizacionais podem influenciar os requisitos para o sistema, bem como o estabelecimento de suas prioridades.
- Clientes e usuários frequentemente não sabem o que realmente querem do sistema, exceto em termos muito gerais. Mesmo quando eles sabem o que querem, eles têm dificuldade para articular os requisitos. Além disso, podem ter demandas não realistas por não estarem cientes dos custos de suas solicitações.
- O ambiente de negócio no qual ocorre o levantamento de requisitos está em constante mudança. Os requisitos podem mudar, a sua importância pode mudar e novos requisitos podem surgir de novos interessados.

Dadas todas essas dificuldades, o levantamento de requisitos deve ser conduzido de forma bastante cuidadosa, fazendo uso de técnicas para capturar e especificar os requisitos levantados.

Uma boa prática consiste em fazer o levantamento de requisitos de forma incremental. Inicialmente, em um levantamento preliminar de requisitos, apenas requisitos de cliente são capturados. Depois, em várias iterações, outros requisitos de cliente são capturados e requisitos de sistema vão sendo detalhados e especificados. Neste contexto, é importante realçar que o levantamento e a análise de requisitos são atividades estreitamente relacionadas e, portanto, devem ocorrer em paralelo. Assim, à medida que os requisitos vão sendo detalhados, eles devem ser modelados e especificados.

O levantamento preliminar de requisitos tem por objetivo prover uma visão do todo para se definir o que é mais importante e depois dividir o todo em partes para especificar os detalhes. Nessa fase, o levantamento é rápido e genérico, sendo feito em extensão e não em profundidade, i.e., o analista deve entender a extensão do que o sistema deve fazer, mas sem entrar em detalhes. Somente nos ciclos iterativos os requisitos serão detalhados, especificados e modelados (WAZLAWICK, 2004).

O levantamento preliminar de requisitos inicia-se com uma declaração de alto nível, informal e incompleta, da missão do projeto. Essa declaração pode ser apresentada na forma de um conjunto de metas, funções e restrições fundamentais para o sistema ou como uma explicação sobre os problemas a serem resolvidos. Esses resultados preliminares formam a base para investigação adicional e para o refinamento dos requisitos, de maneira tipicamente iterativa e incremental. Assim, o levantamento de requisitos pode ser visto como um processo realizado de forma incremental, ao longo de múltiplas sessões, iterativamente em direção a níveis de detalhe cada vez maiores e pelo menos parcialmente em paralelo com outras atividades do processo de software (AURUM; WOHLIN, 2005).

O levantamento de requisitos envolve um conjunto de atividades que deve permitir a comunicação, priorização, negociação e colaboração com todos os interessados relevantes. Deve prover, ainda, uma base para o aparecimento, descoberta e invenção de requisitos, como parte de um processo altamente interativo (AURUM; WOHLIN, 2005).

Zowghi e Coulin apud (AURUM; WOHLIN, 2005) indicam que as atividades do processo de levantamento de requisitos podem ser agrupadas em cinco tipos fundamentais de atividades, a saber:

- Entendimento do Domínio de Aplicação: é importante investigar e examinar em detalhes a porção do mundo real onde o sistema vai residir, dita o domínio de aplicação. Aspectos sociais, políticos e organizacionais, bem como processos de trabalho existentes e problemas a serem resolvidos pelo sistema, precisam ser descritos em relação a metas e questões do negócio.
- Identificação de Fontes de Requisitos: requisitos podem estar espalhados em várias fontes e podem existir em vários formatos. Assim, podem existir muitas fontes de requisitos para um sistema e elas devem ser identificadas. Interessados representam a fonte de requisitos mais óbvia. Em especial, clientes, usuários e especialistas de domínio são os mais indicados para fornecerem informação detalhada sobre os problemas e as necessidades. Sistemas e processos existentes são também fontes de requisitos, especialmente quando o projeto envolve a substituição de um sistema existente. A documentação acerca desses sistemas e processos de negócio, incluindo manuais, formulários e relatórios, bem como de padrões da indústria, leis e regulamentações, provê informação útil sobre a organização e seu ambiente. Outras fontes incluem especificações de requisitos de sistemas (quando o sistema envolve hardware e software e existe uma especificação de mais alto nível), problemas reportados e solicitações de melhoria para sistemas correntes, e observação do dia a dia dos usuários. A necessidade de se obter requisitos a partir de múltiplas perspectivas e fontes ilustra bem a natureza de comunicação intensiva da Engenharia de Requisitos.
- Análise de Interessados: conforme citado anteriormente, interessados (*stakeholders*) são pessoas que têm interesse no sistema ou são afetadas de alguma maneira por ele e, portanto, precisam ser consultadas durante o levantamento de requisitos. Interessados incluem tanto pessoal interno quanto externo à organização. O cliente ou patrocinador do projeto é tipicamente o interessado mais aparente de um projeto. Contudo, os usuários são, na maioria das vezes, os interessados mais importantes. Outras partes cuja esfera de interesse pode ser afetada pela operação do sistema, tais como parceiros e clientes da organização, devem ser consideradas interessadas. Assim, um dos primeiros passos no processo de levantamento de requisitos consiste em analisar e envolver todos os interessados relevantes.
- Seleção de Técnicas de Levantamento de Requisitos: Nenhuma técnica individualmente é suficiente para levantar requisitos. Além disso, a escolha das técnicas a serem adotadas é fortemente dependente de características do projeto e de seus envolvidos. Diferentes técnicas devem ser empregadas, visando capturar diferentes tipos de informação e em diferentes estágios do processo de levantamento de requisitos. Assim, é importante selecionar adequadamente as técnicas a serem aplicadas.
- Levantamento de Requisitos de Interessados e Outras Fontes: Uma vez identificados as fontes de requisitos e os interessados relevantes, o levantamento de requisitos propriamente dito pode ser iniciado, aplicando-se as técnicas selecionadas.

Assim, antes de iniciar a descoberta de requisitos propriamente dita, ou mesmo durante o levantamento de requisitos, é útil realizar algumas tarefas, a saber (KOTONYA; SOMMERVILLE, 1998):

- Compreender os objetivos gerais do negócio a ser apoiado, esboçar uma descrição do problema a ser resolvido e identificar por que o sistema é necessário e quais são restrições sobre o mesmo, tais como restrições orçamentárias, de cronograma e de interoperabilidade.
- Levantar informações do contexto do desenvolvimento, dentre eles conhecimento acerca da organização onde o sistema será implantado, informações sobre o domínio da aplicação e informações sobre sistemas que estão em uso e serão substituídos pelo sistema em desenvolvimento.
- Organizar as informações levantadas, descartando conhecimento irrelevante e priorizando as metas da organização. Além disso, é importante identificar interessados (*stakeholders*) e seus papéis na organização.

O envolvimento de clientes e usuários é um fator crítico para o sucesso do projeto. Assim, é importante engajar representantes deles desde o início do projeto. Para definir esses representantes, deve-se (WIEGERS, 2003):

- Identificar diferentes classes de usuários. Usuários podem ser agrupados por diferentes aspectos, tais como: (i) a frequência com que usam o sistema, (ii) experiência no domínio de aplicação e perícia com sistemas computadorizados, (iii) características do sistema que eles usam, (iv) tarefas que eles realizam no apoio a seus processos de negócio e (v) níveis de privilégio de acesso e segurança.
- Selecionar e trabalhar com indivíduos que representem cada grupo de usuários;
- Estabelecer um acordo sobre quem serão as pessoas responsáveis por tomar decisões relativas a requisitos, sobretudo no que concerne a estabelecer prioridades e resolver conflitos.

Cada classe de usuários tem seu próprio conjunto de requisitos, tanto funcionais quanto não funcionais. Além disso, há classes de usuários que são mais importantes que outras. Essas classes devem ter tratamento preferencial na definição de prioridades e resolução de conflitos (WIEGERS, 2003).

Ainda em relação à seleção de usuários, deve-se evitar obter requisitos de intermediários entre a fonte de requisitos efetivamente e os analistas. Essa prática abre espaço para problemas de comunicação e, portanto, sempre que possível, deve-se ir diretamente à fonte. Contudo, alguns desses intermediários podem adicionar informação importante. Neste caso, ouça-os também. Entretanto, tome cuidado com gerentes de usuário (e também com desenvolvedores) que pensam que sabem as necessidades dos usuários sem sequer consultá-los (WIEGERS, 2003).

No que se refere aos responsáveis por tomar decisões relativas a requisitos, no início do projeto deve-se definir quem vai resolver requisitos conflitantes advindos de diferentes classes de usuário, reconciliar inconsistências, definir prioridades e arbitrar questões de escopo que venham a surgir. Uma boa estratégia é a tomada de decisão consultiva e participativa, na qual se obtém ideias e opiniões de diversos interessados antes de se tomar uma decisão (WIEGERS, 2003).

É interessante notar que, ao longo do processo de levantamento de requisitos, o engenheiro de requisitos (ou analista de sistema, como é mais conhecido popularmente) desempenha diversos papéis e assume diferentes responsabilidades. Por exemplo, um analista frequentemente desempenha o papel de um facilitador em sessões de levantamento de requisitos

em grupo, guiando e apoiando os participantes no endereçamento de questões relevantes, bem como garantindo que os participantes se sentem confortáveis e seguros com o processo, tendo oportunidade suficiente para contribuir. Outro papel importante é o de mediador. Em muitos casos, a priorização de requisitos ou negociação de requisitos conflitantes por diferentes classes de interessados é fonte de debate e disputa. Neste contexto, o analista deve negociar uma solução adequada e obter o compromisso dos envolvidos (AURUM; WOHLIN, 2005).

3.2 – Técnicas de Levantamento de Requisitos

Uma vez que levantar requisitos não é tarefa fácil, é imprescindível que essa atividade seja cuidadosamente conduzida, fazendo uso de diversas técnicas. Dentre as diversas técnicas que podem ser aplicadas para o levantamento de requisitos, destacam-se: entrevistas, questionários, workshops de requisitos, observação, investigação de documentos, prototipagem, cenários, abordagens baseadas em objetivos e reutilização de requisitos.

Kendall e Kendall (2010) classificam os métodos de levantamento de requisitos em dois grandes grupos: métodos interativos e métodos não obstrutivos. Os métodos interativos envolvem a interação com membros da organização, como é o caso de entrevistas e workshops de requisitos. Os métodos não obstrutivos procuram não interferir no trabalho dos membros da organização. Este é o caso de métodos como observação e investigação de documentos.

Alexander e Beus-Dukic (2009), por sua vez, organizam as técnicas de levantamento de requisitos pelo contexto no qual pode se dar a descoberta de requisitos. Esses contextos podem ser a descoberta de requisitos a partir de indivíduos (entrevistas, por exemplo), a partir de grupos (p.ex., workshops de requisitos) ou a partir de coisas (p.ex., investigação de documentos).

Os principais métodos de levantamento de requisitos envolvem um processo geral que contém as seguintes atividades:

- Planejamento: visa definir o objetivo da atividade de levantamento de requisitos a ser conduzida, as pessoas (no caso de métodos interativos) ou as coisas (no caso de métodos não obstrutivos) envolvidas, quando a atividade vai ser realizada e sua duração, onde e como ela vai ser realizada, incluindo material de apoio. Assim, o planejamento envolve cinco perguntas básicas: (i) Por quê? (ii) Quem? (ou O quê?) (iii) Quando? (iv) Onde? (v) Como?
- Condução: é a realização da atividade de levantamento de requisitos propriamente dita.
- Registro: consiste no registro das informações obtidas na atividade realizada.
- Validação dos achados: envolve submeter o registro das informações obtidas para avaliação pelas pessoas que participaram da atividade de levantamento de requisitos.

A seguir algumas das técnicas citadas anteriormente são apresentadas.

3.2.1 – Entrevistas

Entrevistas são, provavelmente, a técnica mais comumente utilizada no levantamento de requisitos (AURUM; WOHLIN, 2005). Uma entrevista é uma conversa direcionada com um propósito específico, que utiliza um formato "pergunta-resposta" (KENDALL; KENDALL, 2010). Entrevistas são usadas em quase todos os esforços de levantamento de requisitos. Nelas, os analistas formulam questões para os interessados e os requisitos são derivados das respostas a essas perguntas (SOMMERVILLE, 2007).

Uma entrevista é feita tipicamente por meio de uma reunião envolvendo o analista (entrevistador) e um interessado no sistema (entrevistado). Assim, é um método interativo de levantamento de requisitos a partir de um indivíduo. Contudo, uma entrevista pode envolver mais de um entrevistador e mais de um entrevistado. Uma entrevista pode ser, por exemplo, realizada por dois entrevistadores, um fazendo a maior parte das perguntas, enquanto o outro registra as informações obtidas e presta atenção em requisitos possivelmente perdidos. É possível, ainda, entrevistar dois ou três interessados de uma só vez, mas deve-se evitar envolver pessoas demais, pois se pode perder o controle da entrevista e o diálogo descambar para uma discussão geral (ALEXANDER; BEUS-DUKIC, 2009).

As entrevistas podem ser de dois tipos principais (SOMMERVILLE, 2007):

- Entrevistas fechadas, nas quais o interessado responde a um conjunto de perguntas predefinidas.
- Entrevistas abertas, nas quais não existe um roteiro predefinido. O analista explora vários assuntos com o interessado e, assim, desenvolve uma maior compreensão de suas necessidades.

Geralmente, as entrevistas são uma combinação desses dois tipos. As respostas a algumas perguntas podem levar a outros questionamentos, discutidos de maneira menos estruturada. As discussões completamente abertas dificilmente funcionam bem. A maioria das entrevistas requer algumas perguntas como ponto de partida e para manter o foco em um aspecto do sistema a ser desenvolvido (SOMMERVILLE, 2007).

Entrevistas são úteis para, dentre outros (SOMMERVILLE, 2007; KENDALL; KENDALL, 2010; KOTONYA; SOMMERVILLE, 1998):

- obter objetivos organizacionais e pessoais;
- obter um entendimento geral sobre o problema, sobre o que os interessados fazem e como eles podem interagir com o sistema;
- conhecer os sentimentos dos entrevistados sobre os sistemas atuais e as dificuldades que eles têm com os mesmos;
- levantar procedimentos informais para interação com tecnologias da informação.

No entanto, entrevistas podem não ser boas para o analista compreender ou aprender sobre o domínio da aplicação. Especialistas de domínio, muitas vezes, usam terminologia e jargões específicos, o que provoca mal entendidos por parte dos analistas. Além disso, alguns conhecimentos são tão familiares para os interessados que são considerados difíceis de explicar; outros são considerados tão básicos que os especialistas de domínio consideram que não vale a pena mencioná-los (SOMMERVILLE, 2007).

Em uma entrevista, o engenheiro de requisitos está, provavelmente, estabelecendo um relacionamento com uma pessoa estranha a ele. Assim, é importante: (i) construir uma base de confiança e entendimento; (ii) manter o controle da entrevista; e (iii) vender a ideia do sistema, provendo informações relevantes ao entrevistado (KENDALL; KENDALL, 2010).

Uma vez que entrevistas são essencialmente atividades sociais envolvendo pessoas, sua efetividade depende em grande extensão da qualidade da interação entre os participantes. Assim, os resultados de entrevistas podem variar bastante, em função das habilidades dos entrevistadores (AURUM; WOHLIN, 2005). As informações obtidas em entrevistas complementam outras informações obtidas de documentos, observações de usuários etc. Assim, essa técnica deve ser usada em conjunto com outras técnicas de levantamento de requisitos (SOMMERVILLE, 2007).

Uma entrevista precisa ser planejada. Tipicamente, o planejamento de uma entrevista, assim como o de outras atividades de levantamento de requisitos, deve considerar as cinco perguntas básicas:

- *Por quê?* A primeira coisa a ser feita é estabelecer os objetivos da entrevista. As primeiras entrevistas têm, normalmente, um caráter exploratório, quando se desejam capturar objetivos da organização para o sistema, propósito do sistema, áreas de negócio afetadas etc. Na medida em que o analista ganha entendimento sobre o problema, seu foco tende a ficar mais restrito, visando um aprofundamento em um tema ou aspecto específico do sistema. Entrevista é uma boa opção, dentre outros, para capturar metas (organizacionais ou pessoais) e sentimentos e necessidades em relação ao sistema (perspectivas de diferentes envolvidos), ou para melhorar/aprofundar o entendimento sobre o problema. Por outro lado, não é uma boa opção para aprender sobre o domínio.
- *Quem?* Tendo em mente o objetivo da entrevista, o próximo passo é identificar quais membros da organização têm conhecimento acerca do assunto a ser tratado e selecionar as pessoas a serem entrevistadas. É interessante levantar, ainda, o papel e a posição do potencial entrevistado na organização. Pessoas da alta gerência têm normalmente uma visão mais abrangente dos objetivos organizacionais e estratégicos, mas, por outro lado, não conhecem detalhes mais operacionais. Assim, o objetivo da entrevista deve guiar a seleção do entrevistado. O cliente ou patrocinador do projeto pode ajudar na identificação das pessoas mais indicadas para uma entrevista. Quando houver muitos bons candidatos a entrevistas em um mesmo papel/posição, pode-se usar amostragem para selecionar uma amostra gerenciável.
- *Quando?* No que se refere à questão temporal de uma entrevista, dois aspectos devem ser considerados: primeiro, a data e o horário; segundo, a duração. No que se refere ao agendamento da entrevista, deve-se marcar a entrevista com certa antecedência (preferencialmente de alguns dias) e informar o objetivo da entrevista e o tema a ser abordado, de modo que o entrevistado possa se preparar para responder às perguntas. No que se refere à duração, deve-se ter em mente que o entrevistado vai interromper seu trabalho para atender o analista. Assim, deve-se evitar tomar muito o seu tempo. Entrevistas com pontos de discussão focados devem ter, em média, uma hora de duração. Entrevistas exploratórias, ou em situações especiais, podem durar um pouco mais (até duas horas). Em qualquer caso, bom senso é fundamental e a preparação para a entrevista deve ser feita com cuidado para

aproveitar ao máximo a oportunidade, sobretudo quando a entrevista envolve membros da alta gerência.

- *Onde?* Definir o local onde se dará a entrevista. Normalmente, o analista vai até o local de trabalho do entrevistado.
- *Como?* Conhecendo o objetivo, o entrevistado (e seu perfil) e o tempo disponível, resta preparar a entrevista cuidadosamente para que a mesma seja o mais produtiva possível. A preparação envolve, dentre outros, a definição do tipo das questões a serem feitas, a redação das questões propriamente dita, a definição da ordem em que as perguntas serão feitas e a definição de como a entrevista será registrada durante a sua condução.

Visando responder às questões acima, Kendall e Kendall (2010) sugerem que o planejamento da entrevista envolva os seguintes passos:

1. *Estudar material existente sobre o domínio e a organização*. Atenção especial deve ser dada à linguagem usada pelos membros da organização, procurando estabelecer um vocabulário comum a ser usado na elaboração das questões da entrevista. Este passo visa, sobretudo, otimizar o tempo despendido nas entrevistas, evitando-se perguntar questões básicas e gerais.
2. *Estabelecer objetivos*. De maneira geral, há algumas áreas sobre as quais o analista desejará fazer perguntas, tais como fontes de informação, formatos da informação, frequência na tomada de decisão, estilo da tomada de decisão etc.
3. *Decidir quem entrevistar*. É importante incluir na lista de entrevistados as pessoas-chave das diversas classes de interessados afetados pelo sistema. O cliente pode ajudar nesta seleção.
4. *Preparar o entrevistado*. Uma entrevista deve ser marcada com antecedência, de modo que o entrevistado tenha tempo para pensar sobre a entrevista.
5. *Preparar a entrevista*. Deve-se decidir, dentre outros, sobre os tipos de questões e a estrutura da entrevista e o modo como a mesma será registrada.

Em relação ao tipo, questões podem ser de três tipos principais (KENDALL; KENDALL, 2010):

- *Questões subjetivas*: permitem respostas abertas. Ex.: "O que você acha de [...]?"; "Explique como você [...]?". Seus pontos positivos são: (i) proveem riqueza de detalhes; (ii) revelam novos questionamentos; e (iii) colocam o entrevistado mais à vontade, permitindo maior espontaneidade. Contudo, há também desvantagens, dentre elas: (i) podem resultar em muitos detalhes irrelevantes; (ii) podem levar à perda do controle da entrevista; (iii) podem ter respostas muito longas para se obter pouca informação útil; e (iv) podem dar a impressão de que o entrevistador está perdido e sem objetivo.
- *Questões objetivas*: limitam as respostas possíveis. Ex: "Quantos [...]?"; "Quem [...]?"; "Quanto tempo [...]?"; "Qual das seguintes informações [...]?". Como vantagens, questões objetivas: (i) permitem ganho de tempo, uma vez que elas vão direto ao ponto em questão, (ii) permitem manter o controle da entrevista e (iii) levam a dados relevantes. Como desvantagens, podem ser citadas: (i) questões objetivas podem ser maçantes para o entrevistado, (ii) podem falhar na obtenção de

detalhes importantes e (iii) não constroem uma afinidade entre entrevistador e entrevistado.

- *Questões de aprofundamento*: permitem explorar os detalhes de uma questão. Podem ser subjetivas ou objetivas. Ex: "Por quê?"; "Você poderia dar um exemplo?"; "Como isto acontece?".

A Tabela 3.1 apresenta um quadro comparativo de características obtidas com questões objetivas e subjetivas.

Tabela 3.1 – Quadro Comparativo Questões Objetivas *x* Subjetivas (adaptado de (KENDALL; KENDALL, 2010)).

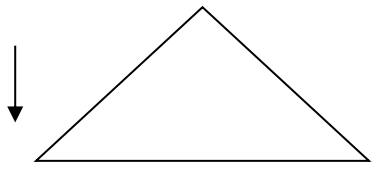
	Subjetivas	Objetivas
Confiabilidade dos dados	Baixa	Alta
Uso eficiente do tempo	Baixo	Alto
Precisão dos dados	Baixa	Alta
Amplitude e profundidade	Alta	Baixa
Habilidade requerida do entrevistador	Alta	Baixa
Facilidade de análise	Baixa	Alta

A elaboração de questões deve ser cuidadosa, pois ela pode levar a problemas como (KENDALL; KENDALL, 2010):

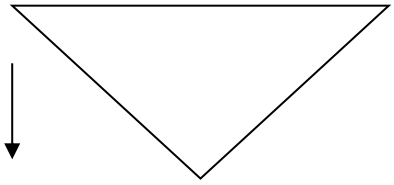
- *Questões tendenciosas*: tendem a levar o entrevistado a responder de uma forma específica. Ex.: Sobre este assunto, você está de acordo com os outros diretores? Uma opção mais adequada seria: O que você pensa sobre este assunto?
- *Duas questões em uma*: O entrevistado pode responder a apenas uma delas, ou pode se confundir em relação à pergunta que está respondendo. Ex.: O que você faz e como?

A estrutura de uma entrevista diz respeito à organização das questões em uma sequência lógica. De acordo com (KENDALL; KENDALL, 2010), há três formas básicas de se organizar as questões de uma entrevista:

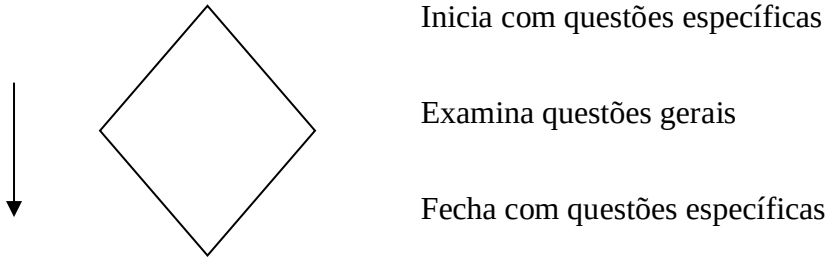
- *Estrutura de Pirâmide* (abordagem indutiva): inicia com questões mais detalhadas e objetivas e, à medida que a entrevista progride, questões mais gerais, subjetivas, são colocadas. Útil para situações em que o entrevistado necessita de um "aquecimento" para falar no assunto ou quando o analista deseja obter uma finalização sobre o assunto.



- *Estrutura de Funil* (abordagem dedutiva): inicia com questões gerais subjetivas e, à medida que a entrevista avança, perguntas mais específicas, usando questões objetivas, são feitas. Essa estrutura provê um meio fácil e mais amigável para se começar uma bateria de entrevistas. Permite levantar informação detalhada, sendo desnecessárias longas sequências de questões objetivas e de aprofundamento.



- *Estrutura de Diamante*: é uma combinação das duas anteriores, começando com questões específicas, passando a questões mais gerais e fechando a entrevista novamente com questões específicas. É uma boa forma de se estruturar uma entrevista, já que mantém o interesse do entrevistado em uma variedade de questões. Contudo, tende a ser mais longa.



Estruturar entrevistas é um meio de planejar a priori a ordem em que as questões serão feitas. Obviamente, há a opção de não se definir essa ordem antecipadamente, em uma abordagem de entrevista não estruturada, na qual não há uma definição da sequência das questões. De acordo com o andar da entrevista, caminhos possíveis são avaliados e a sequência é estabelecida. Normalmente, requer mais tempo e, sobretudo, experiência do entrevistador. Entretanto, vale ressaltar que, ainda que a sequência das questões não seja definida a priori, as questões devem ser definidas antecipadamente, ou seja, o planejamento é necessário. A Tabela 3.2 apresenta um quadro comparativo entre as abordagens estruturada e não estruturada.

Tabela 3.2 – Comparação entre as Abordagens Estruturada e Não Estruturada (adaptado de (KENDALL; KENDALL, 2010)).

	Não Estruturada	Estruturada
Avaliação	Mais Difícil	Mais Fácil
Tempo Requerido	Maior	Menor
Treinamento Requerido	Maior	Menor
Espontaneidade	Maior	Menor
Oportunidades para "*insight*"	Maior	Menor
Flexibilidade	Maior	Menor
Controle	Menor	Maior
Precisão	Menor	Maior
Confiabilidade	Menor	Maior
Amplitude e Profundidade	Maior	Menor

Por fim, o planejamento deve definir de que forma a entrevista será registrada. É importante registrar os principais aspectos de uma entrevista durante a sua realização, pois, caso contrário, as informações obtidas podem ser perdidas logo em seguida. Há duas formas principais, cujas vantagens e desvantagens são apresentadas a seguir (KENDALL; KENDALL, 2010) (ALEXANDER; BEUS-DUKIC, 2009):

- *Gravação / Filmagem*: a entrevista é gravada ou filmada. A gravação / filmagem permite o registro completo da entrevista e a reprodução para outros membros da equipe posteriormente. Contudo, muitos entrevistados não gostam de serem gravados e, por conseguinte, ficam pouco à vontade. Além disso, o fato de saber que a entrevista está sendo gravada pode deixar o entrevistador distraído, comprometendo seu trabalho. Por fim, transcrever registros gravados é uma atividade que consome tempo e confiar em gravações para posteriormente tirar dúvidas pode ser perigoso, deixando de se esclarecer uma dúvida na hora da entrevista.
- *Anotações*: o entrevistador toma notas durante a entrevista. Quando essa abordagem é adotada, deve-se considerar que escrever é um processo lento, enquanto falar é rápido. Assim, as anotações devem capturar a essência do que foi dito. Algumas das vantagens dessa abordagem são: (i) mantém o entrevistador alerta; (ii) um esquema das anotações a serem feitas pode ser usado para fornecer um roteiro para a entrevista; (iii) mostra interesse e preparação do entrevistador. Como desvantagens, podem ser citadas: (i) essa abordagem pode comprometer o andamento da conversa; (ii) pode se dar excessiva atenção a fatos e pouca a sentimentos e opiniões.

Uma vez planejada, a entrevista deve ser realizada. Na véspera da reunião, recomenda-se que o analista entre em contato com o entrevistado para confirmar o horário e o local da entrevista. No dia da entrevista, o analista deve chegar um pouco antes do horário marcado, vestido apropriadamente. Ao iniciar a entrevista, é interessante que o analista se apresente, fale

sucintamente sobre os objetivos da entrevista, diga ao entrevistado o que será feito com as informações coletadas e assegure seu aspecto confidencial. Durante a entrevista, é fundamental gerenciar o tempo. Ao término da entrevista, pergunte se há algo mais sobre o assunto que o entrevistado ache importante você saber, faça um resumo da entrevista e dê um retorno acerca de suas impressões gerais. Informe o entrevistado sobre os passos seguintes e pergunte se há outra pessoa com a qual ele acha que você deveria conversar. Quando for o caso, marque nova entrevista (KENDALL; KENDALL, 2010).

Ao finalizar a entrevista, resta, ainda, escrever um relatório sobre a mesma. O relatório ou ata da entrevista deve capturar os pontos principais da entrevista e deve ser escrito tão rápido quanto possível para assegurar a qualidade (KENDALL; KENDALL, 2010). De maneira geral, os seguintes itens devem ser registrados: entrevistado(s), entrevistador(es), data e hora, duração, assunto, objetivos e principais pontos discutidos. Uma vez escrita, a ata de entrevista deve ser enviada para avaliação dos participantes (validação de achados).

3.2.2 – Técnicas de Coleta Colaborativa de Requisitos

A coleta colaborativa de requisitos é uma técnica muito comumente empregada. Grupos são particularmente efetivos, porque eles envolvem e estabelecem o compromisso diretamente com os interessados e porque promovem cooperação (AURUM; WOHLIN, 2005). Há muitas abordagens diferentes de coleta colaborativa de requisitos, tais como Workshops de Requisitos, JAD e *Brainstorming*. Todas aplicam, de alguma maneira, as seguintes diretrizes básicas (PRESSMAN, 2006): (i) as reuniões envolvem representantes de diferentes grupos de interessados, sendo estabelecidas regras de preparação e participação; (ii) um facilitador, que pode ser o analista ou outro participante, controla a reunião; (iii) mecanismos de anotação, tais como quadro branco, *flipcharts*, anotações em um computador projetadas para todos os participantes etc., são usados para registrar as ideias levantadas; e (iv) a meta é identificar ou debater um problema, propor elementos da solução, negociar diferentes abordagens e especificar um conjunto preliminar de requisitos da solução.

Geralmente, o facilitador desempenha um papel crítico no planejamento, seleção dos participantes e, sobretudo, na condução da reunião, de modo a encorajar a participação para se atingir objetivos de modo consensual. Se uma pessoa está participando pouco, o facilitador deve tentar trazê-la para a discussão, dando espaço para ela se manifestar (WIEGERS, 2003).

Em alguma extensão, o trabalho em grupo se sobrepõe ao trabalho com indivíduos (entrevistas). Entretanto, um grupo reúne informações de várias fontes, coloca-as juntas, permite ouvir vários pontos de vista, refina o entendimento coletivo e atinge concordância de uma maneira que não é possível com outras técnicas de levantamento de requisitos (ALEXANDER; BEUS-DUKIC, 2009).

Para ser bem sucedido, os participantes devem estar de acordo com alguns princípios operacionais básicos, tais como começar e terminar a reunião nos horários predefinidos, manter apenas uma conversa de cada vez, permitir a contribuição de todos e enfocar comentários e críticas em questões e não em indivíduos. Diferentes interessados tipicamente têm um objetivo comum, mas têm diferentes visões do problema e objetivos específicos distintos. Ninguém deve esquecer seus próprios objetivos específicos, mas o credo de todos os participantes deve ser: "Meu ponto de vista é um dentre vários e o mais importante é o objetivo final comum". Deve-se considerar, ainda, que o sucesso de uma reunião de coleta colaborativa de requisitos depende fortemente da habilidade do facilitador e dos participantes trabalharem como uma equipe

inovadora. Além disso, deve ser possível dar voz a todos os participantes (WIEGERS, 2003; ALEXANDER; BEUS-DUKIC, 2009).

Uma técnica de coleta colaborativa de requisitos são os Workshops de Requisitos. Workshop de Requisitos é o nome dado a um número de diferentes tipos de reuniões envolvendo diversas pessoas, cujo foco é a descoberta e desenvolvimento de requisitos (AURUM; WOHLIN, 2005). Workshops de requisitos colocam um grupo de pessoas junto, com o objetivo comum de levantar requisitos para um problema compartilhado, para o qual essas pessoas têm visões distintas. O propósito é obter conhecimento e energia suficientes para levantar requisitos rápida e eficientemente (ALEXANDER; BEUS-DUKIC, 2009).

Um workshop de requisitos não é simplesmente uma reunião. Um workshop é uma reunião com propósito definido e atividades planejadas. Assim, requer planejamento endereçando as cinco questões básicas:

- *Por quê?* A primeira coisa a ser feita é estabelecer os objetivos do workshop. Workshops são provavelmente o principal meio de tomar decisões e fechar um acordo entre membros de um grupo. Várias informações podem ser alvo de descoberta em um workshop de requisitos, dentre elas as influências que interessados têm uns sobre os outros, objetivos, riscos, fronteiras do sistema, restrições e atributos de qualidade (requisitos não funcionais) e prioridades (ALEXANDER; BEUS-DUKIC, 2009).
- *Quem?* Grupos pequenos (até seis participantes) tendem a funcionar melhor. É melhor organizar grupos menores em diferentes workshops do que ter um grupo muito grande. As pessoas devem ser convidadas em função dos objetivos e das atividades planejadas para o workshop e da contribuição que elas podem dar. Especialistas e pessoas com poder de decisão sobre os requisitos são bons candidatos a participantes (WIEGERS, 2003; ALEXANDER; BEUS-DUKIC, 2009).
- *Quando?* Da mesma forma que entrevistas, workshops devem ser agendados com antecedência, em datas acordadas com todos os participantes. No que se refere à duração, deve-se ter em mente que todos os participantes vão interromper seu trabalho para atender ao workshop. Assim, sessões de workshops devem ser tão curtas quanto possível, idealmente com duração de uma a duas horas. Contudo, alguns projetos podem requerer sessões mais longas, algumas vezes com duração de várias horas (ALEXANDER; BEUS-DUKIC, 2009).
- *Onde?* Deve-se definir o local onde a reunião será realizada. Deve-se garantir que haverá espaço suficiente para acomodar confortavelmente todos os participantes. O layout da sala também é importante. Layouts em círculo ou na forma de U funcionam bem, uma vez que eles colocam todos os participantes em um mesmo nível de importância.
- *Como?* Durante a preparação, devem-se definir os recursos necessários (computadores, projetores, quadros brancos, flipcharts etc.) e garantir que eles estarão disponíveis no período do workshop. Proveja o material de preparação necessário para os participantes com antecedência. Por fim, obviamente, a definição de tópicos a serem discutidos deve ser feita cuidadosamente. Tópicos complexos podem ser divididos em séries de questões mais simples, tais como: Quais os objetivos dos interessados para o assunto em questão? Que conflitos podem surgir? Como podemos ver isso funcionando (cenários)? Quais os argumentos de cada lado?

É possível imaginar diferentes soluções para o problema? Quais são prioridades? (ALEXANDER; BEUS-DUKIC, 2009).

Durante a condução do workshop, enfatize que o tempo é limitado e minimize distúrbios, solicitando, p.ex., que as pessoas desliguem seus celulares. Atribua papéis às pessoas, tais como responsável por controlar o tempo, responsável por controlar se a discussão está fugindo do assunto, responsável por fazer anotações etc. Uma vez terminado, as informações descobertas no workshop devem ser registradas em um relatório, o qual deve ser validado pelos participantes (ALEXANDER; BEUS-DUKIC, 2009).

Workshops de requisitos podem ser combinados com diversas técnicas. Durante um workshop de requisitos, por exemplo, cenários podem ser elaborados. Por outro lado, resultados obtidos em diversas entrevistas podem ser levados à discussão em um workshop.

Além dos Workshops de Requisitos, outras duas técnicas de coleta colaborativa de requisitos bastante utilizadas são:

- ***Brainstorming***: neste tipo de reunião, representantes de diferentes grupos de interessados engajam-se em uma discussão informal para gerar tantas ideias quanto possível, sem focar a atenção em nenhuma delas. Normalmente não é propósito de uma sessão de *brainstorming* resolver maiores questões ou tomar decisões. Essa técnica é frequentemente utilizada para desenvolver uma declaração preliminar da missão e dos requisitos para o sistema. Um diferencial dessa técnica é que ela promove a livre expressão, favorecendo a descoberta de soluções novas e inovadoras para problemas existentes (AURUM; WOHLIN, 2005).
- **JAD (*Joint Application Development*)**: envolve a participação de diferentes interessados na investigação, por meio de discussões, tanto de problemas a serem resolvidos quanto das soluções disponíveis para esses problemas. Com as diversas partes envolvidas representadas, decisões podem ser tomadas e questões resolvidas mais rapidamente (AURUM; WOHLIN, 2005). Em uma sessão JAD, ao contrário de uma sessão de *brainstorming*, as metas do sistema já estão definidas. Sessões JAD de requisitos[2] são focadas em um aspecto mais específico relacionado aos requisitos para o sistema.

3.2.3 – Questionários

Questionário ou *survey* é uma técnica de levantamento de informações que permite ao analista capturar, de várias pessoas afetadas pelo sistema, atitudes, crenças, comportamentos e características. Atitudes referem-se ao que as pessoas na organização dizem querer; crenças referem-se a o que as pessoas pensam ser realmente verdade; comportamento é o que as pessoas fazem; características são propriedades de pessoas ou coisas (KENDALL; KENDALL, 2010).

Há muitas similaridades entre questionários e entrevistas e pode ser útil utilizar as duas abordagens em conjunto para refinar respostas não claras de um questionário em uma entrevista ou para projetar um questionário com base no que foi descoberto em uma entrevista. Usando questionários após a realização de entrevistas, um analista pode quantificar o que foi levantado em entrevistas ou determinar como um sentimento expresso em uma entrevista é realmente difundido ou limitado. Por outro lado, questionários podem ser usados para examinar uma

[2] Sessões JAD podem ser aplicadas em outras fases do desenvolvimento de software, tal como projeto (design).

grande amostra de usuários para sentir problemas ou levantar questões importantes, antes de se programar entrevistas (KENDALL; KENDALL, 2010).

Questionários proveem um meio eficiente de coletar informações de vários interessados. Entretanto, são limitados no que tange à profundidade do conhecimento que pode ser levantado, uma vez que não permitem que um tópico seja aprofundado ou que ideias sejam expandidas (AURUM; WOHLIN, 2005). São úteis quando:

- Há um grande número de pessoas envolvidas no projeto do sistema e é necessário saber qual proporção de um dado grupo aprova ou desaprova uma particular característica do sistema proposto. É especialmente útil quando as pessoas necessárias estão geograficamente dispersas.
- Se deseja saber uma opinião global, obtida de muitas pessoas, antes de se definir qualquer direção específica para o projeto, em um estudo exploratório.

Uma vez que questionários e entrevistas seguem uma abordagem "pergunta-resposta", seria bastante razoável pensar que as considerações feitas para entrevistas aplicam-se também para questionários. Contudo, é importante ressaltar que há diferenças fundamentais entre essas técnicas e, portanto, outros aspectos devem ser considerados.

Em primeiro lugar, entrevistas permitem interação direta com o entrevistado a respeito das questões e seus significados. Em uma entrevista, o analista pode refinar uma questão, definir um termo obscuro, alterar o curso do questionamento e controlar o contexto de modo geral. Isto não é verdade para questionários de maneira geral e, portanto, o planejamento de um questionário e de suas questões deve ser mais cuidadoso. Assim, um questionário deve ter questões claras e não ambíguas, fluxo bem definido e administração planejada em detalhes. Além disso, devem-se levantar, antecipadamente, as potenciais dúvidas das pessoas que vão respondê-lo (KENDALL; KENDALL, 2010).

Em relação às cinco perguntas básicas, valem as seguintes considerações:

- *Por quê?* Assim como os demais métodos, a primeira coisa a ser feita é estabelecer os objetivos de um questionário. Questionários podem ser usados para quantificar o que foi levantado com outros métodos, para determinar como um sentimento capturado por meio de outras técnicas de levantamento de requisitos é realmente difundido ou limitado, ou para examinar uma grande amostra de usuários para sentir problemas ou levantar questões importantes.
- *Quem?* A definição de quem deverá responder o questionário deve ser feita em conjunto com a definição de seus objetivos. Devem-se determinar quais classes de pessoas são necessárias e o tipo de respondentes. As pessoas que vão efetivamente responder o questionário são escolhidas, dentre outros, em função de sua posição, tempo de serviço, responsabilidades e interesse no sistema corrente ou proposto. Neste contexto, pode-se usar amostragem. É importante garantir que um número suficiente de respondentes será incluído, de modo a permitir uma amostra razoável, considerando que algumas pessoas não vão responder ou vão responder erradamente e terão seus questionários descartados.
- *Quando? Onde?* Estas questões estão fortemente relacionadas ao método de aplicação do questionário a ser adotado. Quando se decide reunir todos os respondentes em um mesmo lugar e momento, deve-se definir local, data, hora e duração. Quando os respondentes são livres para administrar o preenchimento do

questionário, deve-se indicar até que data isso deve ser feito e qual a duração esperada para se responder o questionário.

- *Como?* Dentre os aspectos a serem considerados no projeto de um questionário, podem ser citados os tipos e a redação das questões, escalas e método de aplicação.

Questionários podem ter questões objetivas ou subjetivas. Questões subjetivas são particularmente adequadas a situações em que se deseja saber a opinião dos membros da organização acerca de algum aspecto do sistema, sendo impossível, portanto, listar efetivamente todas as respostas possíveis para uma pergunta. Quando se decidir utilizar questões subjetivas em um questionário, deve-se antecipar o tipo de resposta que se espera obter. Essas questões devem ser restritas o suficiente para guiar as pessoas, de modo que respondam de uma maneira específica (KENDALL; KENDALL, 2010). Deve-se tomar cuidado com perguntas que permitam respostas muito amplas, pois isso pode dificultar a comparação e a interpretação dos resultados.

Questões objetivas, por outro lado, devem ser utilizadas em um questionário quando o engenheiro de requisitos é capaz de listar as possíveis respostas e quando há uma grande amostra de pessoas a examinar. Respostas a questões objetivas são mais facilmente quantificadas, enquanto respostas a questões subjetivas são analisadas e interpretadas de maneira diferente. A Tabela 3.2 compara o uso de questões objetivas e subjetivas em questionários (KENDALL; KENDALL, 2010).

Tabela 3.2 – Uso de questões subjetivas e objetivas em questionários (adaptado de (KENDALL; KENDALL, 2010)).

	Questões Subjetivas	Questões Objetivas
Tempo gasto para responder	Alto	Baixo
Natureza exploratória	Alta	Baixa
Amplitude e profundidade	Alta	Baixa
Facilidade de preparação	Fácil	Difícil
Facilidade de análise	Difícil	Fácil

Assim como ocorrem com as entrevistas, a linguagem utilizada na elaboração de questionários é extremamente importante para a sua efetividade. É prudente escrever as questões de modo a refletir a terminologia particular do negócio. Assim, tanto as perguntas quanto as respostas serão mais fáceis de interpretar. Para verificar a linguagem utilizada, aplique o questionário antecipadamente em um grupo piloto, pedindo atenção à adequabilidade dos termos empregados. Kendall e Kendall (2010) recomendam que, dentre outras, as seguintes diretrizes sejam observadas na redação de um questionário:

- Sempre que possível, use o vocabulário das pessoas que irão responder. Prime pela simplicidade.
- Utilize perguntas simples e curtas.
- Evite redação tendenciosa.
- Garanta que as questões estão tecnicamente precisas antes de incluí-las no questionário.

Em questionários, escalas são usadas para atribuir números ou outros símbolos para um atributo ou característica com o propósito de medir esse atributo ou característica. Escalas são frequentemente arbitrárias e podem não ser únicas (por exemplo, escalas de temperatura em ºC, ºF, K). Dentre os tipos de escalas de medição comumente usados por analistas de sistemas (KENDALL; KENDALL, 2010), destacam-se:

- *Nominal*: utilizada para classificar coisas. Os valores da escala permitem apenas indicar se um indivíduo pertence ou não a uma classe ou se possui ou não certa característica. Não há qualquer relação de ordenação entre as classes. Assim, é a forma mais "fraca" de medição, uma vez que só obtém totais para cada classe.
  Ex: Que tipo de software você mais usa?
  1- Editor de Texto 2- Planilha 3- Gráfico 4- Outros
- *Ordinária*: também utilizada para classificar coisas, mas pressupõe-se que as diferentes classes estão ordenadas em um ranking, sem, no entanto, quantificar a magnitude das diferenças entre as classes.
  Ex: Qual a sua opinião sobre as telas de ajuda?
  1- Não ajudam nada 2- Ajudam pouco 3- Ajudam muito
- *Métrica*: além de ser possível ordenar os indivíduos, é possível também quantificar as diferenças entre eles. As escalas métricas dividem-se em dois subtipos:
  - Intervalar: é possível quantificar as distâncias entre as medições, mas não há um ponto nulo.
  - de Razão: não só é possível quantificar as diferenças entre as medições, como também estão garantidas certas condições matemáticas vantajosas, como um ponto de nulidade.

  As escalas métricas têm como traço marcante o fato de permitirem que sejam feitas operações matemáticas sobre os dados obtidos do questionário e, portanto, uma análise mais completa. Para obter essa vantagem, uma estratégia bastante utilizada consiste em tornar uma escala ordinária em uma escala métrica, como no exemplo abaixo.
  Quão útil é o suporte técnico do Centro de Informação?
  1- Nada útil 2 3 4 5- Extremamente útil

Claramente, originalmente ela não era uma escala de intervalo, mas o fato de ancorar a escala em ambas as extremidades e distribuir valores em um intervalo permite ao analista assumir que os respondentes vão perceber os intervalos como iguais, tornando análises quantitativas possíveis.

Na construção de escalas, alguns problemas podem ocorrer, a saber (KENDALL; KENDALL, 2010):

- *Condescendência*: a pessoa responde a todas as questões do mesmo jeito. Solução: de uma questão para a outra, mover a categoria "média" para a esquerda ou direita em relação ao centro.
- *Tendência Central*: a pessoa responde tudo "na média". Solução: tornar as diferenças menores nos extremos, ajustar a força dos descritores ou criar uma escala sem um valor que represente a média.
- *Efeito "Auréola"*: a impressão formada em uma questão é levada para a próxima. Solução: mesclar questões sobre objetos diferentes.

Um questionário relevante e bem projetado pode aumentar a taxa de respostas. As seguintes diretrizes podem ser úteis durante o projeto um questionário (KENDALL; KENDALL, 2010):

- Deixe espaços em branco.
- Para questões subjetivas, deixe espaço suficiente para as respostas.
- Em questões objetivas, torne fácil para os respondentes marcar claramente suas respostas.
- Seja consistente no estilo.

Questionários podem ser aplicados de diversas maneiras. Quando se decide aplicar um questionário por email ou pela Web, considerações adicionais de planejamento relativas à confidencialidade, autenticação de identidade e problemas com múltiplas respostas (a mesma pessoa responder mais de uma vez) devem ser levadas em conta. No caso de questionários disponíveis na Web, use formatos de entrada de dados comumente usados, tais como caixas de texto, *check boxes*, *radio buttons*, *drop-down menu* etc. (KENDALL; KENDALL, 2010).

Para ordenar as questões, considere os objetivos e, então, determine a função de cada questão para atingir esses objetivos. Use um grupo piloto para auxiliar ou observe o questionário com olhos de respondedor. Algumas orientações devem ser seguidas, dentre elas (KENDALL; KENDALL, 2010):

- Coloque questões que são importantes para os respondentes primeiro.
- Agrupe itens de conteúdo similar e observe tendências de associação.
- Coloque questões menos controversas primeiro.

Por fim, no que se refere ao método de aplicação do questionário, há diversas possibilidades, cada uma delas apresentando vantagens e desvantagens. Pode-se, por exemplo, reunir todos os respondedores em um mesmo local para a aplicação do questionário. Isso permite alto retorno, instruções uniformes e resultado rápido. Contudo, pode ser difícil reunir todas as pessoas em só lugar ao mesmo tempo e o respondedor pode ter coisas importantes a fazer naquele momento (KENDALL; KENDALL, 2010).

A forma mais comum é permitir que os respondentes administrem seu tempo, decidindo quando vão responder o questionário. Neste caso, corre-se o risco das pessoas esquecerem ou propositalmente ignorarem o questionário. Contudo, reforça-se a sensação de anonimato e, por conseguinte, podem-se obter respostas mais confiáveis (KENDALL; KENDALL, 2010).

Aplicar questionários eletronicamente, seja por email seja por meio de formulário na Web, é um meio de rapidamente chegar aos respondentes. Evitam-se custos com cópias, as respostas podem ser dadas de acordo com a conveniência dos respondedores e automaticamente coletadas e armazenadas eletronicamente. Alguns sistemas para aplicar questionários permitem que o respondedor comece a responder, salve suas respostas e volte ao questionário posteriormente para completar seu preenchimento. Lembretes aos respondentes podem ser facilmente enviados por email, assim como notificações ao analista sobre quando o respondente abriu a mensagem. Pesquisas mostram que questionários pela Web podem encorajar respostas francas e consistentes. Questões que podem ser difíceis de serem colocadas pessoalmente podem ser aceitáveis em um questionário pela Web (KENDALL; KENDALL, 2010).

3.2.4 - Observação

As pessoas, muitas vezes, têm dificuldade em articular detalhes de seu trabalho, pois estão imersas nele e fazem muitas coisas de maneira intuitiva. Contudo, os contextos social e organizacional em que as pessoas trabalham são importantes para o desenvolvimento de um sistema e podem derivar requisitos e restrições (SOMMERVILLE, 2007). Assim, observar o comportamento e o ambiente do indivíduo pode ser uma forma bastante eficaz de levantar informações que, tipicamente, passam despercebidas quando outras técnicas são usadas (KENDALL; KENDALL, 2010).

A etnografia é o estudo de pessoas em seu ambiente natural. No contexto do levantamento de requisitos, envolve a participação ativa ou passiva do analista nas atividades normais dos usuários, durante um período de tempo, enquanto coleta informações a respeito dos processos sendo realizados. Técnicas de etnografia são especialmente úteis para endereçar fatores contextuais e de ambientes de trabalho cooperativo (AURUM; WOHLIN, 2005).

A observação é uma das técnicas de etnografia mais usadas no levantamento de requisitos. Como o próprio nome indica, o analista observa os usuários executando os processos, sem interferência direta (AURUM; WOHLIN, 2005). Ela é empregada para compreender requisitos sociais e organizacionais, bem como para compreender como as tarefas são realizadas efetivamente. O analista se insere no ambiente de trabalho onde o sistema será usado, observa o trabalho do dia-a-dia e faz anotações acerca das tarefas reais nas quais os participantes estão envolvidos (SOMMERVILLE, 2007).

Através da observação é possível capturar (SOMMERVILLE, 2007; KENDALL; KENDALL, 2010):

- Requisitos derivados da maneira como as pessoas realmente trabalham e não da maneira como os processos são documentados ou explicados. É possível derivar requisitos implícitos que refletem os processos reais (e não os formais) com os quais as pessoas estão envolvidas.
- Requisitos derivados do relacionamento entre o indivíduo que toma decisões e outros membros da organização. Esses requisitos são derivados da colaboração e do conhecimento das atividades de outras pessoas.

Por outro lado, essa técnica não é apropriada para obter informações de domínio, bem como pode ser difícil identificar novas características a serem acrescentadas ao sistema. Assim, a observação deve ser combinada com outras técnicas de levantamento de requisitos. Quando aplicadas em conjunto, observação pode ser usada para confirmar ou negar informações de entrevistas e/ou questionários. Também se podem entrevistar as pessoas observadas para completar informações obtidas de uma observação. A observação pode ser combinada também com a prototipagem. Uma vez construído um protótipo, podem-se observar os usuários utilizando o protótipo, de modo a avaliar o mesmo e derivar novos requisitos (SOMMERVILLE, 2007; KENDALL; KENDALL, 2010; KOTONYA; SOMMERVILLE, 1998). Além disso, deve-se ressaltar que a efetividade de uma observação pode variar na medida em que os usuários têm tendência a ajustar o modo como realizam suas tarefas quando sabem que estão sendo observados (AURUM; WOHLIN, 2005).

Como outras técnicas de levantamento de requisitos, a observação envolve planejamento, condução e o registro de resultados. No planejamento, o analista deve definir o que observar, quem observar, quando, onde, por que e como. Kotonya e Sommerville (1998)

apontam que não há uma forma padrão de se conduzir estudos etnográficos, contudo, indicam algumas diretrizes para a aplicação dessa técnica, dentre elas:

- É muito importante despender um tempo conhecendo as pessoas envolvidas e estabelecendo uma relação de confiança.
- Deve-se assumir que as pessoas que estão sendo observadas são boas em seu trabalho e procurar capturar meios não padronizados de trabalhar. Esses meios frequentemente apontam para eficiências no processo de trabalho que foram incorporadas a partir da experiência individual.
- Devem-se tomar notas das práticas de trabalho durante a observação e redigir um relatório. É possível aprender bastante com os detalhes de como as pessoas trabalham. Somente após diversos desses detalhes terem sido coletados, é que um quadro coerente vai emergir.
- É útil que o analista, antes de iniciar o trabalho, informe as pessoas e diga como a observação vai ser conduzida e seu propósito.

No que se refere à definição de quando realizar a observação, é importante não considerar apenas se o indivíduo (ou indivíduos) a ser observado estará trabalhando nos processos de interesse no período agendado, mas também se esse processo de negócio de interesse tem uma ocorrência significativa no período considerado.

3.2.5 – Prototipagem

Muitas vezes as pessoas acham difícil visualizar como um requisito especificado na forma de uma sentença escrita ou por um conjunto de modelos vai se materializar em um sistema de software. De maneira geral, pessoas têm dificuldade de descrever suas necessidades sem ter algo tangível à sua frente. Nesses casos, se um protótipo do sistema é desenvolvido para demonstrar requisitos, fica mais fácil para usuários e outros interessados encontrar problemas e sugerir como os requisitos podem ser melhorados. Afinal, criticar é mais fácil do que criar (KOTONYA; SOMMERVILLE, 1998; WIEGERS, 2003).

Um protótipo é uma versão inicial do sistema que é desenvolvida no início do processo de desenvolvimento. No contexto da Engenharia de Requisitos, um protótipo é desenvolvido com o propósito de apoiar o levantamento e, sobretudo, a validação de requisitos. Assim, nesse contexto, uma característica essencial de um protótipo é que ele seja desenvolvido rapidamente (KOTONYA; SOMMERVILLE, 1998).

A prototipagem é uma técnica valiosa para o levantamento de requisitos. Ela torna os requisitos mais reais e diminui lacunas de entendimento. Ao colocar o usuário na frente de uma porção inicial ou uma imitação do sistema, a prototipagem estimula os usuários a pensar e a estabelecer um diálogo sobre os requisitos. As considerações tecidas sobre o protótipo ajudam a se obter um entendimento compartilhado dos requisitos (WIEGERS, 2003). Além disso, a prototipagem é muito útil quando os interessados estão pouco familiarizados com as soluções disponíveis (AURUM; WOHLIN, 2005). Este é o caso, por exemplo, da introdução de novas tecnologias.

Protótipos podem servir a outros propósitos, além do propósito de clarear e completar o entendimento sobre requisitos. Protótipos podem ser usados para explorar alternativas de projeto (design), sobretudo no projeto de interfaces com o usuário, bem como a prototipagem

pode ser usada como parte de uma estratégia de desenvolvimento, na qual protótipos iniciais vão sendo gradativamente transformados no produto final, através de uma sequência de iterações de desenvolvimento. Contudo, do ponto de vista da Engenharia de Requisitos, foco deste texto, a principal razão para se desenvolver um protótipo é resolver incertezas a respeito dos requisitos do sistema o mais cedo possível. Assim, essas incertezas devem ser usadas para decidir que partes do sistema prototipar e o que se espera aprender com as avaliações do protótipo (WIEGERS, 2003).

A prototipagem permite capturar as reações iniciais do usuário em relação ao sistema. Essas reações podem ser obtidas através de observação, entrevistas ou questionário e podem ser usadas pelo engenheiro de requisitos para guiar iniciativas na direção de melhor atender as necessidades dos usuários, bem como para ajudar a estabelecer (ou rever) prioridades e redirecionar planos. Usuários, por sua vez, podem vislumbrar novas capacidades, não imaginadas antes da interação com o protótipo e que surgiram da experimentação com o mesmo (KENDALL; KENDALL, 2010):

Diferentes tipos de protótipos podem ser desenvolvidos, sendo que diferentes autores denominam esses tipos de forma diferente. Quanto às camadas da arquitetura que são efetivamente implementadas, um protótipo pode ser:

- *Protótipo não-operacional* ou *de interface*: quando apenas a camada de interface com o usuário é implementada; as demais camadas da arquitetura do sistema não são e, portanto, o sistema não faz nenhum processamento propriamente dito. Esse tipo de protótipo é útil para avaliar certos aspectos do sistema quando a codificação requerida pela aplicação é custosa e a noção básica do que é o sistema pode ser transmitida pela análise de suas interfaces (KENDALL; KENDALL, 2010). Protótipos não operacionais mostram as opções de funções que estarão disponíveis para o usuário e a aparência das interfaces. Contudo não contêm funcionalidade real. Às vezes, a navegação pode funcionar, mas em alguns pontos o usuário pode se deparar apenas com uma mensagem descrevendo o que seria realmente mostrado. Os dados que aparecem em resposta a uma consulta são apenas ilustrativos e muitas vezes são valores constantes definidos no próprio código do protótipo. Mesmo quando isso é feito, deve-se usar dados reais para realçar a validade do protótipo como um modelo do sistema real. Normalmente, essa simulação é boa o suficiente para os usuários fazerem um julgamento se há funcionalidade faltando, errada ou desnecessária (WIEGERS, 2003).
- *Protótipo operacional*: funciona como se supõe que o sistema real deveria funcionar e implementa, de alguma forma, todas as camadas da arquitetura do sistema. Esse tipo de protótipo é tipicamente usado para reduzir riscos durante o projeto, podendo ser usado, dentre outros, para avaliar se uma arquitetura é viável e sólida, ou para testar requisitos críticos. Protótipos operacionais devem ser construídos usando ferramentas de produção, ou seja, as mesmas usadas para construir o próprio sistema (WIEGERS, 2003).

Quanto ao uso futuro do protótipo como base para o sistema real, ou não, um protótipo pode ser:

- *Protótipo descartável*: é um protótipo exploratório e não se pretende utilizá-lo como uma parte real do sistema a ser entregue (PFLEEGER, 2004). O protótipo é construído apenas para apoiar o levantamento e a validação de requisitos, sendo descartado após essas fases. Protótipos descartáveis enfatizam o desenvolvimento

rápido, sem prestar atenção a princípios de engenharia e qualidade de software. Eles não devem ser mais elaborados do que o necessário para atingir seus objetivos. Assim, alguns atributos de qualidade, como robustez, confiabilidade e desempenho, podem não ser levados em conta. O uso de protótipos descartáveis é mais apropriado quando a equipe se depara com incertezas, ambiguidade, falta de completeza e imprecisão nos requisitos (WIEGERS, 2003).

- *Protótipo evolutivo*: é desenvolvido para se aprender mais sobre o problema e se ter a base de uma parte ou de todo o software a ser entregue (PFLEEGER, 2004). Em contraste com um protótipo descartável, o protótipo é parte (ou uma versão) do produto final e deve prover uma base para construir esse produto de forma incremental. Portanto, deve considerar princípios de engenharia e qualidade de software (WIEGERS, 2003).

Quanto ao conjunto de funcionalidades provido pelo protótipo, um protótipo pode ser:

- *Protótipo de características selecionadas*: apenas uma porção do sistema é implementada no protótipo.
- *Protótipo completo*: o protótipo apresenta todas as características do que se imagina ser o sistema real.

Essas diferentes classificações de protótipos são, em certa extensão, ortogonais e podem ser combinadas. Por exemplo, um primeiro protótipo que implementa apenas parte das interfaces de um sistema, mas que é usado como base para o desenvolvimento posterior pode ser classificado como um protótipo de interface, evolutivo e de características selecionadas. Já o que Kendall e Kendall (2010) chamam de um protótipo *"arranjado às pressas"* (i.e., um protótipo que possui toda a funcionalidade do sistema final, mas que não foi construído com o devido cuidado e que, portanto, sua qualidade e desempenho são deficientes) pode ser considerado um protótipo operacional e completo, podendo ser descartável ou evolutivo, dependendo se ele for ser usado, ou não, como base para o desenvolvimento do produto final. O que Kendall e Kendall (2010) chamam de um protótipo *"primeiro de uma série"* (i.e., um sistema piloto desenvolvido para ser avaliado antes de ser distribuído) pode ser considerado um protótipo operacional, completo e evolutivo. Contudo, nem todas as categorias podem ser combinadas entre si. Por exemplo, um protótipo não operacional é necessariamente um protótipo de características selecionadas, uma vez que ele certamente não implementa todas as características que se imagina ter o sistema real.

Diferentes tipos de protótipos podem ser combinados para apoiar um mesmo projeto de desenvolvimento. Por exemplo, no estágio de levantamento preliminar de requisitos, protótipos de interface podem ser usados para refinar requisitos. Esses requisitos são, então, gradativamente refinados e implementados em uma série de protótipos evolutivos, considerando inicialmente apenas os principais processos de negócio (características selecionadas) e algumas camadas do software (interface com o usuário e lógica de negócio). Outros elementos são incorporados ao produto final, por meio de incrementos e iterações sucessivas, até se obter o produto de software final.

A prototipagem também requer planejamento. Devem-se definir porque, quando e que tipo de protótipo usar, selecionar usuários para avaliar o protótipo e definir como o feedback do usuário será obtido.

Usuários são fundamentais na prototipagem. Para capturar as reações dos usuários em relação ao protótipo, outras técnicas de levantamento de informação devem ser usadas em

conjunto. Durante a experimentação do usuário com o protótipo, pode-se utilizar observação. Para capturar opiniões e sugestões, podem ser usados questionários e entrevistas (KENDALL; KENDALL, 2010).

Para o desenvolvimento do protótipo, as seguintes diretrizes podem ser úteis (KENDALL; KENDALL, 2010; WIEGERS, 2003):

- Defina o propósito do protótipo antes de começar a construí-lo.
- Trabalhe com módulos gerenciáveis. Para fins de prototipagem não é necessário, e muitas vezes, nem desejável, construir um sistema completo.
- Construa o protótipo rapidamente. A construção de um protótipo durante as fases de levantamento e análise de requisitos não pode consumir tempo em demasia, caso contrário perde sua finalidade. Para acelerar a construção, use ferramentas adequadas.
- Modifique o protótipo em iterações sucessivas. O protótipo deve ser alterado em direção às necessidades do usuário. Cada modificação requer uma nova avaliação.
- Enfatize a interface com o usuário. As interfaces do protótipo devem permitir que o usuário interaja facilmente com o sistema. Um mínimo de treinamento deve ser requerido. Sistemas interativos com interfaces gráficas são muito indicados à prototipagem.

A prototipagem pode trazer uma série de benefícios, dentre eles (KENDALL; KENDALL, 2010):

- Permite alterar o sistema mais cedo no desenvolvimento, adequando-o mais de perto às necessidades do usuário (menor custo de uma alteração).
- Permite descartar um sistema quando este se mostrar inadequado (análise de viabilidade).
- Possibilita desenvolver um sistema que atenda mais de perto as necessidades e expectativas dos usuários, na medida em que permite uma interação com o usuário ao longo de todo o ciclo de vida do desenvolvimento.

Contudo, a prototipagem também pode ser fonte de problemas, dentre eles (KENDALL; KENDALL, 2010; AURUM; WOHLIN, 2005; WIEGERS, 2003):

- *Gerência do projeto*: Normalmente, várias iterações são necessárias para se refinar um protótipo. Sob esta ótica, surge uma importante questão: Quando parar? Se essa questão não for tratada com cuidado, a prototipagem pode se estender indefinidamente. É importante, pois, delinear e seguir um plano para coletar, analisar e interpretar as informações de feedback do usuário. Além disso, em alguns casos, trabalhar com protótipos pode ser caro e demandar muito tempo.
- *Considerar o protótipo como sendo o sistema final*: o maior risco da prototipagem é que usuários, ao verem um protótipo rodando, concluam que o projeto está próximo de seu fim, achando que o protótipo é o sistema final. Analogamente, os desenvolvedores podem se sentir tentados a transformar protótipos descartáveis no sistema, não levando em conta que a qualidade pode não ter sido apropriadamente considerada. Assim, gerenciar expectativas é fundamental para um uso bem sucedido da prototipagem. Todos que veem o protótipo devem entender seu propósito e suas limitações.

3.2.6 – Outras Técnicas de Levantamento de Requisitos

Além das técnicas discutidas anteriormente, há várias outras igualmente úteis. Dentre elas, podem ser citadas:

- **Investigação ou Análise de Documentos**: em qualquer negócio, há vários documentos cuja interpretação pode ajudar no levantamento de informações, tais como relatórios usados na tomada de decisão, fichas e uma variedade de formulários. Documentos com formato pré-determinado, tais como relatórios, fichas e formulários, têm um propósito específico e um público-alvo e trazem informações muito úteis. Relatórios de desempenho, por exemplo, podem mostrar metas da organização, a distância em que a organização se encontra da meta e a tendência atual. Relatórios usados no processo de tomada de decisão mostram informações compiladas e podem incorporar algum conhecimento sobre a estratégia da organização. Formulários, assim como fichas, são muito úteis para o levantamento de requisitos de informação. Tais informações são difíceis de serem obtidas através de outras técnicas de levantamento de requisitos, como entrevistas e observação (KENDALL; KENDALL, 2010).
- **Cenários**: são descrições narrativas de processos correntes e futuros, incluindo ações e interações entre usuários e o sistema. O enredo do cenário se refere a uma porção do trabalho que está sendo estudada. O termo enredo é usado para designar que a porção de trabalho é dividida em um número de passos ou cenas. Explicando-se esses passos, explica-se o trabalho. Muitas vezes, cenários são usados para se chegar a um entendimento acerca de um caso de uso, mostrando, passo a passo, como um caso de uso é realizado. Uma vez que casos de uso capturam uma porção discreta de funcionalidade, é interessante definir cenários para contar a história de um caso de uso. As pessoas geralmente consideram mais fácil relatar exemplos de situações reais do que abstrair descrições e, daí vem a utilidade dos cenários. Um cenário pode começar com um esboço da interação e, durante o levantamento de requisitos, detalhes podem ser adicionados para criar uma descrição mais completa dessa interação. Neste sentido, cenários requerem uma abordagem incremental e interativa de desenvolvimento. Além disso, cenários tipicamente não consideram a estrutura interna do sistema (AURUM; WOHLIN, 2005; ROBERTSON; ROBERTSON, 2006; SOMMERVILLE, 2007).
- **Reúso de Requisitos**: é sempre uma boa prática de engenharia de software reutilizar tanto conhecimento quanto possível durante o desenvolvimento de um novo sistema. Isso não é diferente no caso de requisitos (KOTONYA; SOMMERVILLE, 1998). O reúso de requisitos é possível em diversas situações, dentre elas: (i) requisitos relacionados ao mesmo domínio de aplicação; (ii) requisitos relacionados à mesma tarefa; (iii) requisitos de sistemas considerados similares; e (iv) requisitos que refletem políticas organizacionais. Além disso, modelos podem ser reutilizados. Em especial, há vários catálogos de fragmentos de modelos que aparecem recorrentemente no desenvolvimento de sistemas para certo domínio ou tarefa, ditos padrões de análise. Conhecer e reutilizar esses padrões é uma ótima estratégia para aumentar a qualidade e a produtividade na análise de requisitos, conforme discutido no Capítulo 8 deste texto.

3.2.7 – Aplicando as Técnicas de Levantamento de Requisitos

Duas importantes questões que precisam ser abordadas em relação às técnicas de levantamento de requisitos são (AURUM; WOHLIN, 2005):

- Que técnica(s) aplicar durante uma atividade de levantamento de requisitos?
- Quais dessas técnicas são complementares?

Em última instância, cada situação é, em alguma extensão, única e, portanto, as respostas a essas perguntas são dependentes do contexto do projeto (AURUM; WOHLIN, 2005). De maneira geral, as principais técnicas discutidas neste texto são complementares.

Algumas informações são difíceis de serem obtidas através de entrevistas ou observação, tais como dados sobre um determinado objeto ou evento, informação financeira e contextos da organização. Tais informações revelam, tipicamente, um histórico da organização e sua direção. Nestes casos, a investigação de documentos é uma boa opção, pois fatos obtidos em uma investigação podem explicar o desempenho passado da organização. Por outro lado, metas projetam o futuro. Entrevistas são importantes para se determinar metas, obter necessidades e perspectivas individuais. A coleta colaborativa de requisitos pode ser uma boa alternativa ao uso de entrevistas, sobretudo para se ter um entendimento coletivo e para decidir sobre requisitos.

Questionários podem ser usados para quantificar o que foi levantado usando outras técnicas de levantamento e, portanto, um questionário pode ser definido com base no que foi levantado preliminarmente, por exemplo, em uma entrevista. Questionários também podem ser usados para examinar uma grande amostra de usuários do sistema para sentir problemas ou levantar questões importantes, antes de se programar entrevistas (KENDALL; KENDALL, 2010).

Observação pode ser usada para confirmar ou refutar informações levantadas em entrevistas, workshops de requisitos e/ou questionários, bem como para capturar informação complementar sobre os interessados, seus processos de negócio e o seu ambiente de trabalho. De maneira inversa, podem-se utilizar outras técnicas para completar informações obtidas em uma observação.

A observação pode ser combinada também com a prototipagem. Uma vez construído um protótipo, podem-se observar os usuários utilizando o protótipo, de modo a avaliar o mesmo e derivar novos requisitos. Para capturar as reações dos usuários em relação ao protótipo, outras técnicas de levantamento de requisitos também podem ser usadas, tais como entrevistas e questionários para capturar opiniões e sugestões. Por fim, cenários podem ser usados para derivar protótipos e protótipos podem ser apresentados no contexto da discussão de um cenário, sendo técnicas bastante complementares (KENDALL; KENDALL, 2010; SOMMERVILLE, 2007).

De fato, quase todas as técnicas são complementares em alguma extensão, podendo ser alternativas em outras. Assim, é importante avaliar cada caso e escolher o melhor conjunto de técnicas a serem aplicadas, levando em consideração, dentre outros, a experiência dos analistas no uso das diversas técnicas, o perfil dos interessados e o tempo disponível.

3.3 – Requisitos e Modelagem de Processos de Negócio

O uso de sistemas de informação tem por objetivo principal apoiar ações dos processos de negócio das organizações. Deve-se ter em mente que o principal interesse dos clientes não é o sistema de informação em si, mas sim os efeitos positivos gerados pela sua utilização. Dentre os principais benefícios da implantação de sistemas de informação para apoiar processos de negócio estão (DAVENPORT, 2000): (i) a automatização de tarefas antes realizadas manualmente, (ii) a racionalização dos dados, (iii) a implementação de melhorias nos processos da organização, (iv) ajuste das interfaces entre áreas, (v) aperfeiçoamento dos serviços aos clientes e (vi) geração de informações gerenciais.

Assim, durante o levantamento de requisitos, é necessário compreender o contexto organizacional, no qual o sistema será inserido, bem como se alinhar aos objetivos desse contexto. Se o entendimento do contexto organizacional não ocorre, os requisitos levantados tendem a ficar mais centrados em aspectos tecnológicos, sem levar em consideração fatores organizacionais, tais como (CARVALHO, 2009):

- Regras de negócio que têm impacto sobre o sistema;
- Usuários que utilizam as informações geradas pelo sistema;
- Impactos gerados pelo sistema na forma de execução dos processos de negócio da organização.

É importante notar que os sistemas de informação habilitam os processos de negócio e que, se os processos de negócio e os sistemas não estiverem alinhados, então os sistemas não atenderão às expectativas de seus usuários. Essa falta de alinhamento é apontada como sendo uma das principais causas do fracasso de projetos de desenvolvimento de sistemas de informação (FRYE; GULLEDDGE, 2007). Assim, levantar requisitos tomando por base processos de negócio é importante, pois sistemas de informação mantêm estreita relação com o ambiente organizacional. Um tratamento dissociado do aparato de levantamento de requisitos da visão de negócios pode levar à subutilização de potencialidades tecnológicas (CARVALHO, 2009).

Um modelo de negócio (*business model*) é, na verdade, um conjunto de modelos que provê uma visualização dos processos de negócio, de como estes são executados, quais são as suas metas, como cada processo trabalha para atingir essas metas, quais as unidades organizacionais e os papéis das pessoas envolvidos em cada atividade do processo, quais as localidades onde a organização está distribuída e quais eventos deflagram seus processos e atividades. Como ilustra a Figura 3.1, para modelar esses diferentes aspectos, diferentes tipos de modelos podem ser elaborados, dentre eles (KNIGHT, 2004):

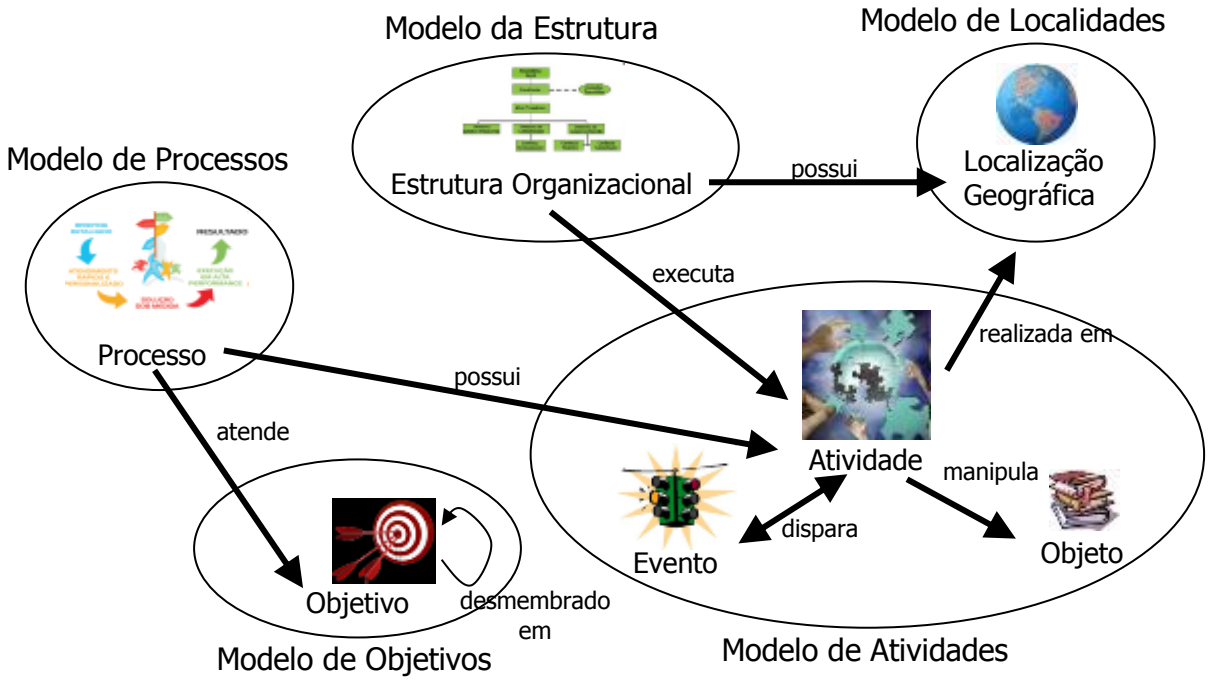


Figura 3.2 – Submodelos do Modelo de Negócio (adaptado de (KNIGHT, 2004)).

- Modelos da Estrutura Organizacional: representam as unidades organizacionais, seus papéis e seus relacionamentos;
- Modelos de Localizações Geográficas: representam as localidades onde a organização está distribuída e os relacionamentos entre localidades e unidades organizacionais;
- Modelos de objetivos: mostram os objetivos da organização e seus relacionamentos, o desdobramento dos objetivos em subobjetivos e o relacionamento entre os objetivos e os processos de negócio;
- Modelos de processos: representam os processos de negócio executados na organização (com suas atividades e relacionamentos) e seus desdobramentos em subprocessos e atividades;
- Modelos de atividades: mostram os relacionamentos entre as atividades dos processos de negócio, seus responsáveis, objetos de negócio e eventos que disparam ou são disparados com a execução das atividades.

Há muitas notações utilizadas para representar os vários tipos de modelos citados anteriormente, sendo que alguns autores propõem o uso de diagramas da UML ou extensões deles para essa finalidade. Em especial, os diagramas de atividades são bastante utilizados para a representação de modelos de processos de negócio e de atividades.

Um dos usos mais comuns de modelos de processo/atividades na Engenharia de Requisitos é a derivação de casos de uso a partir da análise das informações capturadas nos diagramas que representam os aspectos do negócio, incluindo a identificação das atividades dos processos que serão apoiadas pelo sistema, bem como a identificação dos atores dos casos de uso a partir dos executores (pessoas ou máquinas) relacionados aos processos.

No decorrer da modelagem de processos, a reunião dos envolvidos no processo permite que seja estabelecida uma visão abrangente de todo o contexto, possibilitando a identificação de divergências e abrindo espaços para a melhoria, o que pode levar à reengenharia dos

processos de negócio. Processos correntes (processos *AS IS*) podem dar origem a novos processos, mais eficazes, a serem implantados na organização (processos *TO BE*). O contexto no qual o sistema será inserido é considerado como parte integral na aplicação da abordagem orientada a modelos de processos, permitindo que o sistema seja considerado um agente de mudanças e não apenas a automatização das práticas correntes, sejam elas boas ou não.

A modelagem de processos complementa as práticas convencionais de Engenharia de Requisitos, auxiliando o cliente a adquirir maturidade acerca da complexidade do seu próprio negócio e revelando o grau de adequação dos requisitos levantados com os processos (e objetivos) da organização (CARDOSO; ALMEIDA; GUIZZARDI, 2008).

É importante realçar que a modelagem de processos de negócios independe do desenvolvimento de sistemas e pode ser conduzida independentemente para a construção de uma arquitetura organizacional de referência (*enterprise architecture*). Quando necessária a construção de um sistema que dê apoio a partes dos processos modelados nessa arquitetura, é necessário retornar ao cliente somente para levantar requisitos de sistema e não para levantar requisitos de negócio (CARDOSO; ALMEIDA; GUIZZARDI, 2008).

3.4 – Escrevendo e Documentando Requisitos de Cliente

Os resultados do levantamento de requisitos têm de ser registrados em um documento, de modo que possam ser verificados, validados e utilizados como base para outras atividades do processo de software. Para que sejam úteis, os requisitos têm de ser escritos em um formato compreensível por todos os interessados. Além disso, esses interessados devem interpretá-los uniformemente.

Normalmente, requisitos são documentados usando alguma combinação de linguagem natural, modelos, tabelas e outros elementos. A linguagem natural é quase sempre imprescindível, uma vez que é a forma básica de comunicação compreensível por todos os interessados. Contudo, ela geralmente abre espaços para ambiguidades e má interpretação. Assim, é interessante procurar estruturar o uso da linguagem natural e complementar a descrição dos requisitos com outros elementos.

Conforme discutido no Capítulo 2, diferentes abordagens podem ser usadas para documentar requisitos. Neste texto, sugerimos elaborar dois documentos: o Documento de Definição de Requisitos e o Documento de Especificação de Requisitos. O Documento de Definição de Requisitos é mais sucinto, escrito em um nível mais apropriado para o cliente e contempla apenas os requisitos de cliente. O Documento de Especificação de Requisitos é mais detalhado, escrito a partir da perspectiva dos desenvolvedores (PFLEEGER, 2004), normalmente contendo diversos modelos para descrever requisitos de sistema.

Os requisitos de cliente devem ser descritos de modo a serem compreensíveis pelos interessados no sistema que não possuem conhecimento técnico detalhado. Eles devem versar somente sobre o comportamento externo do sistema, em uma linguagem simples, direta e sem usar terminologia específica de software (SOMMERVILLE, 2007).

O Documento de Definição de Requisitos tem como propósito descrever os requisitos de cliente, tendo como público-alvo clientes, usuários, gerentes (de cliente e de fornecedor) e desenvolvedores. Há muitos formatos distintos propostos na literatura para documentos de requisitos. Neste texto, é proposta uma estrutura bastante simples para esse tipo de documento, contendo apenas quatro seções:

- Introdução: breve introdução ao documento, descrevendo seu propósito e estrutura.
- Descrição do Propósito do Sistema: descreve o propósito geral do sistema.
- Descrição do Minimundo: apresenta, em um texto corrido, uma visão geral do domínio, do problema a ser resolvido e dos processos de negócio apoiados, bem como as principais ideias do cliente sobre o sistema a ser desenvolvido.
- Requisitos de cliente: apresenta os requisitos de cliente em linguagem natural.

As três primeiras seções não têm nenhuma estrutura especial, sendo apresentadas na forma de um texto corrido. A introdução deve ser breve e basicamente descrever o propósito e a estrutura do documento, podendo seguir um padrão preestabelecido pela organização. A descrição do propósito do sistema deve ser direta e objetiva, tipicamente em um único parágrafo. Já a descrição do minimundo é um pouco maior, algo entre uma e duas páginas, descrevendo aspectos gerais e relevantes para um primeiro entendimento do domínio, do problema a ser resolvido e dos processos de negócio apoiados. Contém as principais ideias do cliente sobre o sistema a ser desenvolvido, obtidas no levantamento preliminar e exploratório do sistema. Não se devem incluir detalhes.

A Seção 4, por sua vez, não deve ter um formato livre. Ao contrário, deve seguir um formato estabelecido pela organização, contendo, dentre outros: identificador do requisito, descrição, tipo, origem, prioridade, responsável, interessados, dependências em relação a outros requisitos e requisitos conflitantes. A definição de padrões organizacionais para a definição de requisitos é essencial para garantir uniformidade e evitar omissão de informações importantes acerca dos requisitos (SOMMERVILLE, 2007; WIEGERS, 2003). Como consequência, o padrão pode ser usado como um guia para a verificação de requisitos. A Tabela 3.4 apresenta um exemplo de padrão tabular. Sugerem-se agrupar requisitos de um mesmo tipo em uma mesma tabela. Assim, a informação do tipo do requisito não aparece explicitamente no padrão proposto. Além disso, informações complementares podem ser adicionadas em função do tipo de requisito. A seguir, discute-se como cada um dos itens da tabela pode ser tratado, segundo uma perspectiva geral. Na sequência, são tecidas considerações mais específicas sobre a descrição dos diferentes tipos de requisitos.

Tabela 3.4 – Tabela de Requisitos.

Identificador	Descrição	Origem	Prioridade	Responsável	Interessados	Dependências	Conflitos

Os requisitos devem possuir identificadores únicos para permitir a identificação e o rastreamento na gerência de requisitos. Há diversas propostas de esquemas de rotulagem de requisitos. Neste texto, recomenda-se usar um esquema de numeração sequencial por tipo de requisito, sendo usados os seguintes prefixos para designar os diferentes tipos de requisitos: RF – requisitos funcionais; e RNF – requisitos não funcionais. Para outros esquemas de rotulagem, vide (WIEGERS, 2003). É importante destacar que, quando um requisito é eliminado, seu identificador não pode ser atribuído a outro requisito.

A descrição do requisito normalmente é feita na forma de uma sentença em linguagem natural. Ainda que expressa em linguagem natural, é importante adotar um estilo consistente e usar a terminologia do usuário ao invés do jargão típico da computação. Em relação ao estilo, recomenda-se utilizar sentenças em um dos seguintes formatos para descrever requisitos funcionais e não funcionais:

- *O sistema deve <verbo indicando ação, seguido de complemento>*: use o verbo *dever* para designar uma função ou característica requerida para o sistema, ou seja, para descrever um requisito obrigatório. Exemplos: O sistema deve efetuar o controle dos clientes da empresa. O sistema deve processar um pedido do cliente em um tempo inferior a cinco segundos, contado a partir da entrada de dados.
- *O sistema pode <verbo indicando ação, seguido de complemento>*: use o verbo *poder* para designar uma função ou característica desejável para o sistema, ou seja, para descrever um requisito desejável, mas não essencial. Exemplos: O sistema pode notificar usuários em débito. O sistema pode sugerir outros produtos para compra, com base em produtos colocados no carrinho de compras do usuário.

Em algumas situações, pode-se querer explicitar que alguma funcionalidade ou característica não deve ser tratada pelo sistema (dito requisito negativo). Isso pode ser feito indicando-se uma sentença com a seguinte estrutura: *O sistema não deve <verbo indicando ação, seguido de complemento>*. Também é possível registrar que alguma tarefa específica relacionada ao sistema não fará parte do escopo do projeto, tal como a carga de dados de um sistema existente. Nestes casos, pode ser útil ter uma tabela de requisitos negativos.

Wiegers (2003) recomenda diversas diretrizes para a redação de requisitos, dentre elas:

- Escreva frases completas, com a gramática, ortografia e pontuação correta. Procure manter frases e parágrafos curtos e diretos.
- Use os termos consistentemente. Defina-os em um glossário.
- Prefira a voz ativa (o sistema deve fazer alguma coisa) à voz passiva (alguma coisa deve ser feita).
- Sempre que possível, identifique o tipo de usuário. Evite descrições genéricas como "o usuário deve [...]". Se o usuário no caso for, por exemplo, o caixa do banco, indique claramente "o caixa do banco deve [...]".
- Evite termos vagos, que conduzam a requisitos ambíguos e não testáveis, tais como "rápido", "adequado", "fácil de usar" etc.
- Escreva requisitos em um nível consistente de detalhe. Evite requisitos desnecessariamente pequenos, tais como requisitos individuais para tratar de ações de inclusão, remoção, alteração e consulta de um elemento de informação (p.ex., "O sistema deve permitir a alteração de dados de clientes"). Em casos como este, considere agrupar os requisitos menores em um requisito maior (tal como "O sistema deve permitir o controle de clientes"). Por outro lado, evite longos parágrafos narrativos que contenham múltiplos requisitos. Divida um requisito desta natureza em vários.
- O nível de descrição de um requisito deve ser tal que, se o requisito é satisfeito, a necessidade do cliente é atendida. Evite restringir desnecessariamente o projeto (design).
- Escreva requisitos individualmente testáveis. Um requisito bem escrito deve permitir a definição de um pequeno conjunto de testes para verificar se o requisito foi corretamente implementado.

Conforme apontado anteriormente, requisitos devem ser testáveis. Robertson e Robertson (2006) sugerem três maneiras de tornar os requisitos testáveis:

1. Especificar uma descrição quantitativa de cada advérbio ou adjetivo, de modo que o significado dos qualificadores fique claro e não ambíguo.
2. Trocar os pronomes pelos nomes das entidades.
3. Garantir que todo termo importante seja definido em um glossário no documento de requisitos.

A origem de um requisito deve apontar a partir de que entidade (pessoa, documento, atividade) o requisito foi identificado. Um requisito identificado durante uma investigação de documentos, p.ex., tem como origem o(s) documento(s) inspecionado(s). Já um requisito levantado em uma entrevista com certo usuário tem como origem o próprio usuário. A informação de origem é importante para se conseguir rastrear requisitos para a sua origem, prática muito recomendada na Gerência de Requisitos.

Requisitos podem ter importância relativa diferente em relação a outros requisitos. Assim, é importante que o cliente e outros interessados estabeleçam conjuntamente a prioridade de cada requisito.

É muito importante saber quem é o analista responsável por um requisito, bem como quem são os interessados (clientes, usuários etc.) naquele requisito. São eles que estarão envolvidos nas discussões relativas ao requisito, incluindo a tentativa de acabar com conflitos e a definição de prioridades. Assim, deve-se registrar o nome e o papel do responsável e dos interessados em cada requisito.

Um requisito pode depender de outros ou conflitar com outros. Quando as dependências e conflitos forem detectados, devem-se listar os respectivos identificadores nas colunas de dependências e conflitos.

3.4.1 - Escrevendo Requisitos Funcionais

As diretrizes apresentadas anteriormente aplicam-se integralmente a requisitos funcionais. Assim, não há outras diretrizes específicas para os requisitos funcionais. Deve-se realçar apenas que, quando expressos como requisitos de cliente, requisitos funcionais são geralmente descritos de forma abstrata, não cabendo neste momento entrar em detalhes. Detalhes vão ser naturalmente adicionados quando esses mesmos requisitos forem descritos na forma de requisitos de sistema.

Uma alternativa largamente empregada para especificar requisitos funcionais no nível de requisitos de sistema é a modelagem. Uma das técnicas mais comumente utilizadas para descrever requisitos funcionais como requisitos de sistema é a modelagem de casos de uso.

Vale ressaltar que os processos de negócio a serem apoiados pelo sistema tipicamente dão origem a requisitos funcionais. Assim, a partir de uma descrição de minimundo ou em um relatório proveniente de alguma atividade de levantamento de requisitos, podem ser encontrados requisitos funcionais a partir da identificação dos processos a serem apoiados. Além disso, o controle de informações que o negócio precisa gerenciar para apoiar os processos de negócio também deve dar origem a requisitos funcionais representando atividades custodiais (cadastros).

3.4.2 - Escrevendo Requisitos Não Funcionais

Clientes e usuários naturalmente enfocam a especificação de requisitos funcionais. Entretanto para um sistema ser bem-sucedido, é necessário mais do que entregar a funcionalidade correta. Usuários também têm expectativas sobre quão bem o sistema vai funcionar. Características que entram nessas expectativas incluem: quão fácil é usar o sistema, quão rapidamente ele roda, com que frequência ele falha e como ele trata condições inesperadas. Essas características, conhecidas como atributos de qualidade do produto de software, são parte dos requisitos não funcionais do sistema. Essas expectativas de qualidade do produto têm de ser exploradas durante o levantamento de requisitos (WIEGERS, 2003).

Clientes geralmente não apontam suas expectativas de qualidade explicitamente. Contudo, informações providas por eles durante o levantamento de requisitos fornecem algumas pistas sobre o que eles têm em mente. Assim, é necessário definir o que os usuários pensam quando eles dizem que o sistema deve ser amigável, rápido, confiável ou robusto (WIEGERS, 2003).

Há muitos atributos de qualidade que podem ser importantes para um sistema. Uma boa estratégia para levantar requisitos não funcionais de produto consiste em explorar uma lista de potenciais atributos de qualidade que a grande maioria dos sistemas deve apresentar em algum nível. Por exemplo, o modelo de qualidade externa e interna de produtos de software definido na norma ISO/IEC 25010, utilizado como referência para a avaliação de produtos de software, define oito características de qualidade, desdobradas em subcaracterísticas, a saber (ISO/IEC, 2011):

- Aptidão Funcional (*Functional Suitability*): grau em que o produto provê funções que satisfazem às necessidades explícitas e implícitas, quando usado em condições especificadas. Inclui subcaracterísticas que evidenciam a existência de um conjunto de funções e suas propriedades específicas, a saber:
  - Completude Funcional: capacidade do produto de software de prover um conjunto apropriado de funções para tarefas e objetivos do usuário especificados. Refere-se às necessidades declaradas. Um produto no qual falte alguma função requerida não apresenta este atributo de qualidade.
  - Correção Funcional (ou Acurácia): grau em que o produto de software fornece resultados corretos e precisos, conforme acordado. Um produto que apresente dados incorretos, ou com a precisão abaixo dos limites definidos como toleráveis, não apresenta este atributo de qualidade.
  - Adequação Funcional (*Functional Appropriateness*): capacidade do produto de software de facilitar a realização das tarefas e objetivos do usuário. Refere-se às necessidades implícitas.
- Confiabilidade: grau em que o produto executa as funções especificadas com um comportamento consistente com o esperado, por um período de tempo. A confiabilidade está relacionada com os defeitos que um produto apresenta e como este produto se comporta em situações consideradas fora do normal. Suas subcaracterísticas são:
  - Maturidade: é uma medida da frequência com que o produto de software apresenta defeitos ao longo de um período estabelecido de tempo. Refere-se, portanto, à capacidade do produto de software de evitar falhas decorrentes de defeitos no software, mantendo sua operação normal ao longo do tempo.

  - Disponibilidade: capacidade do produto de software de estar operacional e acessível quando seu uso for requerido.
  - Tolerância a falhas: capacidade do produto de software de operar em um nível de desempenho especificado em casos de defeitos no software ou no hardware. Esta subcaracterística tem a ver com a forma como o software reage quando ocorre uma situação externa fora do normal (p.ex., conexão com a Internet interrompida).
  - Recuperabilidade: capacidade do produto de software de se colocar novamente em operação, restabelecendo seu nível de desempenho especificado, e recuperar os dados diretamente afetados no caso de uma falha.

- Usabilidade: grau em que o produto apresenta atributos que permitem que o mesmo seja entendido, aprendido e usado, e que o tornem atrativo para o usuário. Tem como subcaracterísticas:
  - Reconhecimento da Adequação: grau em que os usuários reconhecem que o produto de software é adequado para suas necessidades.
  - Capacidade de Aprendizado: refere-se à facilidade de o usuário entender os conceitos chave do produto e aprender a utilizá-lo, tornando-se competente em seu uso.
  - Operabilidade: refere-se à facilidade do usuário operar e controlar o produto de software.
  - Proteção contra Erros do Usuário: capacidade do produto de software de evitar que o usuário cometa erros.
  - Estética da Interface com o Usuário: capacidade do produto de software de ser atraente ao usuário, lhe oferecendo uma interface com interação agradável.
  - Acessibilidade: capacidade do produto de software ser utilizado por um amplo espectro de pessoas, incluindo portadores de necessidades especiais e com limitações associadas à idade.
- Eficiência de Desempenho: capacidade de o produto manter um nível de desempenho apropriado em relação aos recursos utilizados em condições estabelecidas. Inclui subcaracterísticas que evidenciam o relacionamento entre o nível de desempenho do software e a quantidade de recursos utilizados, a saber:
  - Comportamento em Relação ao Tempo: capacidade do produto de software de fornecer tempos de resposta e de processamento apropriados, quando o software executa suas funções, sob condições estabelecidas.
  - Utilização de Recursos: capacidade do produto de software de usar tipos e quantidades apropriados de recursos, quando executa suas funções, sob condições estabelecidas.
  - Capacidade: refere-se ao grau em que os limites máximos dos parâmetros do produto de software (p.ex., itens que podem ser armazenados, número de usuários concorrentes etc.) atendem às condições especificadas.

- Segurança: grau em que informações e dados são protegidos contra acesso por pessoas ou sistemas não autorizados, bem como grau em que essas informações e dados são disponibilizados para as pessoas ou sistemas com acesso autorizado. Tem como subcaracterísticas:
  - Confidencialidade: grau em que o produto ou sistema garante que os dados estão acessíveis apenas para aqueles autorizados a acessá-los.
  - Integridade: grau em que o sistema, produto ou componente evita acesso não autorizado a, ou modificação de, programas ou dados.
  - Não Repúdio: grau em que se pode provar que ações ou eventos aconteceram, de modo que eles não possam ser repudiados posteriormente.
  - Responsabilização: grau em que as ações de uma entidade (pessoa ou sistema) podem ser rastreadas unicamente a essa entidade.
  - Autenticidade: grau em que se pode provar que a identidade de um sujeito ou recurso é aquela reivindicada.
- Compatibilidade: capacidade do produto de software de trocar informações com outras aplicações e/ou compartilhar o mesmo ambiente de hardware ou software. Suas subcaracterísticas são:
  - Coexistência: grau em que um produto pode desempenhar eficientemente suas funções requeridas, ao mesmo tempo em que compartilha um ambiente e recursos comuns com outros produtos, sem prejuízo para qualquer outro.
  - Interoperabilidade: capacidade do produto de software de interagir com outros sistemas especificados, trocando e usando as informações trocadas.
- Manutenibilidade: capacidade do produto de software de ser modificado. Tem como subcaracterísticas:
  - Modularidade: grau em que um sistema ou programa é composto por componentes discretos (coesos e fracamente acoplados), de modo que mudanças em um componente tenham impacto mínimo sobre os outros.
  - Reusabilidade: capacidade dos componentes do produto de software serem utilizados na construção de outros componentes ou sistemas.
  - Analisabilidade: grau de eficácia e eficiência com que é possível: avaliar o impacto de uma alteração pretendida no produto ou sistema; diagnosticar deficiências ou causas de falhas no produto, ou identificar partes a serem modificadas.
  - Modificabilidade: grau em que um produto ou sistema pode ser eficaz e eficientemente modificado sem introduzir defeitos ou degradar sua qualidade.
  - Testabilidade: grau de eficácia e eficiência com que critérios de teste podem ser estabelecidos para um sistema, produto ou componente, e testes podem ser realizados para determinar se esses critérios foram satisfeitos.

- Portabilidade: refere-se à capacidade do software ser transferido de um ambiente de hardware, software ou operacional para outro. Tem como subcaracterísticas:
    - Adaptabilidade: grau em que um produto ou sistema pode ser eficaz e eficientemente adaptado para ambientes diferentes, ou em evolução, de hardware, software ou operacional.
    - Capacidade para ser instalado (instalabilidade): grau de eficácia e eficiência com que um produto ou sistema pode ser instalado e/ou desinstalado com sucesso em um ambiente especificado.
    - Capacidade de substituição (substituibilidade): grau em que um produto pode substituir outro produto especificado para a mesma finalidade, no mesmo ambiente. Considera, também, a facilidade de atualização para novas versões.

A característica de aptidão funcional e suas subcaracterísticas estão claramente relacionadas a requisitos funcionais. As demais, contudo, podem ser vistas como requisitos não funcionais que quaisquer produtos de software terão de tratar em alguma extensão.

Outro ponto importante a destacar é que diferentes autores listam diferentes características de qualidade, usando classificações próprias. Por exemplo, Bass, Clements e Kazman (2003) consideram, dentre outros, os seguintes atributos de qualidade:

- Disponibilidade: refere-se a falhas do sistema e suas consequências associadas. Uma falha ocorre quando o sistema não entrega mais um serviço consistente com sua especificação.
- Modificabilidade: diz respeito ao custo de modificação do sistema.
- Desempenho: refere-se a tempo.
- Segurança: está relacionada à habilidade do sistema impedir o uso não autorizado, enquanto ainda provê seus serviços para os usuários legítimos.
- Testabilidade: refere-se ao quão fácil é testar o software.
- Usabilidade: diz respeito a quão fácil é para o usuário realizar uma tarefa e o tipo de suporte ao usuário que o sistema provê.

Além das características de qualidade que se aplicam diretamente ao sistema, ditas características de qualidade de produto, Bass, Clements e Kazman (2003) listam outras características relacionadas a metas de negócio, dentre elas: tempo para chegar ao mercado (*time to market*), custo-benefício, tempo de vida projetado para o sistema, mercado alvo, cronograma de implementação e integração com sistemas legados.

Wiegers (2003), por sua vez, agrupa atributos de qualidade do produto em duas categorias principais: atributos importantes para os usuários e atributos importantes para os desenvolvedores. Como atributos importantes para os usuários são apontados os seguintes: disponibilidade, eficiência, flexibilidade, integridade, interoperabilidade, confiabilidade, robustez e usabilidade. Como atributos importantes para os desenvolvedores, são enumerados os seguintes: manutenibilidade, portabilidade, reusabilidade e testabilidade.

Uma vez que não há um consenso sobre quais atributos de qualidade considerar, cada organização deve definir as categorias de requisitos não funcionais a serem consideradas em seus projetos de software. Além disso, essa informação deve ser adicionada à tabela de requisitos não funcionais e, portanto, a Tabela 3.4, quando usada para descrever requisitos não funcionais, deve ter uma coluna adicional para indicar a categoria do requisito não funcional.

Em um mundo ideal, todo sistema deveria exibir os valores máximos para todos os atributos de qualidade. Contudo, como no mundo real isso não é possível, é fundamental definir quais atributos são mais importantes para o sucesso do projeto. Uma abordagem para tratar essa questão consiste em pedir para que diferentes representantes de usuários classifiquem cada atributo em uma escala de 1 (sem importância) a 5 (muito importante). As respostas ajudam o analista a determinar quais atributos são mais importantes. Obviamente, conflitos podem surgir e precisam ser resolvidos (WIEGERS, 2003).

Um aspecto a considerar é que diferentes partes do produto podem requerer diferentes combinações de atributos de qualidade. Assim, é importante também diferenciar características que se aplicam ao produto por inteiro daquelas que são necessárias para certas partes do sistema (WIEGERS, 2003). Para capturar essa diferenciação, é importante introduzir mais uma coluna na tabela de requisitos não funcionais, escopo, podendo assumir dois valores: funcionalidade, quando a característica se aplica apenas a algumas funcionalidades; e sistema, quando a característica se aplica ao sistema como um todo. Quando o escopo de um RNF for de funcionalidade, os requisitos funcionais que precisam incorporar essa característica de qualidade deverão incluir em sua coluna de dependências o identificador desse RNF.

Uma vez priorizados os atributos de qualidade, o analista deve passar, então, a trabalhar com os usuários no sentido de especificar, para cada atributo considerado importante, requisitos mensuráveis e, por conseguinte, testáveis. Se os atributos de qualidade são especificados de maneira não passível de verificação, não é possível dizer posteriormente se eles foram atingidos ou não. Quando apropriado, devem-se indicar escalas ou unidades de medida para cada atributo e os valores mínimo, alvo e máximo. Se não for possível quantificar todos os atributos importantes, deve-se, pelo menos, definir suas prioridades. Pode-se, ainda, perguntar aos usuários o que constituiria um valor inaceitável para cada atributo e definir testes que tentem forçar o sistema a demonstrar tais características (WIEGERS, 2003). É importante destacar, contudo, que essa especificação mais detalhada dos RNFs não deve ser feita no levantamento inicial de requisitos. Ao contrário, ela é considerada como parte da análise de requisitos (corresponde à especificação dos RNFs) e muitas vezes, é até postergada para a fase de projeto (ou transição para a fase de projeto), uma vez que RNFs guardam estreita relação com aspectos tecnológicos, os quais serão tratados na fase de projeto.

Robertson e Robertson (2006) sugerem definir critérios de adequação ou ajuste (*fit criteria*) para permitir quantificar requisitos (tanto funcionais como não funcionais) e associá-los à descrição dos requisitos. À primeira vista, alguns requisitos não funcionais podem parecer difíceis de quantificar. Entretanto, deve ser possível atribuir números a eles. Se não se consegue quantificar e medir um requisito, então é provável que o requisito não seja de fato um requisito. Ele pode ser, por exemplo, vários requisitos descritos em um só.

Seja a seguinte situação. No desenvolvimento de um sistema para uma biblioteca, o usuário coloca o seguinte requisito não funcional: "O sistema deve ser amigável ao usuário". Esse requisito é vago, ambíguo e não passível de ser expresso em números. Como quantificar se o sistema é "amigável ao usuário"? Primeiro, é preciso entender o que o usuário quer dizer com "amigável ao usuário". Significa ser fácil de compreender? Fácil de aprender? Fácil de operar? Atrativo? Clareando a intenção do usuário, é possível sugerir uma escala de medição. Suponha que o usuário diga que ser "amigável ao usuário" significa que os usuários serão capazes de aprender rapidamente a usar o sistema. Uma vez definido que se está falando sobre facilidade de aprender (inteligibilidade), é possível definir como escala de medição o tempo gasto para dominar a execução de certas tarefas. A partir disso, pode-se estabelecer como

critério de aceitação o seguinte: "Novos bibliotecários devem ser capazes de efetuar empréstimos após a quarta tentativa de realizar essa tarefa usando o sistema".

Determinar critérios de aceitação ajuda a clarear um requisito. Ao se estabelecer uma escala de medição e os valores aceitáveis, o requisito é transformado de uma intenção vaga, e até certo ponto ambígua, em um requisito mensurável e bem formado. Estabelecida uma escala, pode-se perguntar ao usuário o que ele considera uma falha em atender ao requisito, de modo a definir o critério de aceitação. Contudo, pode ser difícil, senão impossível, obter um requisito completo e mensurável em primeira instância (ROBERTSON; ROBERTSON, 2006). Assim, na descrição de requisitos de cliente é suficiente capturar a intenção e depois, na especificação de requisitos de sistema, transformar essa intenção em um requisito mensurável, adicionando a ele um critério de aceitação. É muito comum que, neste processo, um requisito não funcional de usuário dê origem a vários requisitos não funcionais de sistema.

3.5 - Regras de Negócio

Toda organização opera de acordo com um extenso conjunto de políticas corporativas, leis, padrões industriais e regulamentações governamentais. Tais princípios de controle são coletivamente designados por regras de negócio. Uma regra de negócio é uma declaração que define ou restringe algum aspecto do negócio, com o propósito de estabelecer sua estrutura ou controlar ou influenciar o comportamento do negócio (WIEGERS, 2003).

Sistemas de informação tipicamente precisam fazer cumprir as regras de negócio. Ao contrário dos requisitos funcionais e não funcionais, a maioria das regras de negócio origina-se fora do contexto de um sistema específico. Assim, as regras a serem tratadas pelo sistema precisam ser identificadas, documentadas e associadas aos requisitos do sistema em questão (WIEGERS, 2003).

Wiegers identifica cinco tipos principais de regras de negócio, cada um deles apresentando uma forma típica de ser escrito:

- Fatos ou invariantes: declarações que são verdade sobre o negócio. Geralmente descrevem associações ou relacionamentos entre importantes termos do negócio. Ex.: Todo pedido tem uma taxa de remessa.
- Restrições: como o próprio nome indica, restringem as ações que o sistema ou seus usuários podem realizar. Algumas palavras ou frases sugerem a descrição de uma restrição, tais como *deve*, *não deve*, *não pode* e *somente*. Ex.: Um aluno só pode tomar emprestado, concomitantemente, até três livros.
- Ativadores de Ações: são regras que disparam alguma ação sob condições específicas. Uma declaração na forma "Se <alguma condição é verdadeira ou algum evento ocorre>, então <algo acontece>" é indicada para descrever ativadores de ações. Ex.: Se a data para retirada do livro é ultrapassada e o livro não é retirado, então a reserva é cancelada. Quando as condições que levam às ações são uma complexa combinação de múltiplas condições individuais, então o uso de tabelas de decisão ou árvores de decisão é indicado. As figuras 3.4 e 3.5 ilustram uma mesma regra de ativação de ações descrita por meio de uma árvore de decisão e de uma tabela de decisão, respectivamente.

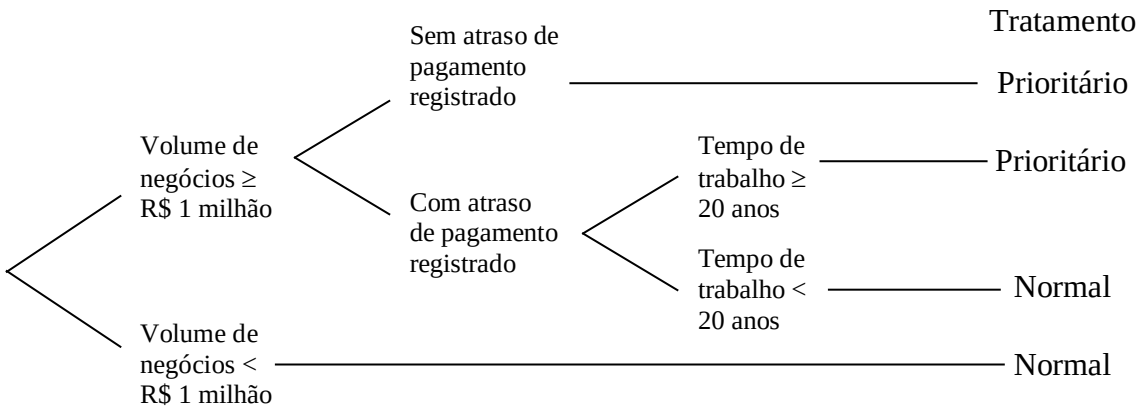


Figura 3.4 – Exemplo de Árvore de Decisão.

Tratamento de Clientes				
Volume de Negócios ≥ R$ 1 milhão?	S	S	S	N
Atraso de pagamento registrado?	N	S	S	-
Tempo de trabalho ≥ 20 anos?	-	S	N	-
Tratamento Prioritário	X	X		
Tratamento Normal			X	X

Figura 3.5 – Exemplo de Tabela de Decisão.

- Inferências: são regras que derivam novos fatos a partir de outros fatos ou cálculos. São normalmente escritas no padrão "se / então", como as regras ativadoras de ação, mas a cláusula então implica um fato ou nova informação e não uma ação a ser tomada. Ex.: Se o usuário não devolve um livro dentro do prazo estabelecido, então ele torna-se um usuário inadimplente.
- Computações: são regras de negócio que definem cálculos a serem realizados usando fórmulas matemáticas ou algoritmos específicos. Podem ser expressas como fórmulas matemáticas, descrição textual, tabelas etc. Ex.: Multa = Valor de Locação * Número de Dias de Atraso.

Ao contrário de requisitos funcionais e não funcionais, regras de negócio não são passíveis de serem capturadas por meio de perguntas simples e diretas, tal como "Quais são suas regras de negócio?" Regras de negócio emergem durante a discussão de requisitos, sobretudo quando se procura entender a base lógica por detrás de requisitos e restrições apontados pelos interessados (WIEGERS, 2003). Assim, não se deve pensar que será possível levantar muitas regras de negócio em um levantamento preliminar de requisitos. Pelo contrário, as regras de negócio vão surgir principalmente durante o levantamento detalhado dos requisitos. Wiegers (2003) aponta diversas potenciais origens para regras de negócio e sugere tipos de questões que o analista pode fazer para tentar capturar regras advindas dessas origens:

- Políticas: Por que é necessário fazer isso desse jeito?
- Regulamentações: O que o governo requer?
- Fórmulas: Como este valor é calculado?
- Modelos de Dados: Como essas entidades de dados estão relacionadas?
- Ciclo de Vida de Objetos: O que causa uma mudança no estado desse objeto?
- Decisões de Atores: O que o usuário pode fazer a seguir?
- Decisões de Sistema: Como o sistema sabe o que fazer a seguir?
- Eventos: O que pode (e não pode) acontecer?

Regras de negócio normalmente têm estreita relação com requisitos funcionais. Uma regra de negócio pode ser tratada no contexto de certa funcionalidade. Assim, a regra de negócio deve ser associada ao requisito funcional. Há casos em que uma regra de negócio conduz a um requisito funcional para fazer cumprir a regra. Neste caso, a regra de negócio é considerada a origem do requisito funcional (WIEGERS, 2003).

É importante destacar a importância das regras de restrição obtidas a partir de modelos conceituais estruturais, ditas *restrições de integridade*. Elas complementam as informações de um modelo deste tipo e capturam restrições relativas a relacionamentos entre elementos de um modelo que normalmente não são passíveis de serem capturadas pelas notações gráficas utilizadas na elaboração de modelos conceituais estruturais. Tais regras devem ser documentadas junto ao modelo conceitual estrutural do sistema. Seja o exemplo do modelo conceitual estrutural da Figura 3.6. Esse fragmento de modelo indica que: (i) um aluno cursa um curso; (ii) um aluno pode se matricular em nenhuma ou várias turmas; (iii) um curso possui um conjunto de disciplinas em sua matriz curricular; (iv) uma turma é de uma disciplina específica. Contudo, nada diz sobre restrições entre o estabelecimento dessas várias relações. Suponha que o negócio indique que a seguinte restrição deve ser considerada: Um aluno só pode ser matricular em turmas de disciplinas que compõe a grade curricular do curso que esse aluno cursa. Essa restrição tem de ser escrita para complementar o modelo. Sugere-se utilizar "RI" como o prefixo para o identificador único de restrições de integridade.

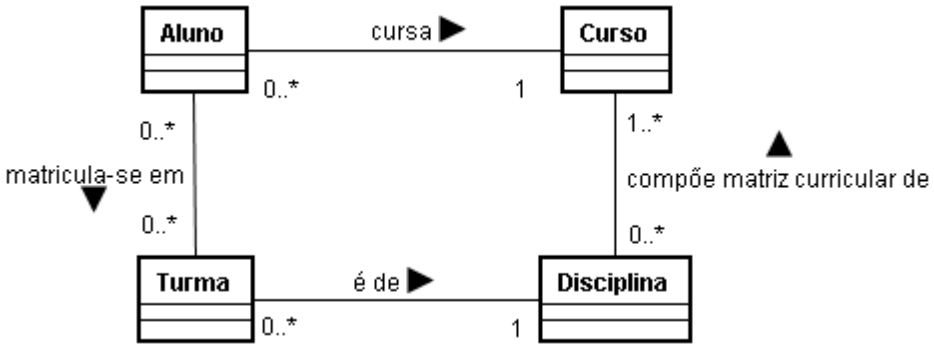


Figura 3.6 – Exemplo de Fragmento de Modelo de Dados com Restrição de Integridade.

Outro tipo de restrição importante são as regras que impõem restrições sobre funcionalidades de inserção, atualização ou exclusão de dados, devido a relacionamentos existentes entre as entidades. Voltando ao exemplo da Figura 3.6, a exclusão de disciplinas não deve ser livre, uma vez que turmas são existencialmente dependentes de disciplinas. Assim, a

seguinte regra deve ser considerada: Ao excluir uma disciplina, devem-se excluir todas as turmas a ela relacionadas. Essas regras são denominadas neste texto como *restrições de processamento* e, uma vez que dizem respeito a funcionalidades, devem ser documentadas junto com a descrição do caso de uso que detalha a respectiva funcionalidade. Neste caso, não há necessidade de se identificar unicamente a restrição de processamento usando identificadores, uma vez que ela será documentada diretamente com o caso de uso relacionado.

3.6 - Suposições

O comportamento do sistema ou os resultados esperados de sua utilização podem ser pautados em suposições. Uma suposição descreve um estado de coisas no ambiente de interesse do sistema que os interessados acreditam ser verdadeiro. Em outras palavras, uma suposição é uma crença de que uma situação específica acontece no ambiente de interesse. Algumas vezes, representar tais situações é útil, pois elas precisam ser consideradas na solução de um problema específico. Mais do que isso, as suposições podem ser imprescindíveis para dizer se um sistema atinge ou não seus objetivos (NEGRI et al., 2017). P.ex., em um sistema de biblioteca, quando um usuário tem uma reserva, ele deve ser notificado quando um exemplar do livro reservado estiver disponível para atender sua reserva. Visando satisfazer o objetivo de comunicar ao usuário a possibilidade de atendimento de sua reserva, uma funcionalidade (requisito funcional) é desenvolvida para enviar um e-mail para o usuário, avisando-o da disponibilidade. Essa funcionalidade sozinha, contudo, não garante que o objetivo de "comunicar o usuário" é atingido. Para que esse objetivo seja, de fato, atingido, há suposições a considerar, tais como o e-mail registrado no sistema está atualizado, a mensagem será entregue e o usuário vai acessar e ler o e-mail.

Representar todas as suposições relacionadas aos requisitos não é uma prática comum. E mais: pode não ser necessário, uma vez que muitas delas são óbvias. Contudo, explicitar suposições pode ser necessário quando as suposições não forem óbvias ou quando o engenheiro de requisitos quiser enfatizá-las (NEGRI et al., 2017). Assim, quando pertinente, tabelas descrevendo suposições podem ser adicionadas, tipicamente, a Documentos de Especificação de Requisitos.

Referências do Capítulo

AURUM, A., WOHLIN, C., *Engineering and Managing Software Requirements*, Springer-Verlag, 2005.

BASS, L., CLEMENTS, P., KAZMAN, R., *Software Architecture in Practice*, Second edition, Addison Wesley, 2003.

CARDOSO, E.C.S., Uma Comparação entre Requisitos de Sistema Gerados por Técnicas de Modelagem de Processos com Requisitos de Sistema Gerados por Técnicas Convencionais de Engenharia de Requisitos, Projeto de Graduação (Engenharia de Computação), Universidade Federal do Espírito Santo, 2007.

CARDOSO, E., ALMEIDA, J.P.A., GUIZZARDI, G., "Uma Experiência com Engenharia de Requisitos baseada em Modelos de Processos". In: Proceedings of the XI Iberoamerican

Workshop on Requirements Engineering and Software Environments (IDEAS 2008), Recife, 2008.

CARVALHO, E.A., Engenharia de Processos de Negócio e a Engenharia de Requisitos: Análise e Comparações de Abordagens e Métodos de Elicitação de Requisitos de Sistemas Orientada por Processos de Negócio, Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia de Produção, COPPE/UFRJ, Rio de Janeiro, 2009.

DAVENPORT, T. H., *Mission Critical: realizing the promise of enterprise systems.* 1st edition. Boston: Harvard Business School Press, 2000.

FRYE, D. W., GULLEDGE, T. R., "End-to-end Business Process Scenarios". *Industrial, Management & Data Systems*, v. 107, n. 6, pp.749–761, 2007.

ISO/IEC 25010, System and Software Engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.

KENDALL, K.E., KENDALL, J.E.; *Systems Analysis and Design*, Prentice Hall, 8th Edition, 2010.

KNIGHT, D. M. Elicitação de Requisitos de Software a partir do Modelo de Negócio. Dissertação (Mestrado em Informática). Núcleo de Computação Eletrônica (NCE), Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2004

KOTONYA, G., SOMMERVILLE, I., *Requirements engineering: processes and techniques*. Chichester, England: John Wiley, 1998.

NEGRI, P.P., SOUZA, V.E.S., LEAL, A.LC., FALBO, R.A., GUIZZARDI, G., Towards an Ontology of Goal-Oriented Requirements, XX Ibero-American Conference on Software Engineering - CIbSE'2017, Buenos Aires, Argentina, 2017.

PFLEEGER, S.L., *Engenharia de Software: Teoria e Prática*, São Paulo: Prentice Hall, 2ª edição, 2004.

PRESSMAN, R.S., *Engenharia de Software*, McGraw-Hill, 6ª edição, 2006.

ROBERTSON, S., ROBERTSON, J. *Mastering the Requirements Process***.** 2nd Edition. Addison Wesley, 2006.

SOMMERVILLE, I., *Engenharia de Software*, 8ª Edição. São Paulo: Pearson – Addison Wesley, 2007.

WAZLAWICK, R.S., *Análise e Projeto de Sistemas de Informação Orientados a Objetos*, Elsevier, 2004.

WIEGERS, K.E., Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. 2nd Edition, Microsoft Press, Redmond, Washington, 2003.

Capítulo 4 – Análise de Requisitos

A investigação do domínio do problema e do negócio a ser apoiado nos leva a perceber que o desenvolvimento de um novo sistema de informação é um agente de mudanças no negócio e na prática dos profissionais envolvidos. Podemos pensar que estamos diante de duas versões de um sistema: o sistema como ele é (*as-is*) e o sistema que se pretende obter (*to-be*). Note que quando falamos do sistema neste contexto não estamos falando de um sistema computacional, mas sim de sistema em uma acepção mais geral da palavra: um conjunto intelectualmente organizado de elementos (pessoas, produtos de software, dispositivos etc.). O sistema *as-is* apresenta problemas, limitações e deficiências. O sistema *to-be* tem a intenção de tratar esses problemas usando oportunidades oferecidas pela tecnologia. Assim, para que o sistema *to-be* seja bem-sucedido, o sistema de software a ser desenvolvido (software *to-be*) precisa cooperar efetivamente com seu ambiente (pessoas, dispositivos e outros sistemas de software existentes). Assim, podemos estruturar o trabalho de ER em termos de três dimensões, como mostra a Figura 4.1 (LAMSWEERDE, 2009):

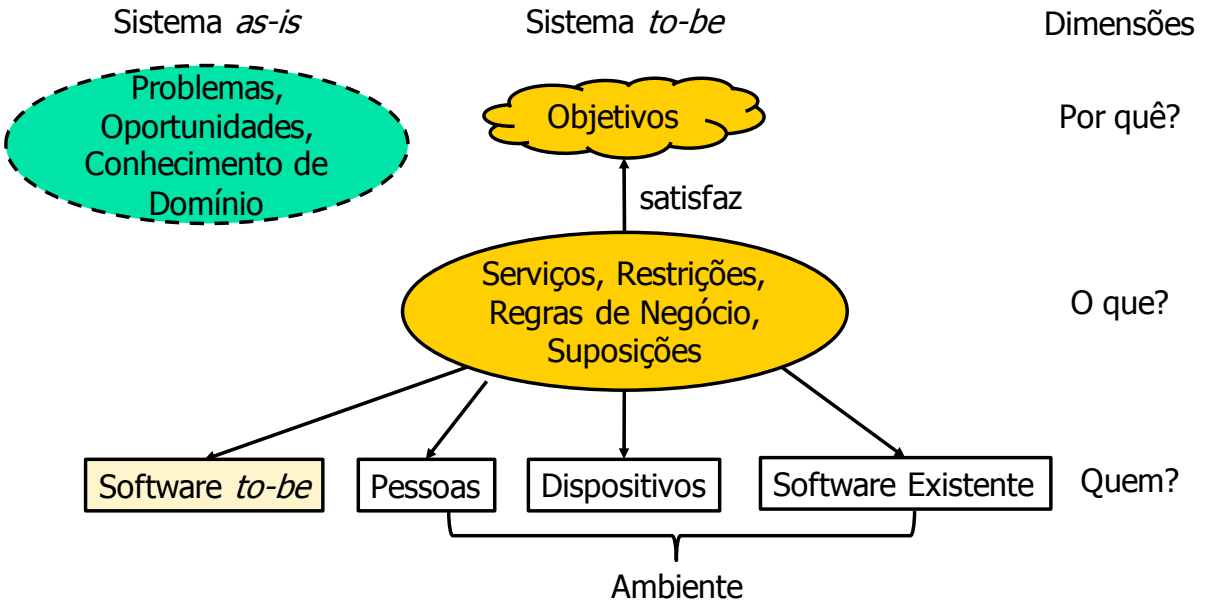


Figura 4.1 – As dimensões da ER (adaptado de (LAMSWEERDE, 2009)).

- **Por quê?** As razões para se desenvolver um novo sistema computacional (software *to-be*) devem ser explicitadas em termos de objetivos a serem satisfeitos.
- **O que?** Esta dimensão preocupa-se com os serviços (requisitos funcionais) que o novo sistema computacional deve prover para satisfazer os objetivos. Esses serviços devem satisfazer restrições (requisitos não funcionais), garantir regras de negócio e apoiam-se em suposições.

- **Quem?** Alguns serviços serão implementados pelo software *to-be*, enquanto outros serão realizados por procedimentos manuais, operações de dispositivos ou por outros produtos de software já existentes.

Para responder a essas perguntas, é necessário um entendimento mais profundo do domínio do problema, do negócio a ser apoiado, das limitações do sistema atual e dos requisitos para o novo sistema. Requisitos de cliente são insuficientes para isso. Eles resultam diretamente da atividade de levantamento de requisitos, sendo tipicamente descritos em linguagem natural, em um baixo nível de detalhes. Assim, uma vez identificados os requisitos de cliente, é necessário detalhá-los, colocando-os no nível de descrição de requisitos de sistema. Este é o propósito da Análise de Requisitos.

Requisitos de sistema resultam dos esforços dos analistas de organizar e detalhar requisitos de cliente, envolvendo a elaboração de um conjunto de modelos abstratos do sistema, agora em um nível alto de detalhes. A correta derivação de requisitos de sistema a partir de requisitos de cliente é fundamental para assegurar que equívocos não serão arbitrariamente introduzidos pelos engenheiros de requisitos na especificação de requisitos (AURUM; WOHLIN, 2005). Durante a análise de requisitos, requisitos funcionais e não funcionais devem ser expressos em um nível de detalhes que permita guiar as etapas subsequentes do desenvolvimento de software (projeto, implementação e testes). Além disso, atenção especial deve ser dada às regras de negócio. Elas têm de ser capturadas e incorporadas às funcionalidades do sistema. Neste contexto, vale destacar que a aplicação de técnicas de levantamento de requisitos é imprescindível e, portanto, o levantamento detalhado de requisitos ocorre em paralelo com a análise de requisitos.

Para descrever requisitos funcionais detalhadamente, é necessário produzir modelos, cada um deles capturando uma perspectiva diferente do sistema. A linguagem natural ainda é utilizada, mas em uma escala reduzida, circunscrita a descrições de certos modelos ou às definições em glossários ou dicionários de dados. Grande parte das informações tratadas na análise de requisitos funcionais é melhor comunicada por meio de diagramas do que por meio de texto. Assim, a modelagem é uma atividade essencial da análise de requisitos.

A modelagem de objetivos pode ser empregada para tratar a dimensão "Por quê?". Um modelo de objetivo procura capturar precisamente os objetivos a serem satisfeitos pelo sistema *to-be*, suas ramificações em termos de subobjetivos, como esses objetivos interagem (conflitos e sinergias) e como eles estão alinhados a objetivos de negócio (LAMSWEERDE, 2009).

A modelagem conceitual visa definir em detalhes as funções requeridas pelo sistema e o conhecimento necessário para realizá-las. O produto principal da modelagem conceitual é o esquema conceitual do sistema (OLIVÉ, 2007). O esquema ou modelo conceitual[3] de um sistema captura as funções e informações que o sistema deve prover e gerenciar. Ele deve ser concebido com foco no domínio do problema e não no domínio da solução, mas deve tratar tanto uma visão externa do sistema (como o sistema é percebido pelos usuários) quanto uma visão interna do mesmo (como as abstrações do domínio são representadas e relacionadas).

[3] Olivé (2007) utiliza o termo "esquema conceitual" para designar o conjunto de artefatos que capturam o conhecimento que um sistema de informação necessita para realizar suas funções. Contudo, a maioria dos textos, tais como (WAZLAWICK, 2004) e (BLAHA; RUMBAUGH, 2006), utiliza o termo "modelo conceitual" para designar esse conjunto de artefatos. Olivé utiliza o termo "modelo conceitual" para designar tipos de modelos, tais como modelos de objetos, modelos de casos de uso etc. Nestas notas de aula, o termo "modelo conceitual" é usado como na maioria dos textos, não sendo feita uma distinção precisa entre as duas acepções do termo. De maneira geral, o contexto indica a que se refere o termo.

Os termos análise de sistemas e análise de requisitos são muitas vezes empregados para designar as atividades de modelagem conceitual (análise de requisitos funcionais). Assim, a maioria dos métodos de análise de sistemas concentra-se na análise de requisitos funcionais, nada falando sobre a análise de requisitos não funcionais. Entretanto, durante a atividade de análise de requisitos, tanto requisitos funcionais quanto requisitos não funcionais devem ser especificados em detalhes.

O produto de trabalho principal da análise de requisitos é o Documento de Especificação de Requisitos. Esse documento deve conter os requisitos funcionais e não funcionais descritos em nível de requisitos de sistema. Conforme citado anteriormente, os requisitos funcionais de sistema são descritos por um conjunto de modelos inter-relacionados. Os requisitos não funcionais de sistema detalham os requisitos não funcionais de usuário, adicionando a eles critérios de aceitação.

É importante apontar que há uma forte dependência entre os métodos e técnicas usados na modelagem conceitual e o paradigma de desenvolvimento adotado. Isso ocorre porque um modelo é uma representação do sistema segundo um particular metamodelo. Esse metamodelo corresponde ao conjunto de elementos de modelagem (estruturais e comportamentais) e regras de uso desses elementos, o qual permite construir modelos segundo o respectivo paradigma (AURUM; WOHLIN, 2005). Assim, para a modelagem conceitual de requisitos funcionais é necessário escolher um paradigma de desenvolvimento (e o correspondente metamodelo subjacente a ele) a partir do qual os modelos serão construídos. O paradigma orientado a objetos, por exemplo, fornece um conjunto de elementos de modelagem que permite modelar um sistema como sendo composto de objetos organizados em classes que se comunicam entre si por meio de troca de mensagens. Uma classe define as propriedades (atributos, relacionamentos e operações) que todos os objetos dela podem possuir. Assim, classes, objetos, associações, atributos e operações, dentre outros, são elementos do metamodelo subjacente ao paradigma orientado a objetos. Neste texto, o paradigma adotado é o da orientação a objetos.

Uma vez que a modelagem conceitual é uma tarefa essencial e complexa, é útil seguir um método. Um método pode ser visto como uma maneira sistemática de trabalhar para se obter um resultado desejado (PASTOR; MOLINA, 2007). Há diversos métodos de análise orientada a objetos propostos na literatura, dentre eles os apresentados em (LARMAN, 2007), (WAZLAWICK, 2004)[4], (BLAHA; RUMBAUGH, 2006) e (PASTOR; MOLINA, 2007). Vale ressaltar que não há um método reconhecido como um método padrão para o desenvolvimento de software orientado a objetos. Diferentes projetos podem requerer diferentes processos e, portanto, não há como definir um método adequado a quaisquer situações. Neste capítulo é apresentado um método que incorpora ideias de vários métodos, o qual tem se mostrado eficaz na prática, sobretudo, no desenvolvimento de sistemas de informação.

Este capítulo trata da análise de requisitos, discutindo a análise de objetivos, requisitos funcionais (modelagem conceitual) e não funcionais. A Seção 4.1 discute a análise de objetivos. A Seção 4.2 discute a modelagem conceitual com foco em sistemas de informação e os tipos de modelos comumente utilizados no desenvolvimento de software orientado a objetos. A Seção 4.3 apresenta a Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML), amplamente usada na modelagem conceitual. A Seção 4.4 apresenta o método de análise de requisitos funcionais sugerido, indicando suas atividades e modelos a serem produzidos. A

[4] O método apresentado em (WAZLAWICK, 2004) é, segundo o próprio autor, "uma interpretação e um detalhamento do método de análise e projeto apresentado por Larman".

Seção 4.5 trata da especificação de requisitos não funcionais. Finalmente, a Seção 4.6 discute a documentação de requisitos em nível de sistema.

4.1 – Análise de Objetivos

Os requisitos funcionais descrevem o que um sistema deve fazer. Os objetivos explicam o porquê. Ao se deixar claros os objetivos subjacentes aos requisitos (tanto funcionais quanto não funcionais), ganha-se um critério de completude/pertinência dos requisitos, abre-se espaço para exploração de alternativas e detecção de conflitos entre *stakeholders*, garantindo maior estabilidade.

Um objetivo é uma declaração de uma intenção que o sistema *to-be* como um todo deve satisfazer por meio de cooperação entre seus agentes (pessoas desempenhando papéis específicos, dispositivos, outros sistemas de software existentes e o novo software *to-be*). Seja o caso do seguinte objetivo a ser satisfeito por um sistema de biblioteca: "Exemplares de livros devem ser devolvidos dentro do prazo previsto". Ele envolve os usuários (que devem devolver os exemplares) e o sistema de informação da biblioteca (que deve acompanhar os empréstimos e emitir lembretes) (LAMSWEERDE, 2009).

Objetivos podem ser descritos em diferentes níveis de abstração. Em níveis mais altos, ditos objetivos estratégicos, os objetivos são mais gerais e relacionados ao negócio ou à organização. Em níveis mais baixos, ditos objetivos táticos ou técnicos, os objetivos são mais específicos e relacionados ao sistema ou a opções de projeto do sistema. A diferença entre níveis de objetivos sugere o uso de um mecanismo de estruturação de objetivos baseado em ligações de contribuição entre objetivos: um objetivo mais geral pode ser refinado em objetivos mais específicos (subobjetivos) contribuindo para a satisfação do primeiro (LAMSWEERDE, 2009). Esse mecanismo é a base para a modelagem de objetivos.

A análise de objetivos inicia com a identificação dos interessados e seus objetivos. Objetivos estratégicos são obtidos junto aos interessados e refinados em objetivos táticos. Esse refinamento é capturado em um modelo de objetivos. Uma vez produzido um modelo de objetivos, o mesmo pode ser usado como insumo para a descrição da funcionalidade do sistema por meio de modelos de casos de uso. A modelagem de objetivos é discuta em mais detalhes no Capítulo 5, o qual trata também da Modelagem de Casos de Uso. Abordagens de ER que têm como ponto de partida objetivos são ditas abordagens de Engenharia de Requisitos Orientadas a Objetivos (*Goal-Oriented Requirements Engineering* – GORE).

4.2 – Modelagem Conceitual

Todo sistema incorpora um esquema conceitual. Assim, para que o desenvolvimento de um sistema seja bem-sucedido, é necessário explicitar esse esquema. Esse é o propósito da modelagem conceitual (OLIVÉ, 2007).

O esquema conceitual de um sistema de informação é a especificação de seus requisitos funcionais. Um sistema de informação é um sistema projetado para coletar, armazenar, processar e distribuir informação sobre o estado de um domínio. Assim, um sistema dessa natureza tem tipicamente três propósitos ou funções principais (OLIVÉ, 2007):

- Função de Memória: sob esta ótica, o objetivo de um sistema de informação é manter uma representação interna do estado do domínio. O estado do domínio geralmente

é alterado com frequência e de muitas maneiras. O sistema precisa acompanhar essas alterações e atualizar sua representação interna adequadamente.

- Função Informativa: o sistema deve prover aos usuários informações sobre o estado do domínio. A função informativa não altera o estado do domínio; ela apenas provê a informação solicitada pelos usuários.
- Função Ativa: o sistema pode realizar ações que modificam o estado do domínio. Para realizar essa função, o sistema precisa conhecer as ações que ele pode tomar, quando elas devem ser tomadas e como elas vão afetar o estado do domínio.

Todos os sistemas de informação convencionais realizam, pelo menos, as funções de memória e informativa. Para ser capaz de realizar suas funções, um sistema tem de ter um conhecimento geral sobre o domínio da aplicação e sobre as funcionalidades que ele deve executar. Na área de sistemas de informação, esse conhecimento é denominado de esquema conceitual (OLIVÉ, 2007).

O modelo conceitual de um sistema é tipicamente composto de vários modelos, cada um deles enfocando uma perspectiva diferente, mas guardando alguma relação com outros modelos. Os diferentes tipos de modelos conceituais podem ser agrupados em duas grandes categorias:

1. Modelos Estruturais: procuram capturar os principais conceitos do domínio, suas relações e propriedades, que são relevantes para o sistema que se está desenvolvendo. Abstração é um mecanismo chave e a definição do que é relevante é definido com base no propósito do sistema. Dentre os tipos de modelos estruturais, destacam-se o Modelo de Entidades e Relacionamentos e o Diagrama de Classes na Orientação a Objetos.
2. Modelos Comportamentais: especificam as ações que o sistema pode realizar e as mudanças válidas no estado do domínio (OLIVÉ, 2007). Tipos de modelos comportamentais bastante utilizados no desenvolvimento orientado a objetos incluem Modelos de Casos de Uso e Modelos Dinâmicos, tais como Diagramas de Transição de Estados e Diagramas de Interação.

4.3 – A Linguagem de Modelagem Unificada

A Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML) é uma linguagem gráfica padrão para especificar, visualizar, documentar e construir artefatos de sistemas de software (BOOCH; RUMBAUGH; JACOBSON, 2006).

A UML foi criada na década de 1990, a partir de uma tentativa de se unificar dois dos principais métodos orientados a objetos utilizados até então: a Técnica de Modelagem de Objetos (*Object Modeling Technique* – OMT) (RUMBAUGH et al., 1994) e o Método de Booch (BOOCH, 1994). Inicialmente, buscava-se criar um método unificado. A este esforço juntou-se também Ivar Jacobson, fundindo também seu método OOSE (JACOBSON, 1992). Contudo, percebeu-se que não era possível estabelecer um único método adequado para todo e qualquer projeto de desenvolvimento. De fato, um método é composto por uma linguagem estabelecendo a notação a ser usada na elaboração dos artefatos a serem produzidos e de um processo descrevendo que artefatos construir e como construí-los. A linguagem pode ser unificada, mas a decisão de quais artefatos produzir e que passos seguir não é passível de padronização, já que pode variar de projeto para projeto. Assim, ao invés de criarem um método

unificado, Rumbaugh, Booch e Jacobson propuseram a UML, incorporando as principais notações para os artefatos de seus métodos e de vários outros, com a colaboração de várias empresas e autores. A UML foi aprovada em novembro de 1997 pelo *Object Management Group* (OMG) pondo fim a uma guerra de métodos orientados a objeto. Sua versão mais recente, a UML 2.5, foi publicada 2015 e é resultado de um esforço de diversos colaboradores, envolvendo empresas e pesquisadores sob a coordenação do OMG.

Vale realçar que a UML é somente uma linguagem e, portanto, é apenas parte de um método de desenvolvimento de software. Ela é independente do processo de software a ser usado, ainda que seja mais adequada a processos de desenvolvimento orientados a objetos. Ela se destina a visualizar, especificar, construir e documentar artefatos de software (BOOCH; RUMBAUGH; JACOBSON, 2006). No contexto da Engenharia de Requisitos, a UML provê diversos diagramas que podem ser usados na modelagem de requisitos, tanto segundo a perspectiva estrutural quanto segundo a perspectiva comportamental. Para a modelagem conceitual estrutural, os diagramas de classes são amplamente utilizados. Para a modelagem comportamental, podem ser usados diagramas de casos de uso, de interação, de estados e de atividades. A seguir, é apresentada uma descrição sucinta de cada um desses diagramas, baseada em (BOOCH; RUMBAUGH; JACOBSON, 2006):

- Diagrama de Classes: modela um conjunto de classes e seus relacionamentos. Diagramas de classes proveem uma visão estática da estrutura de um sistema e, portanto, são usados na modelagem conceitual estrutural.
- Diagrama de Casos de Uso: mostra um conjunto de casos de uso e atores e seus relacionamentos. Os casos de uso descrevem a funcionalidade do sistema percebida pelos atores externos. Um ator interage com o sistema, podendo ser um usuário humano, dispositivo de hardware ou outro sistema. Diagramas de casos de uso proveem uma visão das funcionalidades do sistema.
- Diagrama de Gráfico de Estados (ou simplesmente Diagrama de Estados): mostra os estados pelos quais os objetos de uma classe específica podem passar ao longo de suas vidas, em resposta a estímulos recebidos, juntamente com suas ações. Os diagramas de estados proveem uma visão dinâmica dos objetos de uma classe, sendo importantes para modelar o comportamento de objetos de uma classe em resposta à ocorrência de eventos.
- Diagrama de Atividades: mostra a estrutura de um processo. Provê uma visão dinâmica do sistema (ou de uma porção do sistema) e pode ser usado tanto para modelar processos de negócio quanto para modelar funções do sistema. Um diagrama de atividades dá ênfase ao fluxo de controle entre objetos.
- Diagrama de Interação: mostra um conjunto de objetos interagindo, incluindo as mensagens que podem ser trocadas entre eles. Provê uma visão dinâmica do comportamento de um sistema ou de uma porção do sistema. Há dois tipos de diagramas de interação:
  - Diagrama de Sequência: dá ênfase à ordenação temporal das mensagens.
  - Diagrama de Comunicação: tem o mesmo propósito do diagrama de sequência, apresentando, contudo, ênfase na organização estrutural dos objetos que enviam ou recebem mensagens.

No contexto da Engenharia de Requisitos, além dos diagramas anteriormente citados, merecem atenção também os diagramas de pacotes. Na UML, pacote é um mecanismo de

propósito geral usado para organizar elementos de modelagem em grupos. Assim, um diagrama de pacotes mostra a decomposição de um modelo em unidades menores e suas dependências (BOOCH; RUMBAUGH; JACOBSON, 2006).

Vale ressaltar mais uma vez, que a UML é uma linguagem de modelagem, não um método de desenvolvimento OO. Os métodos consistem, pelo menos em princípio, de uma linguagem de modelagem e um processo de uso dessa linguagem. A UML não prescreve esse processo de utilização e, portanto, deve ser aplicada no contexto de um processo, lembrando que projetos diferentes podem requerer processos diferentes. Sendo assim, na próxima seção é apresentado um método geral de análise de requisitos funcionais que aponta quais diagramas da UML adotar e um processo de elaboração dos mesmos.

4.4 – Um Método de Análise de Requisitos Funcionais

Uma vez que tipicamente diversos modelos do sistema são produzidos, surgem algumas importantes questões: Que modelos produzir? Em que sequência? Quais as relações existentes entre esses modelos? Estas questões podem ser parcialmente respondidas pela adoção de um método de análise.

Um método é composto por uma linguagem estabelecendo a notação a ser usada na elaboração dos artefatos a serem produzidos e de um processo descrevendo que artefatos construir e como construí-los.

O método sugerido neste texto adota a Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML) (BOOCH; RUMBAUGH; JACOBSON, 2006) como linguagem de modelagem e prescreve o processo ilustrado na Figura 4.1. Nessa figura, as atividades de modelagem propriamente ditas estão destacadas em amarelo. As demais atividades correspondem a atividades de documentação e verificação e validação do Documento de Especificação de Requisitos.

Se uma abordagem baseada em objetivos (*Goal-Oriented Requirements Engineering* – GORE) for adotada, uma nova atividade deve ser inserida no início do processo: a Análise de Objetivos (ver Seção 4.2). A análise de objetivos é estudada em mais detalhes no Capítulo 5. Alternativamente, a Análise de Objetivos pode ser vista como uma atividade anterior à Análise de Requisitos e, portanto, fora do escopo do processo proposto. Assim, a Análise de Requisitos seria parte do Levantamento de Requisitos e o modelo de objetivos incorporado ao Documento de Definição de Requisitos.

Em uma abordagem convencional (i.e., não baseada em objetivos), o primeiro modelo a ser construído é o modelo de casos de uso. Sua escolha como primeiro modelo deve-se ao fato do modelo de casos de uso ser muito simples e, portanto, passível de compreensão tanto por desenvolvedores – analistas, projetistas, programadores e testadores – como pela comunidade usuária – clientes, usuários e demais interessados. O modelo de casos de uso é um modelo comportamental, mostrando as funções do sistema de maneira estática. Ele é composto de dois artefatos: os diagramas de casos de uso e as descrições de casos de uso. O diagrama de casos de uso descreve graficamente o sistema, seu ambiente e como sistema e ambiente se relacionam. Assim, ele descreve o sistema segundo uma perspectiva externa. As descrições dos casos de uso descrevem o passo a passo para a realização dos casos de uso e são essencialmente textuais. Elas tratam de como os casos de uso são realizados internamente, complementando os diagramas. A modelagem de casos de uso é estudada em detalhes no Capítulo 5.

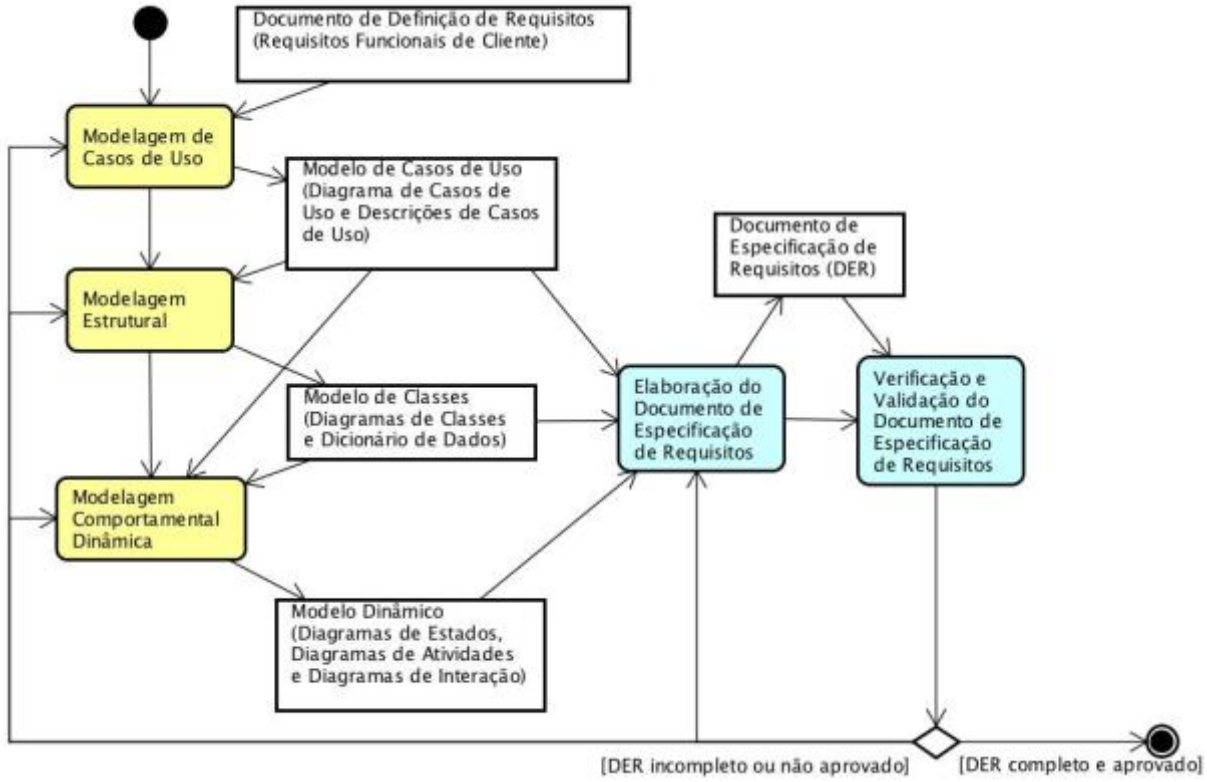


Figura 4.1 – O Método de Análise de Requisitos Funcionais Proposto.

Tomando por base casos de uso e suas descrições, é possível passar à modelagem conceitual estrutural, quando os conceitos e relacionamentos envolvidos no domínio são capturados em um conjunto de diagramas de classes. Neste momento é importante definir, também, o significado dos conceitos e de suas propriedades, bem como restrições sobre eles. Essas definições são documentadas em um dicionário de dados do projeto. A modelagem conceitual estrutural é o foco do Capítulo 6.

Algumas classes do modelo estrutural apresentam um comportamento dependente de seu estado. Para essas classes, é útil elaborar diagramas de estados, mostrando os estados pelos quais um objeto da classe pode passar ao longo de sua existência e os eventos que provocam transições de estados. Além disso, uma vez que casos de uso são representados apenas de maneira estática, pode ser útil também mostrar a dinâmica de um caso de uso, representando como objetos do diagrama de classes interagem para realizar um caso de uso. Os diagramas de interação são elaborados com este propósito. Finalmente, pode ser útil representar os passos de um caso de uso na forma de um diagrama mostrando objetos gerados e os atores envolvidos em cada passo. Para tal, diagramas de atividades podem ser elaborados. A elaboração de diagramas de estados, de interação e de atividades é o foco da atividade de modelagem comportamental dinâmica, a qual é discutida com maiores detalhes no Capítulo 7.

Deve-se frisar que, apesar da Figura 4.1 sugerir que os passos do método são sequenciais, na prática, isso não ocorre. Paralelamente à modelagem de casos de uso, pode-se iniciar a modelagem conceitual estrutural. Os diagramas de atividade podem também ser elaborados em conjunto com a definição dos casos de uso, de maneira a complementar a descrição de casos de uso específicos. Diagramas de estado e de interação requerem a definição preliminar de casos de uso e classes. Contudo, podem ter sido definidos apenas alguns casos de

uso e classes (e não todos eles) para já iniciar a elaboração desses diagramas. Assim, as atividades mostradas na Figura 4.1 são fortemente paralelas e iterativas.

Paralelamente a todas as atividades de modelagem, o documento de especificação de requisitos deve ir sendo elaborado. A verificação e a validação também podem ser feitas por partes e não para o documento como um todo. Por exemplo, é bastante comum validar primeiro somente os casos de uso. Verificações de consistência, tais como as feitas entre casos de uso e classes, ou casos de uso, diagramas de interação e diagramas de classes, podem ser feitas separadamente uma das outras. Assim, as atividades de documentação, verificação e validação são atividades contínuas que ocorrem paralelamente à modelagem conceitual.

4.5 – Especificação de Requisitos Não Funcionais

Assim como os requisitos funcionais precisam ser especificados em detalhes, o mesmo acontece com os requisitos não funcionais. Para os atributos de qualidade considerados prioritários, o analista deve trabalhar no sentido de especificá-los de modo que eles se tornem mensuráveis e, por conseguinte, testáveis.

Para cada atributo de qualidade, devem-se definir as medidas a serem usadas, indicando a unidade da medida e sua escala, e os valores mínimo, alvo e máximo. Pode-se, ainda, perguntar aos usuários o que constituiria um valor inaceitável para o atributo e definir testes que tentem forçar o sistema a demonstrar tais características (WIEGERS, 2003). Ao se estabelecer uma escala de medição e os valores aceitáveis, o requisito é transformado de uma intenção vaga, e até certo ponto ambígua, em um requisito mensurável e bem formado. Estabelecida uma escala, pode-se perguntar ao usuário o que é considerado uma falha em atender ao requisito, de modo a definir o critério de aceitação do mesmo (ROBERTSON; ROBERTSON, 2006). Assim, na especificação de requisitos de sistema, é importante transformar um requisito de usuário em um requisito mensurável, adicionando a ele um critério de aceitação.

A ISO/IEC 25023 – Medidas de Qualidade de Produtos de Software e Sistemas (ISO/IEC, 2016), ou a sua antecessora, a ISO/IEC 9126, pode ser uma boa fonte de medidas. As partes 2 (Medidas Externas) (ISO/IEC, 2003a) e 3 (Medidas Internas) (ISO/IEC, 2003b) da ISO/IEC 9126 apresentam diversas medidas que podem ser usadas para especificar objetivamente os requisitos não funcionais. Essas medidas foram revistas e atualizadas na ISO/IEC 25023. Nessas normas, medidas são sugeridas para as diversas subcaracterísticas de qualidade externa e interna descritas na atual ISO/IEC 25010 (ISO/IEC, 2011), indicando, dentre outros, nome e propósito da medida, método de aplicação e fórmula, e como interpretar os valores da medida.

Seja o exemplo de um sistema que tem como requisito não funcional ser fácil de aprender. Esse requisito poderia ser especificado conforme mostrado na Tabela 4.1.

Tabela 4.1 – Especificação de Requisito Não Funcional.

<table>
<tr><td colspan="3">RNF01 – A funcionalidade “Efetuar Locação de Item” deve ser fácil de aprender.</td></tr>
<tr><td rowspan="4">Medida: Facilidade de Aprendizagem de função (Ease of function learn) (ISO/IEC, 2003a)</td><td>Descrição:</td><td>Facilidade de aprender a realizar uma tarefa em uso.</td></tr>
<tr><td>Propósito:</td><td>Quanto tempo o usuário leva para aprender a realizar uma tarefa especificada eficientemente?</td></tr>
<tr><td>Método de Aplicação:</td><td>Observar o comportamento do usuário desde quando ele começa a aprender até quando ele começa a operar eficientemente.</td></tr>
<tr><td>Medição:</td><td>T = soma do tempo de operação do usuário até que ele consiga realizar a tarefa em um tempo especificado (tempo requerido para aprender a operação para realizar a tarefa).</td></tr>
<tr><td>Critério de Aceitação:</td><td colspan="2">T <= 15 minutos, considerando que o usuário está operando o sistema eficientemente quando a tarefa “Efetuar Locação” é realizada em um tempo inferior a 2 minutos.</td></tr>
</table>

4.6 – O Documento de Especificação de Requisitos

Os requisitos de sistema, assim como foi o caso dos requisitos de cliente, têm de ser especificados em um documento, de modo a poderem ser verificados e validados e posteriormente usados como base para as atividades subsequentes do desenvolvimento de software. O Documento de Especificação de Requisitos tem como propósito registrar os requisitos escritos a partir da perspectiva do desenvolvedor e, portanto, deve incluir os vários modelos conceituais desenvolvidos, bem como a especificação dos requisitos não funcionais detalhados.

Diferentes formatos podem ser propostos para documentos de especificação requisitos, bem como mais de um documento pode ser usado para documentar os requisitos de sistema. Neste texto, propõe-se o uso de um único documento, contendo as seguintes informações:

- Introdução: breve introdução ao documento, descrevendo seu propósito e estrutura.
- Modelo de Casos de Uso: apresenta o modelo de casos de uso do sistema, incluindo os diagramas de casos de uso e as descrições de casos de uso associadas.
- Modelo Estrutural: apresenta o modelo conceitual estrutural do sistema, incluindo os diagramas de classes do sistema.
- Modelo Dinâmico: apresenta os modelos comportamentais dinâmicos do sistema, incluindo os diagramas de estados, diagramas de interação e diagramas de atividades.
- Dicionário do Projeto: apresenta as definições dos principais conceitos capturados pelos diversos modelos e restrições de integridade a serem consideradas, servindo como um glossário do projeto.
- Especificação dos Requisitos Não Funcionais: apresenta os requisitos não funcionais descritos no nível de sistema, o que inclui critérios de aceitação.

É importante frisar que dificilmente um sistema é simples o bastante para ser modelado como um todo. Quase sempre é útil dividir um sistema em unidades menores, mais fáceis de serem gerenciáveis, ditas subsistemas. É útil organizar a especificação de requisitos por subsistemas e, portanto, cada uma das seções propostas acima pode ser subdividida por subsistemas.

Um modelo estrutural para uma aplicação complexa, por exemplo, pode ter centenas de classes e, portanto, pode ser necessário definir uma representação concisa capaz de orientar um leitor em um modelo dessa natureza. O agrupamento de elementos de modelo em subsistemas serve basicamente a este propósito, podendo ser útil também para a organização de grupos de trabalho em projetos extensos. A base principal para a identificação de subsistemas é a complexidade do domínio do problema. Através da identificação e agrupamento de elementos de modelo em subsistemas, é possível controlar a visibilidade do leitor e, assim, tornar os modelos mais compreensíveis.

A UML provê um tipo principal de item de agrupamento, denominado pacote, que é um mecanismo de propósito geral para a organização de elementos da modelagem em grupos. Um diagrama de pacotes mostra a decomposição de um modelo em unidades menores e suas dependências, como ilustra a Figura 4.4. A linha pontilhada direcionada indica que o pacote origem (no exemplo, o pacote Atendimento a Cliente) depende do pacote destino (no exemplo, o pacote Controle de Acervo).

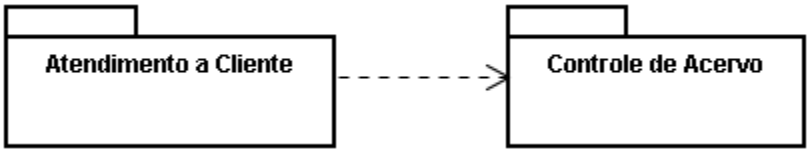


Figura 4.4 – Exemplo de um Diagrama de Pacotes

Por fim, vale ressaltar que, caso uma abordagem baseada em objetivos (GORE) seja adotada e o modelo de objetivos seja considerado parte da especificação de requisitos, então uma seção adicional, no início do documento, deve ser adicionada para apresentar e descrever o modelo de objetivos.

Referências do Capítulo

AURUM, A., WOHLIN, C., *Engineering and Managing Software Requirements*, Springer-Verlag, 2005.

BASS, L., CLEMENTS, P., KAZMAN, R., *Software Architecture in Practice*, Second edition, Addison Wesley, 2003.

BLAHA, M., RUMBAUGH, J., *Modelagem e Projetos Baseados em Objetos com UML 2*, Elsevier, 2006.

BOOCH, G., *Object-Oriented Analysis and Design with Applications*, 2nd edition, Benjamin/Cummings Publishing Company, Inc, 1994.

BOOCH, G., RUMBAUGH, J., JACOBSON, I., *UML Guia do Usuário*, 2a edição, Elsevier Editora, 2006.

ISO/IEC 25010, System and Software Engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.

ISO/IEC 25023, System and Software Engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality, 2016.

ISO/IEC 9126-1, Software Engineering - Product Quality - Part 1: Quality Model, 2001.

ISO/IEC TR 9126-2:2003, Software Engineering – Product Quality – Part 2: External Metrics, 2003a.

ISO/IEC TR 9126-3:2003, Software Engineering – Product Quality – Part 3: Internal Metrics, 2003b.

JACOBSON, I.; *Object-Oriented Software Engineering*, Addison-Wesley, 1992.

LAMSWEERDE, A., *Requirements Engineering – From System Goals to UML Models to Software Specifications*, Wiley, 2009.

LARMAN, C., *Utilizando UML e Padrões*, 3ª edição, Bookman, 2007.

OLIVÉ, A., *Conceptual Modeling of Information Systems*, Springer, 2007.

PASTOR, O., MOLINA, J.C., *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*, Springer, 2007.

ROBERTSON, S., ROBERTSON, J., *Mastering the Requirements Process***.** 2nd Edition. Addison Wesley, 2006.

RUMBAUGH, J., et al.; *Modelagem e Projetos Baseados em Objetos*, 1ª Edição, Editora Campus, 1994.

WAZLAWICK, R.S., *Análise e Projeto de Sistemas de Informação Orientados a Objetos*, Elsevier, 2004.

WIEGERS, K.E., *Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle*. 2nd Edition, Microsoft Press, Redmond, Washington, 2003.

YOURDON, E., *Object-Oriented Systems Design: an Integrated Approach*, Yourdon Press Computing Series, Prentice Hall, 1994.

Capítulo 5 – Modelagem de Objetivos e de Casos de Uso

A análise de requisitos é um processo que envolve a construção de diversos modelos. Diagramas de classes e diagramas de interação incluem detalhes relacionados à estrutura interna dos objetos, suas associações, como eles interagem dinamicamente e como invocam o comportamento dos demais. Essas informações são necessárias para projetar e construir um sistema, mas não são suficientes para comunicar requisitos funcionais com clientes e usuários. Elas não capturam o conhecimento sobre as tarefas a serem realizadas e, portanto, é difícil avaliar se o sistema a ser construído a partir de um modelo desse tipo, isoladamente, vai realmente atender às necessidades dos usuários.

Assim, é importante construir, primeiro, um modelo que descreva o sistema em termos de suas tarefas, seu ambiente e como sistema e ambiente estão relacionados. Tal modelo deve ser passível de compreensão tanto por desenvolvedores – analistas, projetistas, programadores e testadores – como pela comunidade usuária – clientes e usuários. Modelos de casos de uso (*use cases*) são uma forma de estruturar essa visão. Como o próprio nome sugere, um caso de uso é uma maneira de usar o sistema. Usuários interagem com o sistema, interagindo com seus casos de uso. Tomados em conjunto, os casos de uso de um sistema definem a sua funcionalidade. Casos de uso são, portanto, os "itens" que o desenvolvedor negocia com seus clientes.

Como se pode perceber, casos de uso têm uma estreita relação com requisitos funcionais e podem ser derivados diretamente deles. Assim, quanto mais completa e correta for a identificação de requisitos funcionais, melhor será o modelo de casos de uso resultante. Obviamente, novos requisitos funcionais podem ser identificados quando se elabora um modelo de casos de uso, já que o processo é essencialmente iterativo. Contudo, se for possível utilizar outros mecanismos para apoiar a identificação de requisitos funcionais, tanto melhor. Neste contexto, a análise de objetivos pode ser uma importante aliada. Objetivos tanto proveem razões para os requisitos quanto direcionam a identificação de requisitos para suportá-los (LAMSWEERDE, 2009). Assim, antes de se realizar a modelagem de casos de uso é interessante fazer uma análise de objetivos.

Vale destacar que, ao contrário dos demais modelos discutidos ao longo deste texto, a análise de objetivos é uma abordagem que ainda não alcançou plenamente a indústria de software, sendo considerada, portanto, parte ainda do estado da arte. Neste sentido, este capítulo busca apoiar essa transição, introduzindo os principais conceitos da abordagem baseada em objetivos para a Engenharia de Requisitos (*Goal-Oriented Requirements Engineering* - GORE). Para maiores detalhes sobre GORE, recomenda-se ver o livro de Lamsweerde (2009).

Este capítulo aborda a análise de objetivos e a modelagem de casos de uso, discutindo os principais elementos de modelos de ambos. A Seção 5.1 trata da análise de objetivo. A Seção 5.2 enfoca a modelagem de casos de uso.

5.1 – Análise de Objetivos

Conforme discutido no Capítulo 4, o trabalho de ER pode ser estruturado em termos de três dimensões/questões (LAMSWEERDE, 2009): Por quê? O que? E quem? A análise de objetivos focaliza o porquê, tendo objetivo como abstração central.

Objetivos são declarações de intenções ou desejos de interessados a serem alcançados (NEGRI et al., 2017). Objetivos são declarações prescritivas, assim como requisitos. Portanto, objetivos são declarativos, ao contrário dos procedimentos operacionais para atingi-los, que são capturados nos modelos de casos de uso. A satisfação de objetivos requer a cooperação de diversos agentes, dentre eles pessoas, dispositivos, produtos de software existente e o produto de software a ser desenvolvido (software *to-be*) (LAMSWEERDE, 2009).

Esta seção aborda a análise de objetivos e está estruturada da seguinte forma: a Subseção 5.1.1 trata dos níveis de granularidade de objetivos; a Subseção 5.1.2 discute tipos de objetivos; a Subseção 5.1.3 aborda a modelagem de objetivos.

5.1.1 - Níveis de Granularidade de Objetivos

Objetivos podem ser expressos em diferentes níveis de abstração/granularidade (LAMSWEERDE, 2009). Em níveis mais altos, há os objetivos mais gerais declarando metas estratégicas relacionadas ao negócio ou à organização. Nos níveis intermediários, há os objetivos táticos. Nos níveis mais baixos, há os objetivos mais específicos tratando de metas técnicas relacionadas ao sistema, ou seja, os requisitos para o sistema. Essa diferença entre níveis e granularidade de objetivos sugere um mecanismo de estruturação de objetivos baseado em ligações de contribuição (*contribution links*) entre objetivos (LAMSWEERDE, 2009). Um objetivo de nível mais alto pode ser refinado em objetivos de granularidade mais fina, de modo que se possa partir de metas organizacionais em direção a requisitos do software a ser desenvolvido.

Essa noção de granularidade de objetivos nos permite deixar mais claras algumas noções importantes da ER. Quanto mais fina a granularidade de um objetivo, menos agentes são necessários para satisfazê-lo. Um requisito de software pode ser visto como um objetivo sob responsabilidade de um único agente: o software a ser desenvolvido (software *to-be*). Por outro lado, uma expectativa é um objetivo sob responsabilidade do ambiente do software *to-be*. Ao contrário de requisitos, expectativas não podem ser garantidas pelo software *to-be*. Expectativas são, portanto, suposições prescritivas para agentes do ambiente. É importante notar que há outros tipos de suposições, as quais são declarativas, e são ditas hipóteses. Em suma, um objetivo pode corresponder a um requisito de software ou não, dependendo dos agentes envolvidos em sua satisfação (LAMSWEERDE, 2009). A Figura 5.1 mostra os tipos de declarações tipicamente tratados na ER quando adotada uma abordagem GORE.

5.1.2 - Tipos de Objetivos

Em GORE, objetivos são tipicamente classificados ao longo de duas dimensões principais, ortogonais entre si, como mostra a Figura 5.2.

Figura 5.1 - Tipologia de declarações em GORE (adaptado de (LAMSWEERDE, 2009)).

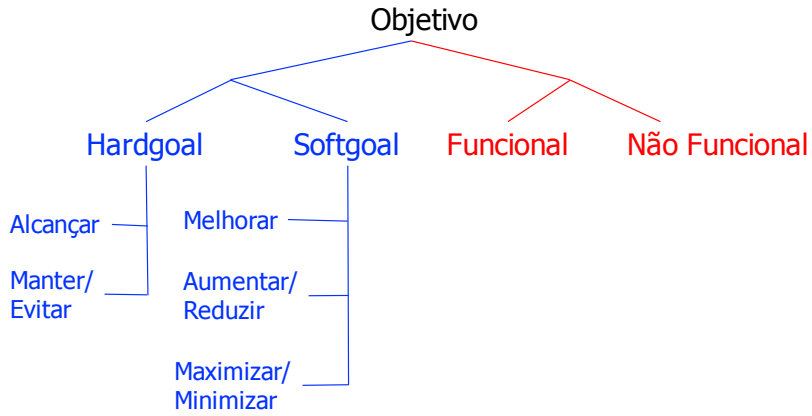


Figura 5.2 - Taxonomia de Objetivos (adaptado de (LAMSWEERDE, 2009)).

A primeira dimensão refere-se ao fato de ser possível, ou não, estabelecer claramente um critério de satisfação para o objetivo. *Hardgoals* (chamados de objetivos comportamentais em (LAMSWEERDE, 2009)) possuem um critério de satisfação claramente estabelecido. Um *hardgoal* é uma proposição que é objetivamente satisfeita por um estado de coisas (GUIZZARDI et al., 2014). Assim, é possível falar em satisfação de um *hardgoal*. Um *softgoal*, por outro lado, é uma expressão vaga de uma intenção para a qual não se consegue estabelecer claramente um critério de satisfação (GUIZZARDI et al., 2014). Ele prescreve preferências em relação a comportamentos alternativos do sistema e, portanto, ele é "mais satisfeito" por certas alternativas e menos por outras. Assim, não faz muito sentido falar em "satisfação do objetivo", pois, na verdade, um *softgoal* é "mais satisfeito" por uma alternativa do que por outra (LAMSWEERDE, 2009).

Hardgoals podem ser objetivos relacionados a alcançar ou manter situações (LAMSWEERDE, 2009). Um *hardgoal* relacionado a alcançar uma situação prescreve comportamentos pretendidos onde se busca alcançar tal situação. Quando se deseja destacar o tipo do objetivo, pode-se preceder o nome do objetivo com a palavra "Alcançar". Assim, um objetivo dessa natureza pode ser escrito da seguinte forma: [Alcançar] *Situação Alvo*. Esse tipo de objetivo indica que, dada a situação atual, mais cedo ou mais tarde a situação alvo deve ser alcançada. P.ex., o objetivo "Alcançar Requisição de Exemplar Satisfeita" (ou simplesmente "Requisição de Exemplar Satisfeita") indica que, se um exemplar de um livro for requisitado (situação atual), então mais cedo ou mais tarde um exemplar do livro será emprestado para o usuário (situação alvo). Caso se deseje especificar o intervalo de tempo para satisfação do objetivo, essa informação deve ser adicionada ao objetivo. P.ex., "Requisição de exemplar

satisfeita em até uma semana". Um *hardgoal* relacionado a manter uma situação prescreve comportamentos pretendidos que buscam manter tal situação. Um objetivo dessa natureza pode ser escrito da seguinte forma: [Manter] *Situação*. P.ex., o objetivo "Manter Classificação de Livros Correta" (ou simplesmente "Classificação de Livros Correta") indica que, se um livro é registrado corretamente na biblioteca, então sua correta classificação deve ser mantida.

Softgoals, por sua vez, podem ser relacionados a melhorar uma certa situação, aumentar/reduzir quantidades, ou maximizar/minimizar certos aspectos. Ex.: O trabalho dos bibliotecários na classificação de livros deve ser reduzido (LAMSWEERDE, 2009).

A segunda dimensão refere-se à distinção entre objetivos funcionais e não funcionais (a mesma distinção já discutida para requisitos de software). Objetivos/requisitos funcionais referem-se a funções que se pretende que o sistema a ser construído proveja. Objetivos/ requisitos não funcionais referem-se a qualidades ou restrições sobre o desenvolvimento ou provisão de um serviço (LAMSWEERDE, 2009; GUIZZARDI et al., 2014).

Essas duas dimensões proveem meios independentes de classificação de objetivos, ou seja, são classificações ortogonais. A partição de objetivos em *hardgoals-softgoals* é completamente diferente de sua classificação em objetivos funcionais e não funcionais. Na prática, contudo, certos objetivos de uma dimensão são mais frequentemente classificados como sendo de um tipo específico da outra. Por exemplo, frequentemente objetivos funcionais são *hardgoals* relacionados a alcançar uma situação. Por outro lado, objetivos relacionados à usabilidade e manutenibilidade tendem a ser *softgoals*.

5.1.3 - Modelagem de Objetivos

Existem várias linguagens/frameworks de modelagem de objetivos. Ainda que essas linguagens tenham diferenças, todas elas proveem elementos de modelagem para representar um conjunto de conceitos básicos: objetivos e relações entre eles (NEGRI et al., 2017).

Segundo Horkoff et at. (2016), *i** ou *iStar* (DALPIAZ et al., 2016) e KAOS (DARDENNE et al., 1993 apud HORKOFF et al. 2016) são as mais usadas em publicações relacionadas a GORE. Assim, *iStar* é a linguagem adotada nesta subseção e todo o texto que se segue é baseado no Guia da Linguagem *iStar* 2.0 (DALPIAZ et al., 2016).

Atores e Ligações entre Atores

Organizações são entidades sociais e sua operação depende da efetiva interação entre um número de atores. Um ator (*actor*) é uma entidade ativa e autônoma que visa alcançar seus objetivos exercendo seu know-how, em colaboração com outros atores. Em *i** 2.0, há dois subtipos de atores: Agente e Papel. Um agente (*agent*) é um ator concreto, com manifestação física, tal como uma pessoa, uma organização ou um departamento. Um papel (*role*) é uma caracterização abstrata de um comportamento de um ator social dentro de um contexto ou iniciativa de esforço específico. O Guia da Linguagem (DALPIAZ et al., 2016) sugere que as representações de ator, agente e papel (mostradas na Figura 5.3) sejam usadas da seguinte forma:

- Use agente quando for possível identificar um indivíduo concreto.
- Use papel quando se deseja caracterizar uma classe abstrata.
- Use ator nos demais casos.

Ator Papel Agente

Figura 5.3 – Representando Atores.

Há dois tipos de ligação entre atores: *is-a* e *participates-in*. *Is-a* representa a noção de generalização/especialização e pode ser aplicado entre papéis (Papel a Papel) ou entre atores (Ator a Ator). *Participates-in* representa qualquer outro tipo de associação entre dois atores diferente de *is-a*. *Participates-in* tem diferentes significados dependendo dos elementos sendo ligados:

- Agente - Papel: representa que o agente desempenha (*plays*) o papel.
- Ligando elementos do mesmo tipo: representa uma relação parte-de (*part-of*).

A Figura 5.4 ilustra a representação de ligações entre atores.

PhD student é um subtipo de *Student*.

Mike White desempenha o papel de *Phd student*.

Univ. trip mgmt IS é parte de *Univ. of Wonderland*.

Figura 5.4 – Representando Ligações entre Atores.

Elementos Intencionais

iStar 2.0 enfoca intenções, i.e., coisas que atores querem. Há quatro tipos de elementos intencionais (*intentional elements*), cujas representações são mostradas na Figura 5.5:

- Objetivo (*Goal*): é um estado de coisas (*state of affairs*) que um ator deseja alcançar e que tem um critério claro de satisfação. Ex.: Artigo publicado, voos reservados.
- Qualidade (*Quality*): é um atributo para o qual um ator deseja algum nível de satisfação. Sendo atributos, qualidades estão sempre relacionadas a uma entidade. Qualidades guiam a busca por meios de se alcançar objetivos. Ex.: Reserva (de uma viagem) rápida.
- Tarefa (*Task*): representa ações que um ator quer que sejam executadas, usualmente dentro do contexto de se alcançar um objetivo. Ex.: Pagar pelos bilhetes comprados.
- Recurso (*Resource*): é uma entidade física ou de informação requerida por um ator para realizar uma tarefa. Ex.: Cartão de Crédito.

Figura 5.5 – Representando Elementos Intencionais.

Elementos intencionais aparecem dentro da fronteira do ator (*actor boundary*), representando a perspectiva do ator no modelo, como ilustra a Figura 5.6.

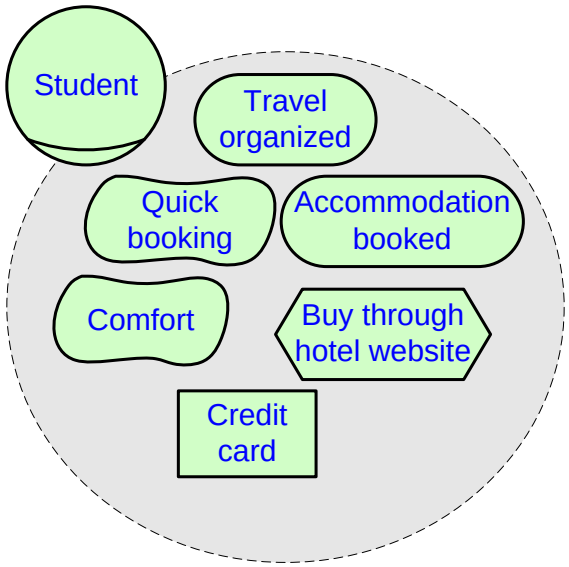


Figura 5.6 – Representando Elementos Intencionais dentro da Fronteira do Ator.

Dependências

Relações sociais são representadas como dependências. Uma dependência é um relacionamento com cinco argumentos, conforme abaixo. A Figura 5.7 mostra um exemplo de dependência.

- *Depender:* ator que depende de alguma coisa (o *Dependum*) a ser provida;
- *DependerElmt*: elemento intencional dentro da fronteira do *Depender* onde a dependência inicia e que explica porque a dependência existe;
- *Dependum*: elemento intencional que é o objeto da dependência;
- *Dependee*: o ator que deve prover o *dependum*;
- *DependeeElmt*: o elemento intencional que explica como o *dependee* pretende atender o *dependum*.

O tipo do *dependum* especializa a semântica da relação de dependência. Se o *dependum* é um objetivo, então o *dependee* é livre para escolher como atingir esse objetivo. Se o *dependum* é uma qualidade, o *dependee* é livre para escolher como satisfazer suficientemente a qualidade. Se o *dependum* é uma tarefa, espera-se que o *dependee* execute a tarefa da forma prescrita. Por fim, se o *dependum* é um recurso, espera-se que o *dependee* disponibilize o recurso para o *depender*. Assim, diferentes tipos de *dependum* dão ao *dependee* diferentes graus de liberdade.

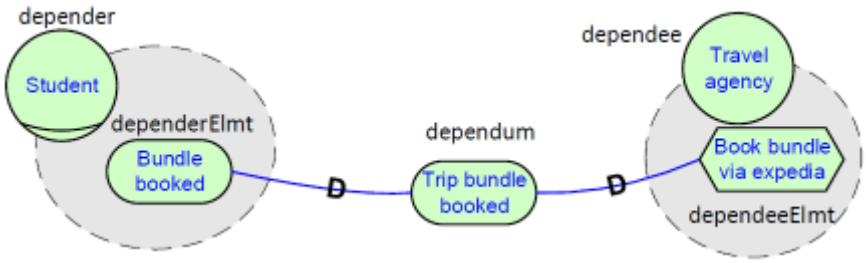


Figura 5.7 – Representando Dependências.

Nem todo os elementos de uma dependência precisam ser representados. P.ex., pode-se omitir o *dependerElmt*, o que implica em não especificar porque a dependência existe. Ao omitir o *dependeeElmt*, por sua vez, não se especifica como a dependência será realizada.

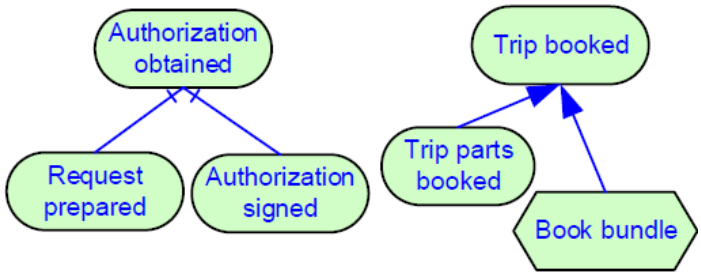
Ligações entre Elementos Intencionais

Os elementos intencionais dentro da fronteira de um ator são inter-relacionados. Em *iStar*, há quatro tipos de ligações entre elementos intencionais, conforme Tabela 5.1.

Tabela 5.1 – Possíveis Ligações entre Elementos Intencionais

		Arrowhead pointing to			
		Goal	*Quality*	*Task*	*Resource*
Link starts from	*Goal*	Refinement	Contribution	Refinement	n/a
	Quality	Qualification	Contribution	Qualification	Qualification
	Task	Refinement	Contribution	Refinement	n/a
	Resource	n/a	Contribution	NeededBy	n/a

Refinamento (*Refinement*) é um relacionamento genérico que liga objetivos e tarefas hierarquicamente. É um relacionamento n-ário ligando um pai a um ou mais filhos. Um elemento intencional só pode ser o pai em no máximo um link de refinamento. Há dois tipos de refinamento: *AND* – a realização de todos os *n* filhos (n≥2) torna o pai realizado; *Inclusive OR* – a realização de pelo menos um dos filhos torna o pai realizado. A Figura 5.8 mostra a notação usada para representar refinamentos.



AND-Refinement *OR-Refinement*

Figura 5.8 – Representando Refinamentos.

A relação Requerido por (*NeededBy*) liga uma tarefa a um recurso e indica que o ator precisa do recurso para executar a tarefa. Essa relação, contudo, não detalha a razão para tal necessidade (consumo, leitura etc.). A Figura 5.9 ilustra a notação de *iStar* para este tipo de ligação. Neste exemplo a tarefa "Pagar por bilhetes (*Pay for tickets*)" requer o recurso "Cartão de Crédito (*Credit Card*)".

A ligação de Qualificação (*Qualification*) relaciona uma qualidade a uma tarefa, objetivo ou recurso. A Figura 5.9 ilustra a notação de *iStar* para este tipo de ligação. Neste exemplo, "Sem erros (*No errors*)" refere-se à qualidade como o objetivo "Requisição preparada (*Request prepared*)" deve ser alcançado, i.e., a requisição deve ser preparada sem erros.

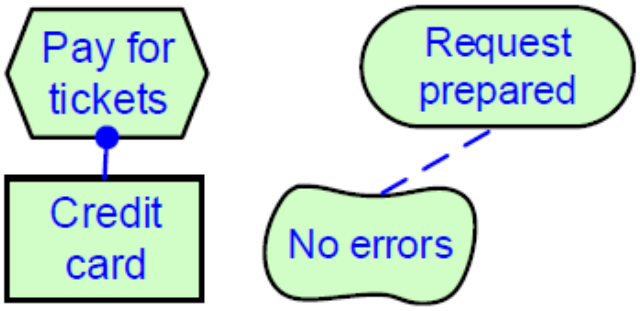


Figura 5.8 – Representando as Relações Requerido por e de Qualificação.

Por fim, ligações de contribuição (*contribution links*) representam os efeitos de elementos intencionais sobre qualidades. Qualidades podem ser cumpridas (ou satisfeitas), quando há evidências positivas suficiente, ou negadas, quando há evidências negativas. Sendo assim, essas ligações são qualitativas e são usadas para apoiar os analistas na tomada de decisão relativa a objetivos/tarefas alternativos. Note que as ligações de contribuição nada falam sobre como o cumprimento/negação é calculado.

Há quatro tipos de ligações de contribuição, expressando o que a fonte fornece, como ilustra a Figura 5.9.

- **Make**: há evidência positiva suficiente para satisfazer o elemento alvo;
- **Help**: há evidência positiva fraca para satisfazer o elemento alvo;
- **Hurt**: há evidência fraca contra a satisfação (ou seja, para a negação) do elemento alvo;
- **Break**: há evidência suficiente contra a satisfação (ou para a negação) do elemento alvo.

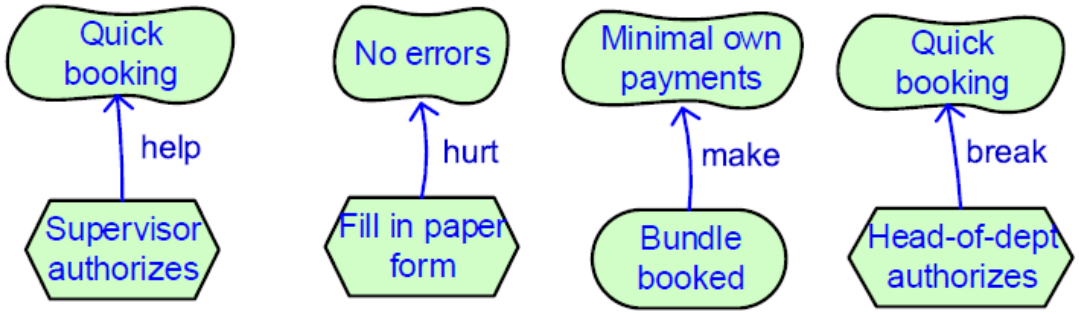


Figura 5.9 – Representando Ligações de Contribuição.

Visões de Modelagem

Quando usando *iStar*, o analista cria um modelo. Esse modelo pode ser visualizado via múltiplas perspectivas ou visões de modelo. As visões de modelo padrão de *i** são:

- Visão de Raciocínio estratégico (*Strategic Rationale* - SR): mostra todos os detalhes capturados no modelo: atores, ligações entre atores, elementos intencionais, dependências e ligações entre elementos intencionais.

- Visão de Dependência estratégica (*Strategic Dependency* - SD): mostra apenas a parte social do modelo, i.e., atores, ligações entre atores e dependências. Não mostra elementos intencionais e suas ligações.

Visões híbridas podem ser definidas. Por exemplo, algumas fronteiras de ator são abertas, mas não todas, omitindo, ainda, ligações entre atores.

5.2 – Modelagem de Casos de Uso

O propósito do modelo de casos de uso é capturar e descrever a funcionalidade que um sistema deve prover. Um sistema geralmente serve a vários atores, para os quais ele provê diferentes serviços. Tipicamente, a funcionalidade a ser provida por um sistema é muito grande para ser analisada como uma única unidade e, portanto, é importante ter um mecanismo de dividir essa funcionalidade em partes menores e mais gerenciáveis. O conceito de caso de uso é muito útil para esse propósito (OLIVÉ, 2007).

É importante ter em mente que modelos de casos de uso são fundamentalmente uma ferramenta textual. Ainda que casos de uso sejam também descritos graficamente (p.ex., fluxogramas ou algum diagrama da UML, dentre eles diagramas de casos de uso, diagramas de sequência e diagramas de atividades), não se deve perder de vista a natureza textual dos modelos de casos de uso. Olhando casos de uso apenas a partir da UML, que não trata do conteúdo ou da escrita de casos de uso, pode-se pensar, equivocadamente, que casos de uso são uma construção gráfica ao invés de textual. Em essência, casos de uso servem como um meio de comunicação entre pessoas, algumas delas sem nenhum treinamento especial e, portanto, o uso de texto para especificar casos de uso é geralmente a melhor escolha. Casos de uso são amplamente usados no desenvolvimento de sistemas, porque, por meio sobretudo de suas descrições textuais, usuários e clientes conseguem visualizar qual a funcionalidade a ser provida pelo sistema, conseguindo reagir mais rapidamente no sentido de refinar, alterar ou rejeitar as funções previstas para o sistema (COCKBURN, 2005). Assim, um modelo de casos de uso inclui duas partes principais: (i) os diagramas de casos de uso e (ii) as descrições de atores e de casos de uso, sendo que essas últimas podem ser complementadas com outros diagramas associados, tais como os diagramas de atividade e de sequência da UML[5].

Outro aspecto a ser realçado é que os modelos de caso de uso são independentes do método de análise a ser usado e até mesmo do paradigma de desenvolvimento. Assim, pode-se utilizar a modelagem de casos de uso tanto no contexto do desenvolvimento orientado a objetos (foco deste texto), como em projetos desenvolvidos segundo o paradigma estruturado. De fato, o uso de modelos de caso de uso pode ser ainda mais amplo. Casos de uso podem ser usados, por exemplo, para documentar processos de negócio de uma organização. Contudo, neste texto, explora-se a utilização de casos de uso para modelar e documentar requisitos funcionais de sistemas. Assim, geralmente são interessados[6] (*stakeholders*) nos casos de uso: as pessoas que usarão o sistema (usuários), o cliente que requer o sistema, outros sistemas com os quais o sistema em questão terá de interagir e outros membros da organização (ou até mesmo de fora dela) que têm restrições que o sistema precisa garantir.

Esta seção enfoca a modelagem de casos de uso e está estruturada conforme descrito a seguir. A Subseção 5.2.1 discute os dois principais conceitos empregados na modelagem de

[5] O uso de diagramas de atividade e de sequência para complementar a especificação de um caso de uso é discutido no Capítulo 7.
[6] Alguém ou algo com interesse no comportamento do sistema sob discussão (COCKBURN, 2005).

casos de uso: atores e casos de uso. A Subseção 5.2.2 aborda os diagramas de casos de uso e sua notação segundo a UML. A Subseção 5.2.3 trata da especificação de casos de uso. A Subseção 5.2.4 discute os tipos de relacionamentos que podem ser estabelecidos entre casos de uso, a saber: inclusão, extensão e generalização / especialização. Finalmente, a Subseção 5.2.5 discute como trabalhar com casos de uso e como usá-los em outras atividades do processo de software.

5.2.1 - Atores e Casos de Uso

Nenhum sistema computacional existe isoladamente. Todo sistema interage com atores humanos ou outros sistemas, que utilizam esse sistema para algum propósito e esperam que o sistema se comporte de certa maneira. Um caso de uso especifica um comportamento de um sistema segundo uma perspectiva externa e é uma descrição de uma sequência de ações realizada pelo sistema para produzir um resultado de valor para um ator (BOOCH; RUMBAUGH; JACOBSON, 2006).

Segundo Cockburn (2005), um caso de uso captura um contrato entre os interessados (*stakeholders*) em um sistema sobre o seu comportamento. Um caso de uso descreve o comportamento do sistema sob certas condições, em resposta a uma requisição feita por um interessado, dito o ator primário do caso de uso. Assim, os dois principais conceitos da modelagem de casos de uso são atores e casos de uso.

Atores

Dá-se nome de ator a um papel desempenhado por entidades físicas (pessoas ou outros sistemas) que interagem com o sistema em questão da mesma maneira, procurando atingir os mesmos objetivos. Uma mesma entidade física pode desempenhar diferentes papéis no mesmo sistema, bem como um dado papel pode ser desempenhado por diferentes entidades (OLIVÉ, 2007).

Atores são externos ao sistema. Um ator se comunica diretamente com o sistema, mas não é parte dele. A modelagem dos atores ajuda a definir as fronteiras do sistema, isto é, o conjunto de atores de um sistema delimita o ambiente externo desse sistema, representando o conjunto completo de entidades para as quais o sistema pode servir (BLAHA; RUMBAUGH, 2006; OLIVÉ, 2007).

Uma dúvida que sempre passa pela cabeça de um iniciante em modelagem de casos de uso é saber se o ator é a pessoa que efetivamente opera o sistema (p.ex., o atendente de uma locadora de automóveis) ou se é a pessoa interessada no resultado do processo (p.ex., o cliente que efetivamente loca o automóvel e é atendido pelo atendente). Essa definição depende, em essência, da fronteira estabelecida para o sistema. Sistemas de informação podem ter diferentes níveis de automatização. Por exemplo, se um sistema roda na Internet, seu nível de automatização é maior do que se ele requer um operador. Assim, é importante capturar qual o nível de automatização requerido e levar em conta o real limite do sistema (WAZLAWICK, 2004). Se o caso de uso roda na Internet (p.ex., um caso de uso de reserva de automóvel), então o cliente é o ator efetivamente. Se o caso de uso requer um operador (p.ex., um caso de uso de locação de automóvel, disponível apenas na locadora e para ser usado por atendentes), então o operador é o ator.

Quando se for considerar um sistema como sendo um ator, deve-se tomar o cuidado para não confundir a ideia de sistema externo (ator) com produtos usados na implementação do sistema em desenvolvimento. Para que um sistema possa ser considerado um ator, ele deve ser um sistema de informação completo (e não apenas uma biblioteca de classes, por exemplo). Além disso, ele deve estar fora do escopo do desenvolvimento do sistema atual. O analista não terá a oportunidade de alterar as funções do sistema externo, devendo adequar a comunicação às características do mesmo (WAZLAWICK, 2004).

Um ator primário é um ator que possui metas a serem cumpridas através do uso de serviços do sistema e que, tipicamente, inicia a interação com o sistema (OLIVÉ, 2007). Um ator secundário é um ator que interage com o sistema para prover um serviço para este último. A identificação de atores secundários é importante, uma vez que ela permite identificar interfaces externas que o sistema usará e os protocolos que regem as interações ocorrendo através delas (COCKBURN, 2005).

De maneira geral, o ator primário é o usuário direto do sistema ou outro sistema computacional que requisita um serviço do sistema em desenvolvimento. O sistema responde à requisição procurando atendê-la, ao mesmo tempo em que protege os interesses de todos os demais interessados no caso de uso. Entretanto, há situações em que o iniciador do caso de uso não é o ator primário. O tempo, por exemplo, pode ser o acionador de um caso de uso. Um caso de uso que roda todo dia à meia-noite ou ao final do mês tem o tempo como acionador. Mas o caso de uso ainda visa atingir um objetivo de um ator e esse ator é considerado o ator primário do caso de uso, ainda que ele não interaja efetivamente com o sistema (COCKBURN, 2005).

Para nomear atores, recomenda-se o uso de substantivos no singular, iniciados com letra maiúscula, possivelmente combinados com adjetivos. Exemplos: Cliente, Bibliotecário, Correntista, Correntista Titular etc.

Casos de Uso

Um caso de uso é uma porção coerente da funcionalidade que um sistema pode fornecer para atores interagindo com ele (BLAHA; RUMBAUGH, 2006). Um caso de uso corresponde a um conjunto de ações realizadas pelo sistema (ou por meio da interação com o sistema), que produz um resultado observável, com valor para um ou mais atores do sistema. Geralmente, esse valor é a realização de uma meta de negócio ou tarefa (OLIVÉ, 2007). Assim, um caso de uso captura alguma função visível ao ator e, em especial, busca atingir uma meta desse ator.

Deve-se considerar que um caso de uso corresponde a uma transação completa, ou seja, um usuário poderia ativar o sistema, executar o caso de uso e desativar o sistema logo em seguida, e a operação estaria completa e consistente e atenderia a uma meta desse usuário (WAZLAWICK, 2004).

Ser uma transação completa é uma característica essencial de um caso de uso[7], pois somente transações completas são capazes de atingir um objetivo do usuário. Casos de uso que necessitam de múltiplas sessões não passam nesse critério e devem ser divididos em casos de uso menores. Seja o exemplo de um caso de uso de concessão de empréstimo. Inicialmente, um atendente interagindo com um cliente informa os dados necessários para a avaliação do pedido de empréstimo. O pedido de empréstimo é, então, enviado para análise por um analista de

[7] Esta regra tem como exceção os casos de uso de inclusão e extensão, conforme discutido mais adiante na seção que trata de relacionamentos entre casos de uso.

crédito. Uma vez analisado e aprovado, o empréstimo é concedido, quando o dinheiro é entregue ao cliente e um contrato é assinado, dentre outros. Esse processo pode levar vários dias e não é realizado em uma sessão única. Assim, o caso de uso de concessão de empréstimo deveria ser subdividido em casos de uso menores, tais como casos de uso para efetuar pedido de empréstimo, analisar pedido de empréstimo e formalizar concessão de empréstimo.

Por outro lado, casos de uso muito pequenos, que não caracterizam uma transação completa, devem ser considerados passos de um caso de uso maior[8]. Seja o exemplo de uma biblioteca a qual cobra multa na devolução de livros em atraso. Um caso de uso específico para apenas calcular o valor da multa não é relevante, pois não caracteriza uma transação completa capaz de atingir um objetivo do usuário. O objetivo do usuário é efetuar a devolução e, neste contexto, uma regra de negócio (a que estabelece a multa) tem de ser levada em conta. Assim, calcular a multa é apenas um passo do caso de uso que efetua a devolução, o qual captura uma ação do sistema para garantir a regra de negócio e, portanto, satisfazer um interesse da biblioteca como organização.

Um caso de uso reúne todo o comportamento relevante de uma parte da funcionalidade do sistema. Isso inclui o comportamento principal normal, as variações de comportamento normais, as condições de exceção e o cancelamento de uma requisição. O conjunto de casos de uso captura a funcionalidade completa do sistema (BLAHA; RUMBAUGH, 2006).

Casos de uso fornecem uma abordagem para os desenvolvedores chegarem a uma compreensão comum com os usuários finais e especialistas do domínio, acerca da funcionalidade a ser provida pelo sistema (BOOCH; RUMBAUGH; JACOBSON, 2006).

Os objetivos dos atores são um bom ponto de partida para a identificação de casos de uso. Pode-se propor um caso de uso para satisfazer cada um dos objetivos de cada um dos atores. A partir desses objetivos, podem-se estudar as possíveis interações do ator com o sistema e refinar o modelo de casos de uso.

Cada caso de uso tem um nome. Esse nome deve capturar a essência do caso de uso. Para nomear casos de uso sugere-se usar frases iniciadas com verbos no infinitivo, seguidos de complementos, que representem a meta ou tarefa a ser realizada com o caso de uso. As primeiras letras (exceto preposições) de cada palavra devem ser grafadas em letra maiúscula. Exemplos: Cadastrar Cliente, Devolver Livro, Efetuar Pagamento de Fatura etc.

Um caso de uso pode ser visto como um tipo cujas instâncias são cenários. Um cenário é uma execução de um caso de uso com entidades físicas particulares desempenhando os papéis dos atores e em um particular estado do domínio de informação. Um cenário, portanto, exercita um certo caminho dentro do conjunto de ações de um caso de uso (OLIVÉ, 2007).

Alguns cenários mostram o objetivo do caso de uso sendo alcançado; outros terminam com o caso de uso sendo abandonado (COCKBURN, 2005). Mesmo quando o objetivo de um caso de uso é alcançado, ele pode ser atingido seguindo diferentes caminhos. Assim, um caso de uso deve comportar todas essas situações. Para tal, um caso de uso é normalmente descrito por um conjunto de fluxos de eventos, capturando o fluxo de eventos principal, i.e., o fluxo de eventos típico que conduz ao objetivo do caso de uso, e fluxos de eventos alternativos, descrevendo exceções ou variantes do fluxo principal.

[8] As mesmas exceções da nota anterior aplicam-se aqui, conforme discutido mais adiante.

5.2.2 - Diagrama de Casos de Uso

Basicamente, um diagrama de casos de uso mostra um conjunto de casos de uso e atores e seus relacionamentos, sendo utilizado para ilustrar uma visão estática das maneiras possíveis de se usar o sistema (BOOCH; RUMBAUGH; JACOBSON, 2006).

Os diagramas de casos de uso da UML podem conter os seguintes elementos de modelo, ilustrados na Figura 5.10 (BOOCH; RUMBAUGH; JACOBSON, 2006):

- Assunto: o assunto delimita a fronteira de um diagrama de casos de uso, sendo normalmente o sistema ou um subsistema. Os casos de uso de um assunto descrevem o comportamento completo do assunto. O assunto é exibido em um diagrama de casos de uso como um retângulo envolvendo os casos de uso que o compõem. O nome do assunto (sistema ou subsistema) pode ser mostrado dentro do retângulo.
- Ator: representa um conjunto coerente de papéis que os usuários ou outros sistemas desempenham quando interagem com os casos de uso. Tipicamente, um ator representa um papel que um ser humano, um dispositivo de hardware ou outro sistema desempenha com o sistema em questão. Atores não são parte do sistema. Eles residem fora do sistema. Atores são representados por um ícone de homem, com o nome colocado abaixo do ícone.
- Caso de Uso: representa uma funcionalidade que o sistema deve prover. Casos de uso são parte do sistema e, portanto, residem dentro dele. Um caso de uso é representado por uma elipse com o nome do caso de uso dentro ou abaixo dela.
- Relacionamentos de Dependência, Generalização e Associação: são usados para estabelecer relacionamentos entre atores, entre atores e casos de uso, e entre casos de uso.

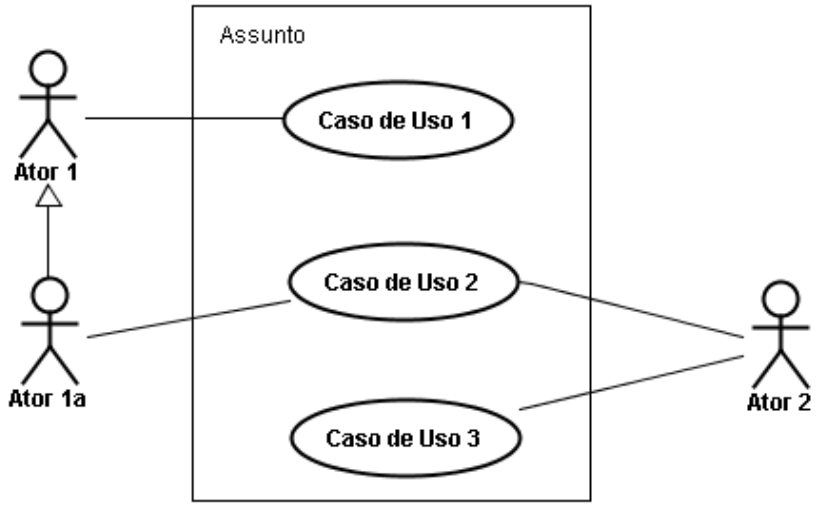


Figura 5.10 - Diagrama de Casos de Uso – Conceitos e Notação.

Atores só podem estar conectados a casos de uso por meio de associações. Uma associação entre um ator e um caso de uso significa que estímulos podem ser enviados entre atores e casos de uso. A associação entre um ator e um caso de uso indica que o ator e o caso de uso se comunicam entre si, cada um com a possibilidade de enviar e receber mensagens (BOOCH; RUMBAUGH; JACOBSON, 2006).

Atores podem ser organizados em hierarquias de generalização / especialização, de modo a capturar que um ator filho herda o significado e as associações com casos de uso de seu pai, especializando esse significado e potencialmente adicionando outras associações como outros casos de uso.

A Figura 5.11 mostra um diagrama de casos de uso para um sistema de caixa automático. Nesse diagrama, o assunto é o sistema como um todo. Os atores são: os clientes do banco, o sistema bancário e os responsáveis pela manutenção do numerário no caixa eletrônico. Cliente e mantenedor são atores primários, uma vez que têm objetivos a serem atingidos pelo uso do sistema. O sistema bancário é um ator secundário, pois o sistema do caixa automático precisa interagir com o sistema bancário para realizar os casos de uso *Efetuar Saque*, *Emitir Extrato* e *Efetuar Pagamento*.

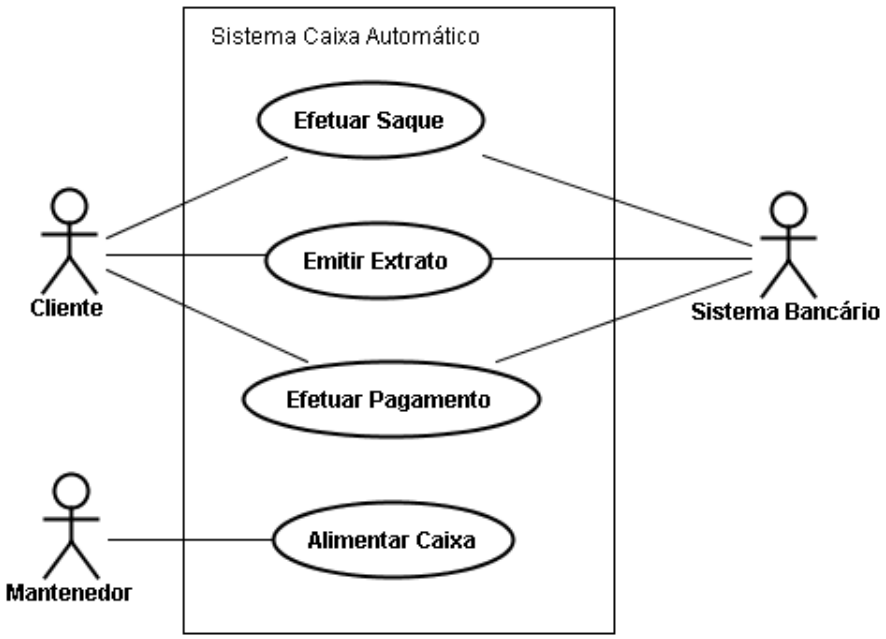


Figura 5.11 - Diagrama de Casos de Uso – Caixa Automático.

Um caso de uso descreve o que um sistema deve fazer. O diagrama de casos de uso provê uma visão apenas parcial disso, uma vez que mostra as funcionalidades por perspectiva externa. É necessário, ainda, capturar uma visão interna de cada caso de uso, especificando o comportamento do caso de uso pela descrição do fluxo de eventos que ocorre internamente (passos do caso de uso). Assim, uma parte fundamental do modelo de casos de uso é a descrição dos casos de uso.

5.2.3 - Descrevendo Casos de Uso

Um caso de uso deve descrever *o que* um sistema faz. Exceto para situações muito simples, um diagrama de casos de uso é insuficiente para este propósito. Assim, deve-se especificar o comportamento de um caso de uso pela descrição textual de seu fluxo de eventos, de modo que outros interessados possam compreendê-lo.

A especificação ou descrição de um caso de uso deve conter, dentre outras informações, um conjunto de sentenças, cada uma delas designando um passo simples, de modo que aprender

a ler um caso de uso não requeira mais do que uns poucos minutos. Dependendo da situação, diferentes estilos de escrita podem ser adotados (COCKBURN, 2005).

Cada passo do fluxo de eventos de um caso de uso tipicamente descreve uma das seguintes situações: (i) uma interação entre um ator e o sistema, (ii) uma ação que o sistema realiza para atingir o objetivo do ator primário ou (iii) uma ação que o sistema realiza para proteger os interesses de um interessado. Essas ações podem incluir validações e mudanças do estado interno do sistema (COCKBURN, 2005).

Não há um padrão definido para especificar casos de uso. Diferentes autores propõem diferentes estruturas, formatos e conteúdos para descrições de casos de uso, alguns mais indicados para casos de uso essenciais e mais complexos, outros para casos de uso cadastrais e mais simples. Mais além, pode ser útil utilizar mais de um formato dentro do mesmo projeto, em função das peculiaridades de cada caso de uso. De todo modo, é recomendável que a organização defina modelos de descrição de casos de uso a serem adotados em seus projetos, devendo definir tantos modelos quantos julgar necessários.

Cockburn (2005) recomenda que pelo menos dois modelos de descrição de casos de uso sejam definidos: um casual, escrito como um texto corrido livre, a ser usado em projetos com pouca formalidade; outro completo, com uma estrutura bem definida, para projetos de maior formalidade.

As seguintes informações são um bom ponto de partida para a definição de um modelo de descrição de casos de uso:

- Nome: nome do caso de uso, capturando a sua essência
- Escopo: diz respeito ao que está sendo documentado pelo caso de uso. Tipicamente pode ser um processo de negócio, um sistema ou um subsistema. Vale lembrar que este texto não aborda a utilização de casos de uso para a modelagem de processos de negócio. Assim, o escopo vai apontar o sistema / subsistema do qual o caso de uso faz parte.
- Descrição do Propósito: uma descrição sucinta do caso de uso, na forma de um único parágrafo, procurando descrever o objetivo do caso de uso.
- Ator Primário: nome do ator primário, ou seja, o interessado que tem um objetivo em relação ao sistema, o qual pode ser atingido pela execução do caso de uso.
- Interessados e Interesses: um interessado é alguém ou algo (um outro sistema) que tem um interesse no comportamento do caso de uso sendo descrito. Nesta seção são descritos cada um dos interessados no sistema e qual o seu interesse no caso de uso, incluindo o ator primário.
- Pré-condições: o que deve ser verdadeiro antes da execução do caso de uso. Se as pré-condições não forem satisfeitas, o caso de uso não pode ser realizado.
- Pós-condições: o que deve ser verdadeiro após a execução do caso de uso, considerando que o fluxo de eventos normal é realizado com sucesso.
- Fluxo de Eventos Normal: descreve os passos do caso de uso realizados em situações normais, considerando que nada acontece de errado e levando em conta a maneira mais comum do caso de uso ser realizado.
- Fluxo de Eventos Alternativos: descreve formas alternativas de realizar certos passos do caso de uso. Há duas formas alternativas principais: fluxos variantes, que são considerados dentro da normalidade do caso de uso; e fluxos de exceção, que se referem ao tratamento

de erros durante a execução de um passo do fluxo normal (ou de um fluxo variante ou até mesmo de um outro fluxo de exceção).

- Requisitos Relacionados: listagem dos identificadores dos requisitos (funcionais, não funcionais e regras de negócio) tratados pelo caso de uso sendo descrito, de modo a permitir rastrear os requisitos. Casos de uso podem ser usados para conectar vários requisitos, de tipos diferentes. Assim, essa listagem ajuda a manter um rastro entre requisitos funcionais, não funcionais e regras de negócio, além de permitir verificar se algum requisito deixou de ser tratado.
- Classes / Entidades: classes (no paradigma orientado a objetos) ou entidades (no paradigma estruturado) necessárias para tratar o caso de uso sendo descrito. Esta seção é normalmente preenchida durante a modelagem conceitual estrutural e é igualmente importante para permitir rastrear requisitos para as etapas subsequentes do desenvolvimento (projeto e implementação, sobretudo).

Descrevendo Fluxos de Eventos

Uma vez que o conjunto inicial de casos de uso estiver estabilizado, cada um deles deve ser descrito em mais detalhes. Primeiro, deve-se descrever o fluxo de eventos principal (ou curso básico), isto é, o curso de eventos mais importante, que normalmente ocorre. O fluxo de eventos normal (ou principal) é uma informação essencial na descrição de um caso de uso e não pode ser omitido em nenhuma circunstância. O fluxo de eventos normal é, portanto, a principal seção de uma descrição de caso de uso, a qual descreve o processo quando tudo dá certo, ou seja, sem a ocorrência de nenhuma exceção (WAZLAWICK, 2004).

Variantes do curso básico de eventos e tratamento de exceções que possam vir a ocorrer devem ser descritos em cursos alternativos. Normalmente, um caso de uso possui apenas um único curso básico, mas diversos cursos alternativos. Seja o exemplo de um sistema de caixa automático de banco, cujo diagrama de casos de uso é mostrado na Figura 5.11. O caso de uso *Efetuar Saque* poderia ser descrito como mostrado na Figura 5.12.

Como visto nesse exemplo, um caso de uso pode ter um número de cursos alternativos que podem levar o caso de uso por diferentes caminhos. Tanto quanto possível, esses cursos alternativos, muitos deles cursos de exceção, devem ser identificados durante a especificação do fluxo de eventos normal de um caso do uso.

Vale realçar que uma exceção não é necessariamente um evento que ocorre muito raramente, mas sim um evento capaz de impedir o prosseguimento do caso de uso, se não for devidamente tratado. Uma exceção também não é algo que impede o caso de uso de ser iniciado, mas algo que impede a sua conclusão. Condições que impedem um caso de uso de ser iniciado devem ser tratadas como pré-condições. As pré-condições nunca devem ser testadas durante o processo do caso de uso, pois, por definição, elas impedem que o caso de uso seja iniciado. Logo, seria inconsistente imaginar que elas pudessem ocorrer durante a execução do caso de uso. Se uma pré-condição é falsa, então o caso de uso não pode ser iniciado (WAZLAWICK, 2004).

Observa-se que a maioria das exceções ocorre nos passos em que alguma informação é passada dos atores para o sistema. Isso porque, quando uma informação é passada para o sistema, muitas vezes ele realiza validações. Quando uma dessas validações falha, tipicamente ocorre uma exceção (WAZLAWICK, 2004).

Nome: Efetuar Saque

Escopo: Sistema de Caixa Automático

Descrição do Propósito: Este caso de uso permite que um cliente do banco efetue um saque, retirando dinheiro de sua conta bancária.

Ator Primário: Cliente
Interessados e Interesses:

1. Cliente: deseja efetuar um saque.
2. Banco: garantir que apenas o próprio cliente efetuará saques e que os valores dos saques sejam compatíveis com o limite de crédito do cliente.

Pré-condições: O caixa automático deve estar conectado ao sistema bancário.

Pós-condições: O saque é efetuado, debitando o valor da conta do cliente e entregando o mesmo valor para o cliente em espécie.

Fluxo de Eventos Normal

O cliente insere seu cartão no caixa automático, que analisa o cartão e verifica se ele é aceitável. Se o cartão é aceitável, o caixa automático solicita que o cliente informe a senha. O cliente informa a senha. O caixa automático envia os dados do cartão e da senha para o sistema bancário para validação. Se a senha estiver correta, o caixa solicita que o cliente informe o tipo de transação a ser efetuada. O cliente seleciona a opção saque e o caixa solicita que seja informada a quantia. O cliente informa a quantia a ser sacada. O caixa envia uma requisição para o sistema bancário para que seja efetuado um saque na quantia especificada. Se o saque é autorizado, as notas são preparadas e liberadas.

Fluxos de Eventos de Exceção

- O cartão não é aceitável: Se o cartão não é aceitável, seja porque sua tarja magnética não é passível de leitura seja porque é de um tipo incompatível, uma mensagem de erro de leitura é mostrada.
- Senha incorreta: Se a senha informada está incorreta, uma mensagem é mostrada para o cliente que poderá entrar com a senha novamente. Caso o cliente informe três vezes senha incorreta, o cartão deverá ser bloqueado.
- Saque não autorizado: Se o saque não for aceito pelo sistema bancário, uma mensagem de erro é exibida e a operação é abortada.
- Não há dinheiro suficiente disponível no caixa eletrônico: Uma mensagem de erro é exibida e a operação é abortada.
- Cancelamento: O cliente pode cancelar a transação a qualquer momento, enquanto o saque não for autorizado pelo sistema bancário.

Requisitos Relacionados: RF01, RN01, RNF01, RNF02[9]

Classes: Cliente, Conta, Cartão, Transação, Saque.

Figura 5.12 – Descrição do Caso de Uso *Efetuar Saque*.

[9] São as seguintes as descrições dos requisitos listados: RF01 – O sistema de caixa automático deve permitir que clientes efetuem saques em dinheiro; RN01 – Não devem ser permitidas transações que deixem a conta do cliente com saldo inferior ao de seu limite de crédito; RNF01 – O sistema de caixa automático deve estar integrado ao sistema bancário; RNF02 – As operações realizadas no caixa automático devem dar respostas em até 10s a partir da entrada de dados.

Em sistemas de médio a grande porte, pode ser útil considerar a fusão de casos de uso fortemente relacionados em um único caso de uso, contendo mais de um fluxo de eventos normal. Em muitos sistemas é necessário dar ao usuário a possibilidade de cancelar ou alterar dados de uma transação efetuada anteriormente com sucesso. Se cada uma dessas possibilidades for considerada como um caso de uso isolado, o número de casos de uso pode crescer demasiadamente, aumentando desnecessariamente a complexidade do modelo de casos de uso. Além disso, o fluxo de eventos normal de um caso de uso desse tipo tende a ser muito simples, não justificando documentar todo um conjunto de informações para adicionar apenas duas ou três linhas descrevendo os passos do caso de uso. Assim, em situações dessa natureza, é interessante considerar apenas um caso de uso, contendo diversos fluxos de eventos principais. Essa abordagem é bastante recomendada para casos de uso cadastrais, em que um único caso de uso inclui fluxos de eventos normais para criar, alterar, consultar e excluir entidades.

Fluxos de eventos normais podem ser descritos de diferentes maneiras, dependendo do nível de formalidade que se deseja. Dentre os formatos possíveis, há dois principais:

- Livre: o fluxo de eventos normal é escrito na forma de um texto corrido, como no exemplo da Figura 5.12;
- Enumerado: cada passo do fluxo de eventos normal é numerado, de modo que possa ser referenciado nos fluxos de eventos alternativos ou em outros pontos do fluxo de eventos normal. A Figura 5.13 reapresenta o exemplo da Figura 5.12 neste formato. As seções iniciais foram omitidas por serem iguais às da Figura 5.12. Neste texto, advogamos em favor do uso do formato enumerado.

Cada exceção deve ser tratada por um fluxo alternativo de exceção. Fluxos alternativos de exceção devem ser descritos contendo as seguintes informações (WAZLAWICK, 2004): um identificador, uma descrição sucinta da exceção que ocorreu, os passos para tratar a exceção (ações corretivas) e uma indicação de como o caso de uso retorna ao fluxo principal (se for o caso) após a execução das ações corretivas.

Quando um formato de descrição enumerado é utilizado, não é necessário colocar uma verificação como uma condicional no fluxo principal. Por exemplo, no caso da Figura 5.13, o passo 3 não deve ser escrito como "3. Se o cartão é válido, o caixa automático solicita que o cliente informe a senha.". Basta o fluxo alternativo, no exemplo, o fluxo 2a.

Ainda quando o formato de descrição enumerado é utilizado, o identificador da exceção deve conter a linha do fluxo de eventos principal (ou eventualmente de algum outro fluxo de eventos alternativo) no qual a exceção ocorreu e uma letra para identificar a própria exceção (WAZLAWICK, 2004), como ilustra o exemplo da Figura 5.13.

Uma informação que precisa estar presente na descrição de um fluxo de eventos de exceção diz respeito a como finalizar o tratamento de uma exceção. Wazlawick (2004) aponta quatro formas básicas para finalizar o tratamento de uma exceção:

- Voltar ao início do caso de uso, o que não é muito comum nem prático.
- Voltar ao início do passo em que ocorreu a exceção e executá-lo novamente. Esta é a situação mais comum.
- Voltar para algum um passo posterior. Esta situação ocorre quando as ações corretivas realizam o trabalho que o passo (ou a sequência de passos) posterior deveria executar. Neste caso, é importante verificar se novas exceções não poderiam ocorrer
- Abortar o caso de uso. Neste caso, não se retorna ao fluxo principal e o caso de uso não atinge seus objetivos.

Nome: Efetuar Saque

Fluxo de Eventos Normal

1. O cliente insere seu cartão no caixa automático.
2. O caixa automático analisa o cartão e verifica se ele é aceitável.
3. O caixa automático solicita que o cliente informe a senha.
4. O cliente informa a senha.
5. O caixa automático envia os dados do cartão e da senha para o sistema bancário para validação.
6. O caixa automático solicita que o cliente informe o tipo de transação a ser efetuada.
7. O cliente seleciona a opção saque.
8. O caixa automático solicita que seja informada a quantia.
9. O cliente informa a quantia a ser sacada.
10. O caixa automático envia uma requisição para o sistema bancário para que seja efetuado um saque na quantia especificada.
11. As notas são preparadas e liberadas.

Fluxos de Eventos de Exceção

2a – O cartão não é aceitável: Se o cartão não é aceitável, seja porque sua tarja magnética não é passível de leitura seja porque é de um tipo incompatível, uma mensagem de erro de leitura é mostrada e se retorna ao passo 1.

5a – Senha incorreta:
5a.1 – 1ª e 2ª tentativas: Uma mensagem de erro é mostrada para o cliente. Retornar ao passo 3.
5a.2 – 3ª tentativa: bloquear o cartão e abortar a transação.

10a - Saque não autorizado: Uma mensagem de erro é exibida e a operação é abortada.

11a - Não há dinheiro suficiente disponível no caixa eletrônico: Uma mensagem de erro é exibida e a operação é abortada.

1 a 9: Cancelamento: O cliente pode cancelar a transação, enquanto o saque não for autorizado pelo sistema bancário. A transação é abortada.

Figura 5.13 – Descrição do Caso de Uso *Efetuar Saque* – Formato Enumerado

Além dos fluxos de exceção, há outro tipo de fluxo de eventos alternativo: os fluxos variantes. Fluxos variantes são considerados dentro da normalidade do caso de uso e indicam formas diferentes, mas igualmente normais, de se realizar uma certa porção de um caso de uso. Seja o caso de um sistema de um supermercado, mais especificamente um caso de uso para efetuar uma compra. Um passo importante desse caso de uso é a realização do pagamento, o qual pode se dar de três maneiras distintas: pagamento em dinheiro, pagamento em cheque, pagamento em cartão. Nenhuma dessas formas de pagamento constitui uma exceção. São todas maneiras diferentes, mas normais, de realizar um certo passo do caso de uso e, portanto, pode-se dizer que o fluxo principal possui três variações. A descrição de um fluxo variante deve conter: um identificador, uma descrição sucinta do passo especializado e os passos enumerados, como ilustra a Figura 5.14.

Nome: Efetuar Compra

Fluxo de Eventos Normal

...
1. De posse do valor a ser pago, o atendente informa a forma de pagamento.
2. Efetuar o pagamento:
    2a. Em dinheiro
    2b. Em cheque
    2c. Em cartão
3. O pagamento é registrado.

Fluxos de Eventos Variantes

2a – Pagamento em Dinheiro:
    2a.1 – O atendente informa a quantia em dinheiro entregue pelo cliente.
    2a.2 – O sistema informa o valor do troco a ser dado ao cliente.

2b – Pagamento em Cheque:
    2b.1 – O atendente informa os dados do cheque, a saber: banco, agência, conta e valor.

2c – Pagamento em Cartão:
    2c.1 – O atendente informa os dados do cartão e o valor da compra.
    2.c.2 – O sistema envia os dados informados no passo anterior, junto com a identificação da loja para o serviço de autorização do Sistema de Operadoras de Cartão de Crédito.
    2c.3 – O Sistema de Operadoras de Cartão de Crédito autoriza a compra e envia o código da autorização.

Figura 5.14 – Descrição Parcial do Caso de Uso *Efetuar Compra* – com Variantes

Por fim, em diversas situações, pode ser desnecessariamente trabalhoso especificar casos de uso segundo um formato completo, seja usando uma descrição dos fluxos de eventos no formato livre seja no formato enumerado. Para esses casos, um formato simplificado, na forma de uma tabela, pode ser usado. O formato tabular é normalmente empregado para casos de uso que possuem uma estrutura de interação simples, seguindo uma mesma estrutura geral, tais como casos de uso cadastrais (ou CRUD[10]) e consultas. Casos de uso cadastrais de baixa complexidade tipicamente envolvem inclusão, alteração, consulta e exclusão de entidades e seguem o padrão de descrição mostrado na Figura 5.15.

[10] CRUD – do inglês: Create, Read, Update and Delete; em português: Criar, Consultar, Atualizar e Excluir, ou seja, casos de uso que proveem as funções básicas de manipulação de dados de uma entidade de interesse do sistema.

Fluxos de Eventos Normais

Criar [Novo Objeto]
O [ator] informa os dados do [novo objeto], a saber: [atributos e associações do objeto]. Caso os dados sejam válidos, as informações são registradas.

Alterar Dados
O [ator] informa o [objeto] do qual deseja alterar dados e os novos dados. Os novos dados são validados e a alteração registrada.

Consultar Dados
O [ator] informa o [objeto] que deseja consultar. Os dados do [objeto] são apresentados.

Excluir [Objeto]
O [ator] informa o [objeto] que deseja excluir. Os dados do [objeto] são apresentados e é solicitada uma confirmação. Se a exclusão for confirmada, o [objeto] é excluído.

Fluxos de Eventos de Exceção

Incluir [Novo Objeto] / Alterar Dados
- Dados do [objeto] inválidos: uma mensagem de erro é exibida, solicitando correção da informação inválida

Figura 5.15 – Padrão Típico de Descrição de Casos de Uso Cadastrais.

Assim, para simplificar a descrição de casos de uso cadastrais, recomenda-se utilizar o modelo tabular mostrado na Tabela 5.2. Quando essa tabela for empregada, estar-se-á assumindo que o caso de uso envolve os fluxos de eventos indicados (I para inclusão, A para alteração, C para consulta e E para exclusão), com a descrição base mostrada na Figura 5.15.

Tabela 5.2 – Modelo de Descrição de Casos de Uso Cadastrais

Caso de Uso	Ações Possíveis	Observações	Requisitos	Classes
<nome do caso de uso>	< I, A, C, E >			

A coluna **Observações** é usada para listar informações importantes relacionadas às ações, tais como os itens informados na inclusão, uma restrição a ser considerada para que a exclusão possa ser feita, uma informação que não pode ser alterada ou uma informação do objeto que não é apresentada na consulta. Deve-se indicar antes da observação a qual ação ela se refere ([I] para inclusão, [A] para alteração, [C] para consulta e [E] para exclusão).

As colunas **Requisitos** e **Classes** indicam, respectivamente, os requisitos tratados pelo caso de uso e as classes do domínio do problema necessárias para a realização do caso de uso. O objetivo dessas colunas é manter a rastreabilidade dos casos de uso para requisitos e classes, respectivamente, de maneira análoga ao recomendado no formato completo. A Tabela 5.3 ilustra a descrição de dois casos de usos cadastrais do subsistema Controle de Frota de uma locadora de carros.

Tabela 5.3 – Descrição de Casos de Uso Cadastrais – Controle de Frota

Caso de Uso	Ações Possíveis	Observações	Requisitos	Classes
Cadastrar Modelo	I, A, C, E	[I] Informar: nome do modelo, marca e capacidade de passageiros. [E] Não é permitida a exclusão de modelos que tenham carros associados.	RF9	Modelo
Cadastrar Carro	I, A, C, E	[I] Informar: placa, número do chassis, número do renavam, cor, ano, modelo, potência do motor, número de portas, filial de lotação inicial, equipamentos e grupo a que pertence. [E] Não é permitido excluir um carro que tenha locações associadas.	RF8	Carro, Modelo, Filial, Equipamento, Grupo

Para casos de uso de consulta mais abrangente do que a consulta de um único objeto (já tratada como parte dos casos de uso cadastrais), mas ainda de baixa complexidade (tais como consultas que combinam informações de vários objetos envolvendo filtros), sugere-se utilizar o formato tabular mostrado na Tabela 5.4.

Tabela 5.4 – Modelo de Descrição de Casos de Uso de Consulta

Caso de Uso	Observações	Requisitos	Classes
<nome do caso de uso>			

A coluna **Observações** deve ser usada para listar informações importantes relacionadas à consulta, tais como dados que podem ser informados para a pesquisa, totalizações feitas em relatórios etc.

As colunas **Requisitos** e **Classes** têm a mesma função de suas homônimas no modelo da Tabela 5.2, ou seja, indicam, respectivamente, os requisitos tratados pelo caso de uso e as classes do domínio do problema necessárias para a realização do mesmo. A Tabela 5.5 ilustra a descrição de um caso de usos de consulta do subsistema Controle de Frota de uma locadora.

Tabela 5.5 – Descrição de Casos de Uso de Consulta – Controle de Frota

Caso de Uso	Observações	Requisitos	Classes
Consultar Frota	A consulta a carros que compõem a frota deve ser feita por filial. Assim, a filial deve ser sempre informada. As seguintes informações podem ser usadas (de maneira combinada) para refinar a busca: grupo, modelo, ano, número de portas, equipamentos e potência do motor.	RF15, RNF3	Carro, Modelo, Filial, Equipamento, Grupo

Descrevendo Informações Complementares

As descrições dos fluxos de eventos principal, variantes e de exceção são cruciais em uma descrição de casos de uso. Contudo, há outras informações complementares que são bastante úteis e, portanto, que devem ser levantadas e documentadas. Conforme listado no início desta seção, são informações complementares importantes: atores, interessados e interesses, pré-condições, pós-condições, requisitos relacionados e classes relacionadas.

Conforme discutido na Subseção 5.2.1, um ator representa um papel que entidades físicas ou sociais podem desempenhar na interação com o sistema. Essas entidades físicas são tipicamente pessoas, dispositivos ou outros sistemas que são externos ao sistema em desenvolvimento. Muitas vezes, apenas o nome de um ator, como mostrado em um diagrama de casos de uso, pode ser pouco para um real entendimento do que representa esse ator. Assim, é importante que uma descrição sucinta dos atores seja feita. Uma vez que um mesmo ator pode atuar em vários casos de uso, a descrição dos atores não deve ser feita dentro da descrição dos casos de uso, mas separada, como uma seção específica dentro do Documento de Especificação de Requisitos. Inicialmente, atores podem ser documentados em uma tabela de duas colunas, contendo o nome e a descrição do ator.

Como atores interagindo com o sistema definem as interfaces do sistema com o mundo externo, pode ser útil adicionar informações sobre o perfil do ator nessa interação. Quando o ator é um ator humano, esse perfil indicaria as habilidades e a experiência do ator, informações valiosas para o projeto da interface com o usuário. Adicionalmente, pode-se incluir uma classificação segundo aspectos como nível de habilidade, nível na organização e membros em diferentes grupos. Pressman (2006) propõe uma classificação que considera três grupos principais:

- *Usuário novato*: conhece pouco a interface para utilizá-la eficientemente (conhecimento sintático; p.ex., não sabe como atingir uma funcionalidade desejada) e entende pouco as funções e objetivos do sistema (conhece pouco a semântica da aplicação) ou não sabe bem como usar computadores em geral;
- *Usuário conhecedor e esporádico*: possui um conhecimento razoável da semântica da aplicação, mas tem relativamente pouca lembrança dos mecanismos de interação providos pela interface (informações sintáticas necessárias para utilizar a interface);
- *Usuário conhecedor e frequente*: possui bom conhecimento tanto sintático quanto semântico e buscam atalhos e modos abreviados de interação.

Não são apenas os atores os interessados em um caso de uso. Outras pessoas ou unidades de uma organização podem ter interesse nos resultados do caso de uso. Seja o caso de uma locadora de automóveis. Em um caso de uso de locação, o único papel a interagir com o sistema é o de funcionário do atendimento. Contudo, o cliente, o setor de preparação de automóveis, a contabilidade, dentre outros, são também interessados neste caso de uso. Assim, mesmo que essas pessoas não interajam diretamente com o sistema para a realização do caso de uso, elas devem ser listadas como interessados. Deve-se lembrar que o sistema deve satisfazer os interesses de todos os envolvidos, direta ou indiretamente. Assim, na seção "Interessados e Interesses", deve-se listar os diversos interessados e uma descrição sucinta de seus interesses em relação à execução do caso de uso. Ao analisar esses interesses é possível, dentre outros, capturar regras de negócio e informações e descobrir ações que o sistema tem de realizar para atender a essas expectativas, tais como validações, atualizações e registros. (WAZLAWICK, 2004; COCKBURN, 2005).

Pré-condições estabelecem o que precisa ser verdadeiro antes de se iniciar um caso de uso. Pós-condições, por sua vez, estabelecem o que será verdadeiro após a execução do caso de uso. Pré-condições precisam ser verdadeiras para que o caso de uso possa ser iniciado. Não se deve confundi-las com exceções. Pré-condições não são testadas durante a execução do caso de uso (como ocorre com as condições que geram exceções). Ao contrário, elas são testadas antes de iniciar o caso de uso. Se a pré-condição é falsa, então não é possível executar o caso de uso. Para documentar as pré-condições, recomenda-se listar as condições que têm de ser satisfeitas

na seção "Pré-condições". Pré-condições devem ser escritas como uma simples asserção sobre o estado do mundo no momento em que o caso de uso inicia (COCKBURN, 2005).

Muitas vezes, uma pré-condição para ser atendida requer que um outro caso de uso já executado tenha estabelecido essa pré-condição. Contudo, um erro bastante comum é escrever como uma pré-condição algo que frequentemente, mas não necessariamente, é verdadeiro (COCKBURN, 2005). Seja o caso de uma locadora de vídeos em que clientes em atraso não podem locar novos itens até que regularize suas pendências. Neste caso, uma pré-condição do tipo "cliente não está em atraso" como pré-condição de um caso de uso "efetuar locação" é inadequada. Observe que a identificação do cliente é parte do caso de uso efetuar locação e, portanto, não é possível garantir que o cliente não está em atraso antes de iniciar o caso de uso. Esta situação tem de ser tratada como uma exceção e não como uma pré-condição.

As seções de requisitos e classes relacionados são importantes para a gerência de requisitos. A primeira estabelece um rastro entre casos de uso e os requisitos de cliente documentados no Documento de Requisitos, permitindo, em um primeiro momento, analisar se algum requisito não foi tratado. Em um segundo momento, quando uma alteração em um requisito é solicitada, é possível usar essa informação para analisar o impacto da alteração. Para documentar os requisitos relacionados, recomenda-se listar os identificadores de cada um dos requisitos na seção de "Requisitos Relacionados".

A seção de classes relacionadas indica quais são as classes do modelo conceitual estrutural necessárias para a realização do caso de uso. Essa seção permite rastrear casos de uso para classes em vários níveis, uma vez que há uma grande tendência de as mesmas classes do modelo conceitual estrutural estarem presentes nos modelos de projeto e no código-fonte. Para documentar as classes relacionadas, recomenda-se listar o nome de cada uma das classes envolvidas na seção de "Classes Relacionadas". Vale ressaltar que essa informação é tipicamente preenchida durante a modelagem conceitual estrutural ou até mesmo depois, durante a elaboração de modelos de interação. A partir das informações de requisitos e classes relacionados, pode-se, por exemplo, construir matrizes de rastreabilidade.

5.2.4 - Relacionamentos entre Casos de Uso

Para permitir uma modelagem mais apurada dos casos de uso em um diagrama, três tipos de relacionamentos entre casos de uso podem ser empregados. Casos de uso podem ser descritos como versões especializadas de outros casos de uso (relacionamento de **generalização/ especialização**); casos de uso podem ser incluídos como parte de outro caso de uso (relacionamento de **inclusão**); ou casos de uso podem estender o comportamento de um outro caso de uso (relacionamento de **extensão**). O objetivo desses relacionamentos é tornar um modelo mais compreensível, evitar redundâncias entre casos de uso e permitir descrever casos de uso em camadas. A seguir esses tipos de relacionamentos são abordados.

Inclusão

Uma associação de inclusão de um *caso de uso base* para um *caso de uso de inclusão* significa que o comportamento definido no caso de uso de inclusão é incorporado ao comportamento do caso de uso base. Ou seja, a relação de inclusão incorpora um caso de uso (o caso de uso incluído) dentro da sequência de comportamento de outro caso de uso (o caso de uso base) (BLAHA; RUMBAUGH, 2006; OLIVÉ, 2007).

Esse tipo de associação é útil para extrair comportamento comum a vários casos de uso em uma única descrição, de modo que esse comportamento não tenha de ser descrito repetidamente. O caso de uso de inclusão pode ou não ser passível de utilização isoladamente. Assim, ele pode ser apenas um fragmento de uma funcionalidade, não precisando ser uma transação completa. A parte comum é incluída por todos os casos de uso base que têm esse caso de uso de inclusão em comum e a execução do caso de uso de inclusão é análoga a uma chamada de subrotina (OLIVÉ, 2007).

Na UML, o relacionamento de inclusão entre casos de uso é mostrado como uma dependência (seta pontilhada) estereotipada com a palavra-chave *include*, partindo do caso de uso base para o caso de uso de inclusão, como ilustra a Figura 5.16.

Figura 5.16 – Associação de Inclusão na UML

Uma associação de inclusão deve ser referenciada também na descrição do caso de uso base. O local em que esse comportamento é incluído deve ser indicado na descrição do caso de uso base, através de uma referência explícita à chamada ao caso de uso incluído. Assim, a descrição do fluxo de eventos (principal ou alternativo) do caso de uso base deve conter um passo que envolva a chamada ao caso de uso incluído, referenciada por "Incluir *nome do caso de uso incluído*". Para destacar referências de um caso de uso para outro, sugere-se que o nome do caso de uso referenciado seja sublinhado e escrito em itálico.

No exemplo do caixa automático, todos os três casos de uso têm em comum uma porção que diz respeito à validação inicial do cartão. Neste caso, um relacionamento de inclusão deve ser empregado, conforme mostra a Figura 5.17.

O caso de uso *Validar Cartão* extrai o comportamento descrito na Figura 5.18. Ao isolar este comportamento no caso de uso de *Validar Cliente*, o caso de uso *Efetuar Saque* passaria a apresentar a descrição mostrada na Figura 5.19.

Deve-se observar que não necessariamente o comportamento do caso de uso incluído precisa ser executado todas as vezes que o caso de uso base é realizado. Assim, é possível que a inclusão esteja associada a alguma condição. O caso de uso incluído é inserido em um local específico dentro da sequência do caso de uso base, da mesma forma que uma subrotina é chamada de um local específico dentro de outra subrotina (BLAHA; RUMBAUGH, 2006).

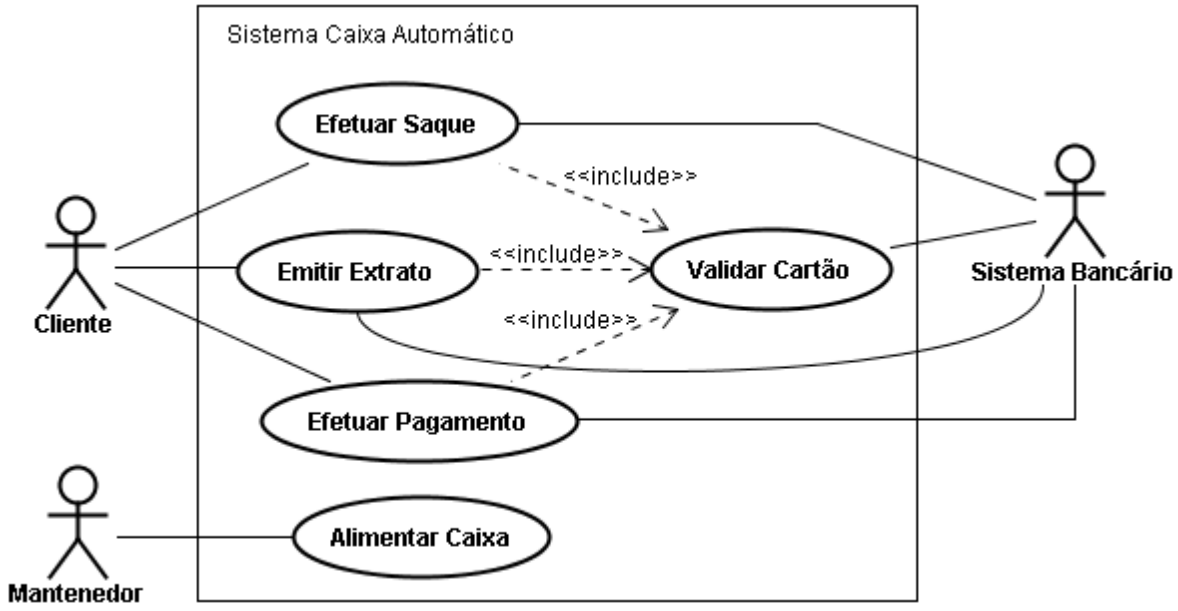


Figura 5.17 - Diagrama de Casos de Uso – Caixa Automático com Inclusão.

Nome: Validar Cartão

Fluxo de Eventos Normal

1. O cliente insere o cartão no caixa automático.
2. O caixa automático analisa o cartão e verifica se ele é aceitável.
3. O caixa automático solicita que o cliente informe a senha.
4. O cliente informa a senha.
5. O caixa automático envia os dados do cartão e da senha para o sistema bancário para validação.
6. O caixa automático solicita que o cliente informe o tipo de transação a ser efetuada.

Fluxos de Eventos de Exceção

2a – O cartão não é aceitável: Se o cartão não é aceitável, seja porque sua tarja magnética não é passível de leitura seja porque é de um tipo incompatível, uma mensagem de erro de leitura é mostrada e se retorna ao passo 1.

5a – Senha incorreta:

5a.1 – 1ª e 2ª tentativas: Uma mensagem de erro é mostrada para o cliente. Retornar ao passo 3.

5a.2 – 3ª tentativa: bloquear o cartão e abortar a transação.

1 a 5: Cancelamento: O cliente solicita o cancelamento da transação e a transação é abortada.

Figura 5.18 – Descrição do Caso de Uso *Validar Cartão*

Nome: Efetuar Saque

Fluxo de Eventos Normal

1. Incluir *Validar Cartão*.
2. O cliente seleciona a opção saque.
3. O caixa automático solicita que seja informada a quantia.
4. O cliente informa a quantia a ser sacada.
5. O caixa automático envia uma requisição para o sistema bancário para que seja efetuado um saque na quantia especificada.
6. As notas são preparadas e liberadas.

Fluxos de Eventos de Exceção

5a - Saque não autorizado: Uma mensagem de erro é exibida e a operação é abortada.

6a - Não há dinheiro suficiente disponível no caixa eletrônico: Uma mensagem de erro é exibida e a operação é abortada.

1 a 3: Cancelamento: O cliente pode cancelar a transação, enquanto o saque não for autorizado pelo sistema bancário. A transação é abortada.

Figura 5.19 – Descrição do Caso de Uso *Efetuar Saque* com inclusão.

Por fim, é importante frisar que não há um consenso sobre a possibilidade (ou não) de um caso de uso incluído poder ser utilizado isoladamente. Diversos autores, dentre eles Olivé (2007) e Blaha e Rumbaugh (2006), admitem essa possibilidade; outros não. Em (BOOCH; RUMBAUGH; JACOBSON, 2006), diz-se explicitamente que um "caso de uso incluído nunca permanece isolado, mas é apenas instanciado como parte de alguma base maior que o inclui". Neste texto, admitimos a possibilidade de um caso de uso incluído poder ser utilizado isoladamente, pois isso permite representar situações em que um caso de uso chama outro caso de uso (como uma chamada de subrotina), mas este último pode também ser realizado isoladamente. A Figura 5.20 ilustra uma situação bastante comum, em que, ao se realizar um processo de negócio (no caso a reserva de um carro em um sistema de locação de automóveis), caso uma informação necessária para esse processo (no caso o cliente) não esteja disponível, ela pode ser inserida no sistema. Contudo, o cadastro da informação também pode ser feito dissociado do processo de negócio que o inclui (no caso, o cliente pode se cadastrar fora do contexto da reserva de um carro). Ao não se admitir a possibilidade de um caso de uso incluído poder ser utilizado isoladamente, não é possível modelar situações desta natureza, as quais são bastante frequentes.

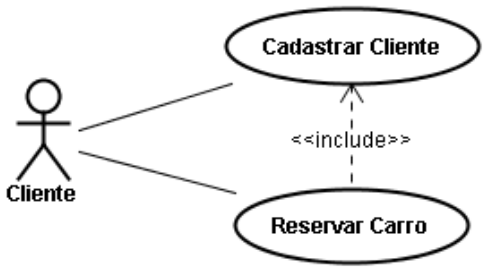


Figura 5.20 – Exemplo de Associação de Inclusão.

Extensão

Uma associação de extensão entre um *caso de uso de extensão* e um *caso de uso base* significa que o comportamento definido no caso de uso de extensão pode ser inserido dentro do comportamento definido no caso de uso base, em um local especificado indiretamente pelo caso de uso de extensão. A extensão ocorre em um ou mais pontos de extensão específicos definidos no caso de uso base. A extensão pode ser condicional. Neste caso, a extensão ocorre apenas se a condição é verdadeira quando o ponto de extensão especificado é atingido. O caso de uso base é definido de forma independente da extensão e é significativo independentemente do caso de uso de extensão (OLIVÉ, 2007; BOOCH; RUMBAUGH; JACOBSON, 2006).

Um caso de uso pode ter vários pontos de extensão e esses pontos são referenciados por seus nomes (BOOCH; RUMBAUGH; JACOBSON, 2006). O caso de uso base apenas indica seus pontos de extensão. O caso de uso de extensão especifica em qual ponto de extensão ele será inserido. Por isso, diz-se que o caso de uso de extensão especifica indiretamente o local onde seu comportamento será inserido.

A associação de extensão é como uma relação de inclusão olhada da direção oposta, em que a extensão se incorpora ao caso de uso base, em vez de o caso de uso base incorporar explicitamente a extensão. Ela conecta um caso de uso de extensão a um caso de uso base. O caso de uso de extensão é geralmente um fragmento, ou seja, ele não aparece sozinho como uma sequência de comportamentos. Além disso, na maioria das vezes, a relação de extensão possui uma condição associada e, neste caso, o comportamento de extensão ocorre apenas se a condição for verdadeira. O caso de uso base, por sua vez, precisa ser, obrigatoriamente, um caso de uso válido na ausência de quaisquer extensões (BLAHA; RUMBAUGH, 2006). Uma importante diferença entre as associações de inclusão e extensão é que, na primeira o caso de uso base está ciente do caso de uso de inclusão, enquanto na segunda o caso de uso base não está ciente dos possíveis casos de uso de extensão (OLIVÉ, 2007).

Na UML, a associação de extensão é mostrada como uma dependência (seta pontilhada) estereotipada com a palavra chave *extend*, partindo do caso de uso de extensão para o caso de uso base, como ilustra a Figura 5.21. Pontos de extensão podem ser indicados no compartimento da elipse do caso de uso, denominado "*extension points*" (pontos de extensão). Opcionalmente, a condição a ser satisfeita e a referência ao ponto de extensão podem ser mostradas por meio de uma nota[11] anexada à associação de extensão (OLIVÉ, 2007). Assim, no exemplo da Figura 5.21, o *Caso de Uso de Extensão 1* é executado quando o *ponto de extensão 1* do *Caso de Uso Base* for atingido, se a *condição* for verdadeira.

Assim como no caso da inclusão, uma associação de extensão deve ser referenciada na descrição do caso de uso base. Neste caso, contudo, o caso de uso base apenas aponta o ponto de extensão, sem fazer uma referência explícita ao caso de uso de extensão. O local de cada um dos pontos de extensão deve ser indicado na descrição do caso de uso base, através de uma referência ao nome do ponto de extensão seguido de ": ponto de extensão". Assim, a descrição

[11] Nota é o único item de anotação da UML. Notas são usadas para explicar partes de um modelo da UML. São comentários incluídos para descrever, esclarecer ou fazer alguma observação sobre qualquer elemento do modelo. Assim, uma nota é apenas um símbolo para representar restrições e comentários anexados a um elemento ou a uma coleção de elementos. Graficamente, uma nota é representada por um retângulo com um dos cantos com uma dobra de página, acompanhado por texto, e anexada ao(s) elemento(s) anotados por meio de linha(s) pontilhada(s) (BOOCH; RUMBAUGH; JACOBSON, 2006). No exemplo da Figura 5.21, a nota está anexada ao relacionamento de extensão, adicionando-lhe informações sobre o ponto de extensão e a condição associados à extensão.

do fluxo de eventos (principal ou alternativo) do caso de uso base deve conter indicações explícitas para cada ponto de extensão.

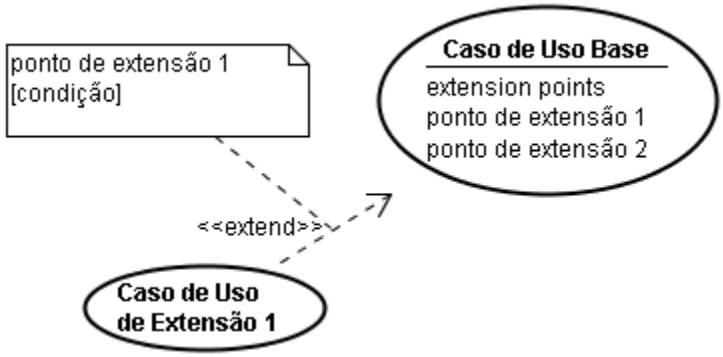


Figura 5.21 – Associação de Extensão na UML

No exemplo do caixa automático, suponha que se deseja coletar dados estatísticos sobre os valores das notas entregues nos saques, de modo a permitir alimentar o caixa eletrônico com as notas mais adequadas para saque. Poder-se-ia, então, estender o caso de uso *Efetuar Saque*, de modo que, quando necessário, outro caso de uso, denominado *Coletar Estatísticas de Notas*, contasse e acumulasse o tipo das notas entregues em um saque, conforme mostra a Figura 5.22. A Figura 5.23 mostra a descrição do caso de uso *Efetuar Saque* indicando o ponto de extensão *entrega do dinheiro*.

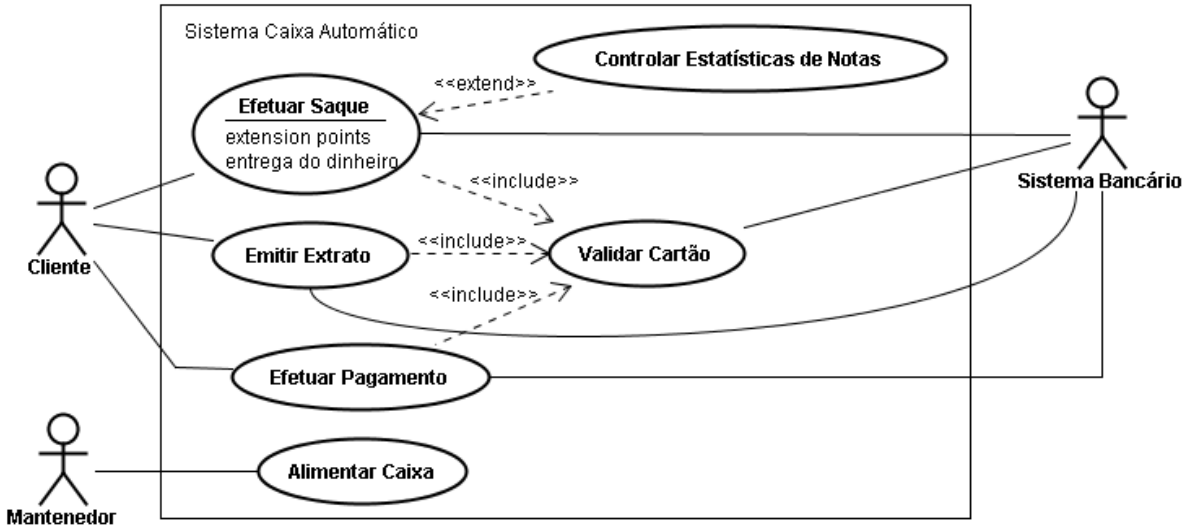


Figura 5.12 - Diagrama de Casos de Uso – Caixa Automático com Extensão.

Nome: Efetuar Saque

Fluxo de Eventos Normal

1. Incluir *Validar Cartão*.
2. O cliente seleciona a opção saque.
3. O caixa automático solicita que seja informada a quantia.
4. O cliente informa a quantia a ser sacada.
5. O caixa automático envia uma requisição para o sistema bancário para que seja efetuado um saque na quantia especificada.
6. As notas são preparadas.

entrega do dinheiro: ponto de extensão.

7. As notas são liberadas

Fluxos de Eventos de Exceção

5a - Saque não autorizado: Uma mensagem de erro é exibida e a operação é abortada.

6a - Não há dinheiro suficiente disponível no caixa eletrônico: Uma mensagem de erro é exibida e a operação é abortada.

1 a 3: Cancelamento: O cliente pode cancelar a transação, enquanto o saque não for autorizado pelo sistema bancário. A transação é abortada.

Figura 5.23 – Descrição do Caso de Uso *Efetuar Saque* com extensão.

Generalização / Especialização

Um relacionamento de generalização / especialização entre um *caso de uso pai* e um *caso de uso filho* significa que o caso de uso filho herda o comportamento e o significado do caso de uso pai, acrescentando ou sobrescrevendo seu comportamento (OLIVÉ, 2007; BOOCH; RUMBAUGH; JACOBSON, 2006). Na UML, relacionamentos de generalização / especialização são representados como uma linha cheia direcionada com uma seta aberta (símbolo de herança), como ilustra a Figura 5.24.

Figura 5.24 – Associação de Generalização / Especialização entre Casos de Uso na UML

Voltando ao exemplo do sistema de caixa automático, suponha que haja duas formas adotadas para se validar o cartão: a primeira através de senha, como descrito anteriormente, e a segunda por meio de análise da retina do cliente. Neste caso, poderiam ser criadas duas especializações do caso de uso *Validar Cliente*, como mostra a Figura 5.25.

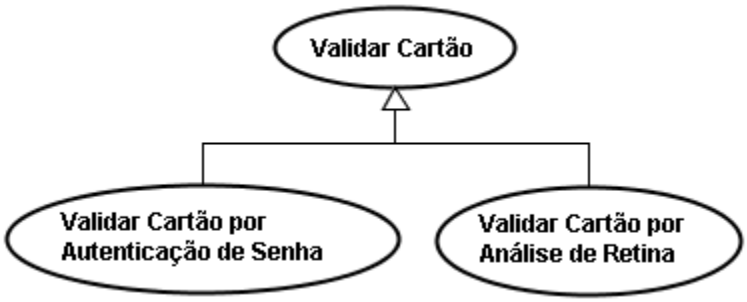


Figura 5.25 – Exemplo de Generalização / Especialização entre Casos de Uso

A descrição do caso de uso pai teria de ser generalizada para acomodar diferentes tipos de validação. Esses tipos de validação seriam especializados nas descrições dos casos de uso filhos. A Figura 5.26 mostra as descrições desses três casos de uso.

A generalização / especialização é aplicável quando um caso de uso possui diversas variações. O comportamento comum pode ser modelado como um caso de uso abstrato e especializado para as diferentes variações (BLAHA; RUMBAUGH, 2006). Contudo, avalie se não fica mais simples e direto descrever essas variações como fluxos alternativos variantes na descrição de casos de uso. Quando forem poucas e pequenas as variações, muito provavelmente será mais fácil capturá-las na descrição, ao invés de criar hierarquias de casos de uso. A Figura 5.27 mostra uma solução análoga à da Figura 5.26, sem usar, no entanto, especializações do caso de uso.

Diretrizes para o Uso dos Tipos de Relacionamentos entre Casos de Uso

Os relacionamentos entre casos de uso devem ser utilizados com cuidado para evitar a introdução de complexidade desnecessária. As seguintes orientações são úteis para ajudar a decidir quando usar relacionamentos entre casos de uso em um diagrama de casos de uso:

- A inclusão é tipicamente aplicável quando se deseja capturar um fragmento de comportamento comum a vários casos de uso. Na maioria das vezes, o caso de uso de inclusão é uma atividade significativa, mas não como um fim em si mesma (BLAHA; RUMBAUGH, 2006). Ou seja, o caso de uso de inclusão não precisa ser uma transação completa. Um relacionamento de inclusão é empregado quando há uma porção de comportamento que é similar ao longo de um ou mais casos de uso e não se deseja repetir a sua descrição. Para evitar redundância e assegurar reúso, extrai-se essa descrição e se compartilha a mesma entre diferentes casos de uso. Desta maneira, utiliza-se a inclusão para evitar ter de descrever o mesmo fragmento de comportamento várias vezes, capturando o comportamento comum em um caso de uso próprio (BOOCH; RUMBAUGH; JACOBSON, 2006).
- Não se deve utilizar o relacionamento de generalização/especialização para compartilhar fragmentos de comportamento. Para este propósito, deve-se usar a relação de inclusão (BLAHA; RUMBAUGH, 2006).
- A relação de extensão é bastante útil em situações em que se pode definir um caso de uso significativo com recursos adicionais. O comportamento básico é capturado no caso de uso base e os recursos adicionais nos casos de uso de extensão. Use a relação de extensão quando o sistema puder ser usado em diferentes configurações, algumas com os recursos adicionais e outras sem eles (BLAHA; RUMBAUGH, 2006).

Nome: Validar Cartão

Fluxo de Eventos Normal

3. O cliente insere o cartão no caixa automático.
4. O caixa automático analisa o cartão e verifica se ele é aceitável.
5. O caixa automático solicita informação para identificação do cliente.
6. O cliente informa sua identificação.
7. O caixa automático envia os dados do cartão e da identificação para o sistema bancário para validação.
8. O caixa automático solicita que o cliente informe o tipo de transação a ser efetuada.

Fluxos de Eventos de Exceção

2a – O cartão não é aceitável: Se o cartão não é aceitável, seja porque sua tarja magnética não é passível de leitura seja porque é de um tipo incompatível, uma mensagem de erro de leitura é mostrada e se retorna ao passo 1.

5a – Dados de Identificação Incorretos:

5a.1 – 1ª e 2ª tentativas: Uma mensagem de erro é mostrada para o cliente. Retornar ao passo 3.

5a.2 – 3ª tentativa: bloquear o cartão e abortar a transação.

1 a 5: Cancelamento: O cliente solicita o cancelamento da transação e a transação é abortada.

Nome: Validar Cartão por Análise de Retina

Fluxo de Eventos Normal

3. O caixa automático solicita que o cliente se posicione corretamente para a captura da imagem da retina.
4. O caixa automático retira uma foto da retina do cliente.
5. O caixa automático envia os dados do cartão e a foto da retina para o sistema bancário para validação.

Nome: Validar Cartão por Autenticação de Senha

Fluxo de Eventos Normal

1. O caixa automático solicita a senha.
2. O cliente informa a senha.
3. O caixa automático envia os dados do cartão e a senha para o sistema bancário para validação.

Figura 5.26 – Descrição do Caso de Uso *Validar Cartão* e suas Especializações.

- Tanto a inclusão quanto a extensão podem ser usadas para dividir o comportamento em partes menores. A inclusão, entretanto, implica que o comportamento incluído é uma parte necessária de um sistema configurado, mesmo que seu comportamento não seja executado todas as vezes, ou seja, mesmo que o comportamento incluído esteja

associado a uma condição. A extensão, por sua vez, implica que o sistema sem o comportamento adicionado pela extensão é significativo (BLAHA; RUMBAUGH, 2006).

Nome: Validar Cartão

Fluxo de Eventos Normal

- O cliente insere o cartão no caixa automático.
- O caixa automático analisa o cartão e verifica se ele é aceitável.
- Validar cartão.
- O caixa automático solicita que o cliente informe o tipo de transação a ser efetuada.

Fluxos de Eventos Variantes

3a – Validar cartão por autenticação de senha:

3a.1 – O caixa automático solicita a senha.

3a.2 – O cliente informa a senha.

3a.3 – O caixa automático envia os dados do cartão e a senha para o sistema bancário para validação.

3b – Validar cartão por análise de retina:

3b.1 – O caixa automático solicita que o cliente se posicione corretamente para a captura da imagem da retina.

3b.2 – O caixa automático retira uma foto da retina do cliente.

3b.3 – O caixa automático envia os dados do cartão e a foto da retina para o sistema bancário para validação.

Fluxos de Eventos de Exceção

2a – O cartão não é aceitável: Se o cartão não é aceitável, seja porque sua tarja magnética não é passível de leitura seja porque é de um tipo incompatível, uma mensagem de erro de leitura é mostrada e se retorna ao passo 1.

5a – Dados de Identificação Incorretos:

5a.1 – 1ª e 2ª tentativas: Uma mensagem de erro é mostrada para o cliente. Retornar ao passo 3.

5a.2 – 3ª tentativa: bloquear o cartão e abortar a transação.

1 a 5: Cancelamento: O cliente solicita o cancelamento da transação e a transação é abortada.

Figura 5.27 – Descrição do Caso de Uso *Validar Cartão* com Variantes.

5.2.5 – Trabalhando com Casos de Uso

Para se utilizar a modelagem de casos de uso para o refinamento de requisitos de cliente em requisitos de sistema é necessário proceder um exame detalhado do processo de negócio a ser apoiado pelo sistema. Assim, atividades de levantamento de requisitos, como entrevistas, observação, workshop de requisitos e cenários, dentre outras, certamente acontecerão em paralelo com a modelagem de casos de uso.

Uma boa maneira de trabalhar com casos de uso consiste em, a partir dos requisitos funcionais de usuário descritos no Documento de Definição de Requisitos, procurar derivar casos de uso. Este é apenas um ponto de partida, uma vez que vários casos de uso podem ser derivados a partir de um mesmo requisito funcional de usuário.

Uma maneira complementar de identificar casos de uso é começar pela identificação de atores. Cada ator deve ter um propósito único e coerente, o qual deve ser descrito e documentado. Para cada ator identificado, pode-se, então, levantar quais são as funcionalidades por ele requeridas, listando-as na forma de casos de uso. As funcionalidades devem ser extraídas do Documento de Definição de Requisitos. Cada caso de uso deve representar uma transação completa que seja algo de valor para os atores envolvidos. Contudo, antes de identificar atores e casos de uso, é necessário determinar claramente os limites do sistema. Sem deixar claro quais são os limites do sistema, é muito difícil identificar atores ou casos de uso (BLAHA; RUMBAUGH, 2006).

Uma vez identificados atores e casos de uso, pode-se elaborar uma versão preliminar do diagrama de casos de uso. Vale lembrar que, até mesmo para sistemas de pequeno porte, é útil trabalhar com subsistemas, procurando agrupar casos de uso em pacotes. Assim, é importante construir também diagramas de pacotes à medida que os casos de uso vão sendo agrupados.

Uma vez identificados e agrupados os casos de uso, é interessante fazer uma descrição sucinta de seu propósito. Não se deve partir diretamente para os detalhes, descrevendo fluxos de eventos e outras informações. Fazendo apenas uma descrição sucinta, é possível levar mais rapidamente os casos de uso à discussão com os clientes e usuários, permitindo identificar melhor quais são efetivamente os casos de uso a serem contemplados pelo sistema. Além disso, pode-se dividir o trabalho, designando diferentes analistas para trabalhar com casos de uso (ou pacotes) específicos.

Somente então se deve passar para a descrição detalhada dos casos de uso. Inicialmente, o foco deve ser no fluxo de eventos principal, ou seja, aquele em que tudo dá certo na interação. Depois de descrever o fluxo de eventos normal, deve-se analisar de forma crítica cada passo desses fluxos de eventos, procurando verificar o que pode dar errado (WAZLAWICK, 2004), bem como se devem investigar maneiras alternativas, ainda normais, de realizar o caso de uso, permitindo a identificação de fluxos variantes. A partir da identificação de possíveis exceções e variações, deve-se trabalhar na descrição de fluxos alternativos de exceção (descrevendo procedimentos para contornar os problemas) e variantes (descrevendo maneiras alternativas de realizar com sucesso uma certa porção do caso de uso).

Assim, uma maneira adequada para trabalhar com casos de uso consiste em identificá-los, modelá-los e descrevê-los com diferentes níveis de precisão. O seguinte processo resume a abordagem descrita anteriormente:

- Listar atores e casos de uso relacionados: neste momento, é montada apenas uma lista dos atores associados aos casos de uso de seu interesse. Apenas o nome do caso de uso é indicado.

- Para cada caso de uso identificado, fazer uma descrição sucinta do mesmo. Essa descrição deve conter, em essência, o objetivo do caso de uso.
- Elaborar um ou mais diagramas de casos de uso.
- Revisar a exatidão e a completude do conjunto de casos de uso com os interessados e priorizar os casos de uso.
- Definir o formato de descrição de caso de uso a ser usado (e o correspondente modelo de descrição de caso de uso a ser adotado) para cada caso de uso.
- Definir os fluxos de eventos principais a serem comportados pelo caso de uso: de maneira análoga ao passo 1, apenas uma lista dos fluxos de eventos principais é elaborada, sem descrevê-los ainda.
- Descrever cada um dos fluxos principais de eventos do caso de uso, segundo o modelo de descrição de caso de uso estabelecido no passo anterior. De acordo com o modelo predefinido, levantar informações adicionais como pré-condições e requisitos relacionados.
- Identificar fluxos alternativos: neste momento, é levantada apenas uma lista de exceções e variações que podem ocorrer no fluxo principal de eventos do caso de uso, sem no entanto definir como o sistema deve tratá-las.
- Descrever os passos dos fluxos alternativos: descrever como o sistema deve responder a cada exceção ou como ele deve funcionar em cada variação.

Vale ressaltar que a descrição de casos de uso na fase de análise de requisitos deve ser feita sem considerar a tecnologia de interface. Neste momento não interessa saber a forma das interfaces do sistema, mas quais informações são trocadas entre o sistema e o ambiente externo (atores). O analista deve procurar abstrair a tecnologia e se concentrar na essência das informações trocadas. Assim, diz-se que a descrição de caso de uso na fase de análise é uma descrição essencial. A tecnologia de interface será objeto da fase de projeto do sistema. Agindo dessa maneira, abre-se caminho para se pensar em diferentes alternativas de interfaces durante o projeto do sistema (WAZLAWICK, 2004).

Uma técnica de levantamento de requisitos bastante útil para apoiar a escrita de casos de uso são os cenários. Pode-se pedir para que o usuário descreva alguns cenários na forma de exemplos situados de um caso de uso em ação, mostrando o ator usando o sistema para realizar o caso de uso em questão (COCKBURN, 2005).

Um cenário é uma sequência específica de ações que ilustra o comportamento de um caso de uso. Assim, os cenários são, na verdade, instâncias de um caso de uso.

Os modelos de casos de uso são uma maneira eficaz para analistas, clientes, especialistas de domínio e usuários chegarem a uma compreensão comum acerca das funcionalidades que o sistema deve prover. Além disso, servem para ajudar a verificar e validar o sistema à medida que ele vai sendo desenvolvido. Neste contexto, os casos de uso podem ser utilizados como base para o projeto de casos de teste para o sistema, em uma abordagem de testes baseada em casos de uso, na qual casos de teste são projetados a partir dos fluxos de eventos principal e alternativos dos casos de uso, procurando explorar diferentes cenários de uso do sistema.

No contexto da Engenharia de Requisitos, casos de uso têm dois importantes papéis:

- *Casos de uso especificam os requisitos funcionais de um sistema*. Um modelo de caso de uso descreve detalhadamente o comportamento de um sistema através de um conjunto de casos de uso. O ambiente do sistema é definido pela descrição dos diferentes atores que utilizam o sistema realizando os casos de uso.
- *Casos de uso oferecem uma abordagem para a modelagem de sistemas*. Para gerenciar a complexidade de sistemas reais, é comum apresentar os modelos do sistema em um número de diferentes visões. Em uma abordagem guiada por casos de uso, pode-se construir uma visão para cada caso de uso, isto é, em cada visão são modelados apenas aqueles elementos que participam de um caso de uso específico. Essa abordagem é especialmente útil para a modelagem comportamental feita utilizando diagramas de atividade e de sequência. Um particular elemento (uma classe, p.ex.) pode, é claro, participar de vários casos de uso. Isto significa que um modelo do sistema completo só é visto através de um conjunto de visões. Para se definir todas as responsabilidades de um elemento, deve-se olhar os casos de uso onde esse elemento tem um papel.

É importante destacar que a modelagem casos de uso pode (e deve) ser realizada com algum grau de paralelismo em relação à modelagem conceitual estrutural. A identificação de conceitos relevantes para tratar um caso de uso pode ajudar a descobrir outros casos de uso relevantes, sobretudo de natureza cadastral. Assim, uma vez iniciada a descrição dos casos de uso, a modelagem conceitual estrutural pode ser também iniciada.

Além de serem uma ferramenta essencial na especificação dos requisitos funcionais de um sistema, casos de uso têm um papel fundamental no planejamento e controle de projetos iterativos. Casos de uso podem ser usados para definir o escopo de uma iteração do projeto ou mesmo do projeto como um todo. Neste contexto, uma técnica interessante para administrar discussões de escopo são as listas dentro / fora (COCKBURN, 2005). Uma lista dentro / fora é, na verdade, uma tabela com três colunas. A primeira coluna enumera casos de uso; as duas outras colunas são rotuladas “Dentro” e “Fora”. Sempre que não for claro se um caso de uso está dentro ou fora do escopo da discussão (projeto ou iteração), ele é incluído na tabela e deve-se perguntar aos interessados se o caso de uso está dentro ou fora do escopo. Assim, é possível capturar as diferentes visões dos diferentes interessados, sendo essas visões muitas vezes conflitantes. Identificados conflitos, os mesmos devem ser negociados e resolvidos.

Ainda no que se refere ao planejamento e controle de projetos iterativos, pode ser uma boa estratégia priorizar os casos de uso, de modo a definir o que considerar ou não em uma iteração (ou mesmo no projeto). Para os casos de uso considerados dentro do escopo do projeto, pode-se indicar em qual versão o caso de uso deveria ser tratado. Por exemplo, se foram planejados três iterações para o desenvolvimento de um certo sistema, os interessados poderiam indicar em qual versão (1, 2 ou 3) cada caso de uso deveria ser tratado. Essas listas de prioridades são usadas como ponto de partida para a negociação e o planejamento das iterações do projeto.

Referências do Capítulo

BLAHA, M., RUMBAUGH, J., *Modelagem e Projetos Baseados em Objetos com UML 2*, Elsevier, 2006.

BOOCH, G., RUMBAUGH, J., JACOBSON, I., *UML Guia do Usuário*, 2a edição, Elsevier Editora, 2006.

COCKBURN, A., *Escrevendo Casos de Uso Eficazes: Um guia prático para desenvolvedores de software*, Porto Alegre: Bookman, 2005.

DALPIAZ, F., FRANCH, X., HORKOFF, J., *iStar 2.0 Language Guide*, 2016.

GUIZZARDI, R.S.S, LI, F.L., BORGIDA, A., GUIZZARDI, G., HORKO, J., MYLOPOULOS, J., An Ontological Interpretation of Non-Functional Requirements. In: Proceedings of the 8th International Conference on Formal Ontology in Information Systems, FOIS 2014, Rio de Janeiro, Brazil, 2014.

HORKOFF, J., AYDEMIR, F.B., CARDOSO, E., LI T., MATE, A., PAJA, E., SALNITRI, M., MYLOPOULOS, J., GIORGINI, P., Goal-Oriented Requirements Engineering: A Systematic Literature Map. In: Proceedings of the 24th International Conference on Requirements Engineering - RE 2016, Beijing, China, 2016.

LAMSWEERDE, A., *Requirements Engineering – From System Goals to UML Models to Software Specifications*, Wiley, 2009.

NEGRI, P., SOUZA, V.E.S., LEAL, A.L.C., FALBO, R.A., GUIZZARDI, G., Towards an Ontology of Goal-Oriented Requirements, In: Proceedings of the 20th Ibero-American Conference on Software Engineering – CibSE 2017, Buenos Aires, Argentina, 2017.

OLIVÉ, A., *Conceptual Modeling of Information Systems*, Springer, 2007.

WAZLAWICK, R.S., *Análise e Projeto de Sistemas de Informação Orientados a Objetos*, Elsevier, 2004.

Capítulo 6 – Modelagem Conceitual Estrutural

Um modelo conceitual é uma abstração da realidade segundo uma conceituação. Ele pode ser usado para comunicação, aprendizado e análise de aspectos relevantes do domínio subjacente (GUIZZARDI, 2005). O modelo conceitual estrutural de um sistema tem por objetivo descrever as informações que esse sistema deve representar e gerenciar.

A modelagem conceitual é a atividade de descrever alguns dos aspectos do mundo físico e social a nossa volta, com o propósito de entender e comunicar. Os modelos resultantes das atividades de modelagem conceitual são essencialmente destinados a serem usados por pessoas e não por máquinas (MYLOPOULOS, 1992). Assim, modelos conceituais devem ser concebidos com foco no domínio do problema e não no domínio da solução e, por conseguinte, um modelo conceitual estrutural é um artefato do domínio do problema e não do domínio da solução. As informações a serem capturadas em um modelo conceitual estrutural devem existir independentemente da existência de um sistema computacional para tratá-las. Assim, o modelo conceitual deve ser independe da solução computacional a ser adotada para resolver o problema e deve conter apenas os elementos de informação referentes ao domínio do problema em questão. Elementos da solução, tais como interfaces, formas de armazenamento e comunicação, devem ser tratados apenas na fase de projeto (WAZLAWICK, 2004).

Uma vez que requisitos não-funcionais de produto (atributos de qualidade) são inerentes à solução computacional, de maneira geral, eles não são alvo na modelagem conceitual. Ou seja, não se consideram elementos de informação para tratar aspectos como desempenho, segurança, confiabilidade, formas de armazenamento etc. Esses atributos de qualidade do produto são considerados posteriormente, na fase de projeto.

Os elementos de informação básicos da modelagem conceitual estrutural são os tipos de entidades e os tipos de relacionamentos. A identificação de quais os tipos de entidades e os tipos de relacionamentos que são relevantes para um particular sistema de informação é uma meta crucial da modelagem conceitual (OLIVÉ, 2007).

Diferentes tipos de modelos podem ser usados na modelagem conceitual estrutural, tais como Diagramas de Entidades e Relacionamentos (ER) e Diagramas de Classe da UML. Este último é baseado no paradigma orientado a objetos.

Na modelagem conceitual segundo o paradigma orientado a objetos, tipos de entidades são modelados como classes. Tipos de relacionamentos são modelados como atributos e associações. Assim, o propósito da modelagem conceitual estrutural orientada a objetos é definir as classes, atributos e associações que são relevantes para tratar o problema a ser resolvido. Para tal, as seguintes tarefas devem ser realizadas:

- Identificação de Classes
- Identificação de Atributos e Associações
- Especificação de Hierarquias de Generalização/Especialização

É importante notar que essas atividades são dependentes umas das outras e que, durante o desenvolvimento, elas são realizadas de forma paralela e iterativa, sempre visando ao entendimento do domínio do problema, desconsiderando aspectos de implementação.

Este capítulo aborda a modelagem conceitual estrutural quando realizada segundo o paradigma orientado a objetos. A Seção 6.1 apresenta os principais conceitos da orientação a objetos. A Seção 6.2 discute como identificar classes. A Seção 6.3 aborda a identificação de atributos e associações. Finalmente, a Seção 6.4 discute a especificação de hierarquias de generalização/especialização.

6.1 – Conceitos da Orientação a Objetos

Para conduzir a modelagem conceitual estrutural, é necessário adotar um paradigma de desenvolvimento. Um dos paradigmas mais adotados atualmente é a orientação a objetos (OO). Segundo esse paradigma, o mundo é visto como sendo composto por *objetos*, onde um objeto é uma entidade que combina estrutura de dados e comportamento funcional. No paradigma OO, os sistemas são modelados como objetos que interagem.

A orientação a objetos oferece um número de conceitos bastante apropriados para a modelagem de sistemas. Os modelos baseados em objetos são úteis para a compreensão de problemas, para a comunicação com os especialistas e usuários das aplicações, e para a realização das tarefas ao longo do processo de desenvolvimento de software. O paradigma OO utiliza uma perspectiva humana de observação da realidade, incluindo objetos, classificação e compreensão hierárquica.

O mundo real é extremamente complexo. Quanto mais de perto o observamos, mais claramente percebemos sua complexidade. A orientação a objetos tenta gerenciar a complexidade inerente dos problemas do mundo real, abstraindo conhecimento relevante e encapsulando-o em objetos. De fato, alguns princípios básicos gerais para a administração da complexidade norteiam o paradigma orientado a objetos, entre eles abstração, encapsulamento e modularidade.

Uma das principais formas do ser humano lidar com a complexidade é através do uso de **abstrações**. As pessoas tipicamente tentam compreender o mundo, construindo modelos mentais de partes dele. Tais modelos são uma visão simplificada de algo, onde apenas os elementos relevantes são considerados. Modelos, portanto, são mais simples do que os complexos sistemas que eles modelam.

Seja o exemplo de um mapa representando um modelo de um território. Um mapa é útil porque abstrai apenas as características do território que se deseja modelar. Se um mapa incluísse todos os detalhes do território, provavelmente teria o mesmo tamanho do território e, portanto, não serviria a seu propósito.

Da mesma forma que um mapa precisa ser significativamente menor que o território que ele mapeia, incluindo apenas as informações selecionadas, um modelo conceitual deve abstrair apenas as características relevantes de um sistema para seu entendimento. Assim, pode-se definir abstração como sendo o princípio de ignorar aspectos não relevantes de um assunto, segundo a perspectiva de um observador, tornando possível uma concentração maior nos aspectos principais do mesmo. De fato, a abstração consiste na seleção que um observador faz de alguns aspectos de um assunto, em detrimento de outros que não demonstram ser relevantes para o propósito em questão. A Figura 6.1 ilustra o conceito de abstração.

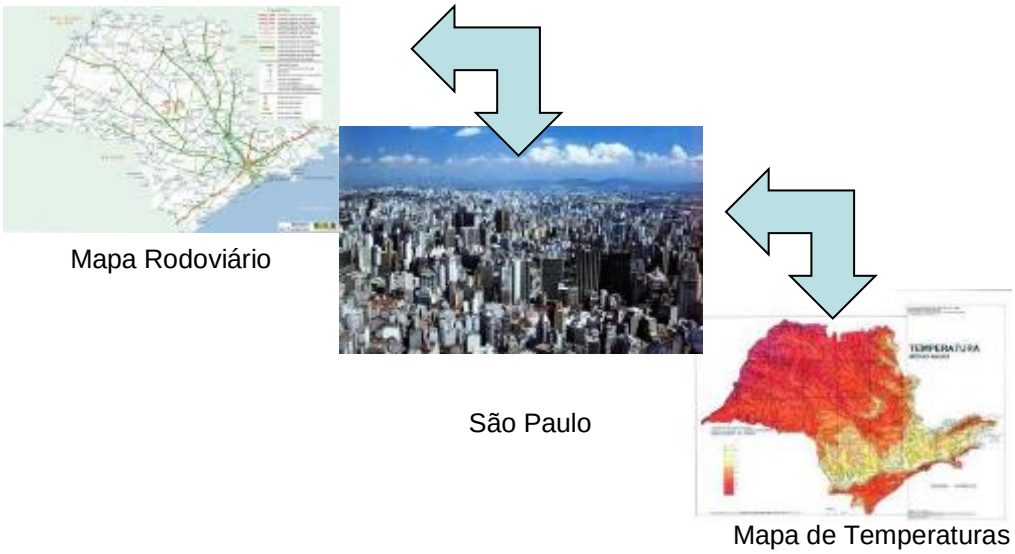


Figura 6.1 – Ilustração do Conceito de Abstração.

No mundo real, um objeto pode interagir com outro sem conhecer seu funcionamento interno. Uma pessoa, por exemplo, utiliza um forno de micro-ondas sem saber efetivamente qual a sua estrutura interna ou como seus mecanismos internos são ativados. Para utilizá-lo, basta saber usar seu painel de controle (a interface do aparelho) para realizar as operações de ligar/desligar, informar o tempo de cozimento etc. Como essas operações produzem os resultados, não interessa ao cozinheiro, como ilustra a Figura 6.2. Na OO, a este princípio de administração da complexidade, dá-se o nome de encapsulamento.

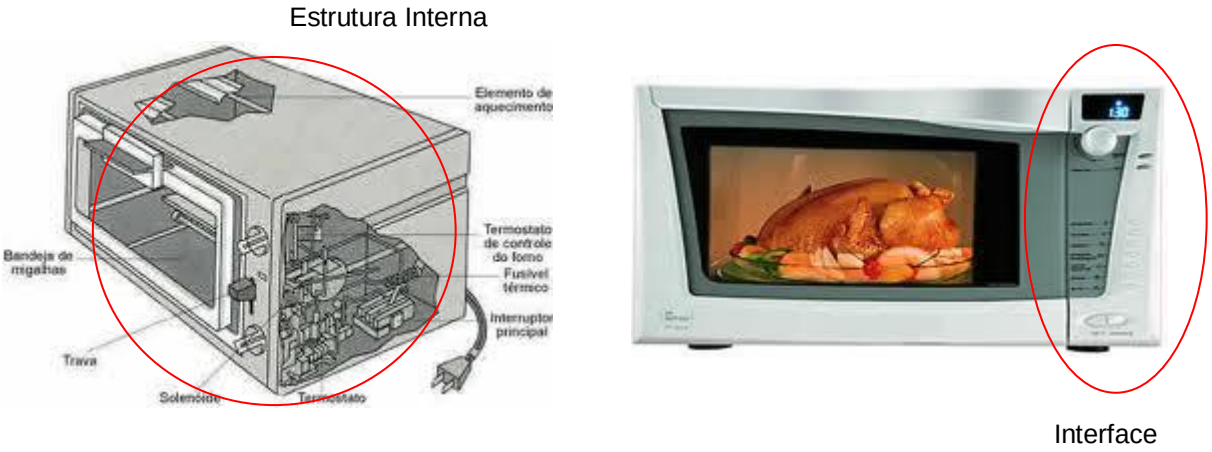


Figura 6.2 – Ilustração do Conceito de Encapsulamento.

O **encapsulamento** consiste na separação dos aspectos externos de um objeto, acessíveis por outros objetos, de seus detalhes internos de implementação, que ficam ocultos dos demais objetos (BLAHA; RUMBAUGH, 2006). A interface de um objeto deve ser definida de forma a revelar o menos possível sobre o seu funcionamento interno.

Abstração e encapsulamento são conceitos complementares: enquanto a abstração enfoca o comportamento observável de um objeto, o encapsulamento oculta a implementação que origina esse comportamento. Encapsulamento é frequentemente conseguido através da ocultação de informação, isto é, escondendo detalhes que não contribuem para suas características essenciais. Tipicamente, em um sistema OO, a estrutura de um objeto e a

implementação de seus métodos são encapsuladas (BOOCH, 1994). Assim, o encapsulamento serve para separar a interface contratual de uma abstração e sua implementação. Os usuários têm conhecimento apenas das operações que podem ser requisitadas e precisam estar cientes apenas do *quê* as operações realizam e não *como* elas estão implementadas.

A principal motivação para o encapsulamento é garantir estabilidade aos sistemas. Um encapsulamento bem feito pode servir de base para a localização de decisões de projeto que necessitam ser alteradas. Uma operação pode ter sido implementada de maneira ineficiente e, portanto, pode ser necessário escolher um novo algoritmo. Se a operação está encapsulada, apenas o objeto que a define precisa ser modificado, garantindo estabilidade ao sistema.

Por fim, os métodos de desenvolvimento de software buscam obter sistemas modulares, isto é, construídos a partir de elementos que sejam autônomos e conectados por uma estrutura simples e coerente. **Modularidade** visa à obtenção de sistemas decompostos em um conjunto de módulos coesos e fracamente acoplados e é crucial para se obter sistemas fáceis de manter e estender. Abstração, encapsulamento e modularidade são princípios sinergéticos, isto é, ao se trabalhar bem com um deles, está-se aperfeiçoando os outros também.

O paradigma OO procura fazer uso dos mecanismos de administração da complexidade para modelar, projetar e implementar sistemas. De maneira simples, o paradigma OO traz uma visão de mundo em que os fenômenos e domínios são vistos como coleções de objetos interagindo entre si. Essa visão simplista do paradigma OO esconde conceitos importantes da orientação a objetos que são a base para o desenvolvimento OO, tais como classes, associações, generalização etc. A seguir os principais conceitos da orientação a objetos são discutidos.

Objetos

O mundo real é povoado por entidades que interagem entre si, onde cada uma delas desempenha um papel específico. Na orientação a objetos, essas entidades são ditas *objetos*. Objetos podem ser coisas concretas ou abstratas, tais como um carro, uma reserva de passagem aérea, uma organização etc.

Do ponto de vista da modelagem de sistemas, um objeto é uma entidade que incorpora uma abstração relevante no contexto de uma aplicação. Um objeto possui um estado (informação), exibe um comportamento bem definido (dado por um número de operações para examinar ou alterar seu estado) e tem identidade única. O estado de um objeto compreende o conjunto de suas propriedades, associadas a seus valores correntes. Propriedades de objetos são geralmente referenciadas como atributos e associações. Portanto, o estado de um objeto diz respeito aos seus atributos/associações e aos valores a eles associados. Operações são usadas para recuperar ou manipular a informação de estado de um objeto e se referem apenas às estruturas de dados do próprio objeto, não devendo acessar diretamente estruturas de outros objetos. Caso a informação necessária para a realização de uma operação não esteja disponível, o objeto terá de colaborar com outros objetos.

A comunicação entre objetos dá-se por meio de *troca de mensagens*. Para requisitar uma operação de um objeto, é necessário enviar uma mensagem para ele. Uma mensagem é composta do nome da operação sendo requisitada e dos argumentos requeridos. Assim, o comportamento de um objeto representa como esse objeto reage às mensagens a ele enviadas. Em outras palavras, o conjunto de mensagens a que um objeto pode responder representa o seu comportamento. Um objeto é, pois, uma entidade que tem seu estado representado por um conjunto de atributos e associações (uma estrutura de informação) e seu comportamento representado por um conjunto de operações.

Cada objeto tem uma identidade própria que lhe é inerente. Todos os objetos têm existência própria, ou seja, dois objetos são distintos, mesmo se seus estados e comportamentos forem iguais. A identidade de um objeto transcende os valores correntes de suas propriedades.

Classes

No mundo real, diferentes objetos desempenham um mesmo papel. Seja o caso de duas cadeiras iguais. Apesar de serem objetos diferentes, elas compartilham uma mesma estrutura e um mesmo comportamento. Entretanto, não há necessidade de se despender tempo modelando cada cadeira. Basta definir, em um único lugar, um modelo descrevendo a estrutura e o comportamento desses objetos. A esse modelo dá-se o nome de *classe*. Uma classe descreve um conjunto de objetos com a mesma estrutura (atributos e associações), o mesmo comportamento (operações) e a mesma semântica. Objetos que se comportam da maneira especificada pela classe são ditos *instâncias* dessa classe.

Todo objeto pertence a uma classe, ou seja, é instância de uma classe. De fato, a orientação a objetos norteia o processo de desenvolvimento através da *classificação de objetos*, isto é, objetos são agrupados em classes, em função de exibirem características similares, sem, no entanto, perda de sua individualidade, como ilustra a Figura 6.3. Assim, a modelagem orientada a objetos consiste, basicamente, na definição de classes. O comportamento e a estrutura de informação de uma instância são definidos pela sua classe. Objetos com propriedades e comportamento idênticos são descritos como instâncias de uma mesma classe, de modo que a descrição de suas propriedades possa ser feita uma única vez, de forma concisa, independentemente do número de objetos que tenham tais propriedades em comum. Deste modo, uma classe captura as características comuns a todas as suas instâncias.

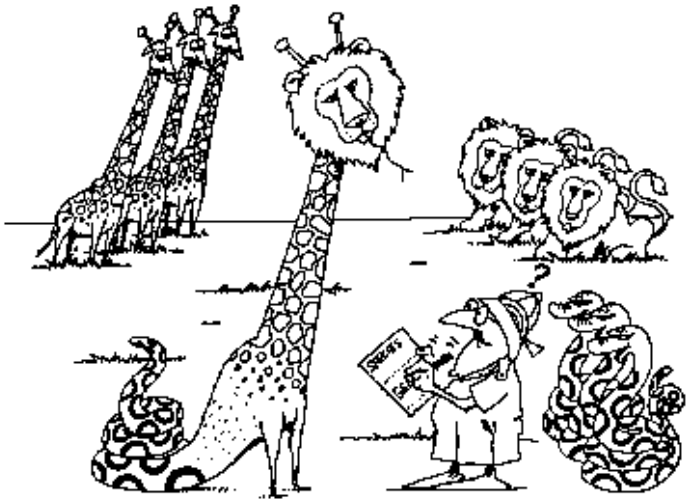


Figura 6.3 – Ilustração do conceito de Classificação (BOOCH, 1994).

Enquanto um objeto individual é uma entidade real, que executa algum papel no sistema como um todo, uma classe captura a estrutura e o comportamento comuns a todos os objetos que ela descreve. Assim, uma classe serve como uma espécie de contrato que deve ser estabelecido entre uma abstração e todos os seus clientes.

Ligações e Associações

Em qualquer sistema, objetos relacionam-se uns com os outros. Por exemplo, em "*o empregado João trabalha no Departamento de Pessoal*", temos um relacionamento entre o objeto *empregado João* e o objeto *Departamento de Pessoal*.

Ligações e associações são meios de se representar relacionamentos entre objetos e entre classes, respectivamente. Uma ligação é uma conexão entre objetos. No exemplo anterior, há uma ligação entre os objetos *João* e *Departamento de Pessoal*. Uma associação, por sua vez, descreve um conjunto de ligações com estrutura e semântica comuns. No exemplo anterior, há uma associação entre as classes *Empregado* e *Departamento*. Todas as ligações de uma associação interligam objetos das mesmas classes e, assim, uma associação descreve um conjunto de potenciais ligações da mesma maneira que uma classe descreve um conjunto de potenciais objetos (BLAHA; RUMBAUGH, 2006).

Generalização / Especialização

Muitas vezes, um conceito geral pode ser especializado, adicionando-se a ele novas características. Seja o exemplo do conceito de *estudante* no contexto de uma universidade. De modo geral, há características que são intrínsecas a quaisquer estudantes da universidade. Entretanto, é possível especializar esse conceito para mostrar especificidades de subtipos de estudantes, tais como estudantes de graduação e estudantes de pós-graduação.

De maneira inversa, pode-se extrair de um conjunto de conceitos características comuns que, quando generalizadas, formam um conceito geral. Por exemplo, ao se avaliar os conceitos de carros, motos, caminhões e ônibus, pode-se notar que eles têm características comuns que podem ser generalizadas em um supertipo *veículo automotor terrestre*.

As abstrações de especialização e generalização são muito úteis para a estruturação de sistemas. Com elas, é possível construir hierarquias de classes. A *herança* é um mecanismo para modelar similaridades entre classes, representando as abstrações de generalização e especialização. Através da herança, é possível tornar explícitos atributos, associações e operações comuns em uma hierarquia de classes. O mecanismo de herança possibilita reutilização, captura explícita de características comuns e definição incremental de classes. No que se refere à definição incremental de classes, a herança permite conceber uma nova classe como um refinamento de outras classes. A nova classe pode herdar as similaridades e definir apenas as novas características.

A herança é, portanto, um relacionamento entre classes (em contraposição às associações que representam relacionamentos entre objetos das classes), no qual uma classe compartilha a estrutura e o comportamento definidos em outras (uma ou mais) classes. A classe que herda características[12] é dita *subclasse* e a que fornece as características, *superclasse*. Desta forma, a herança representa uma hierarquia de abstrações na qual uma subclasse herda de uma ou mais superclasses.

Tipicamente, uma subclasse aumenta ou redefine características de suas superclasses. Assim, se uma classe *B* herda de uma classe *A*, todas as características descritas em *A* tornam-se automaticamente parte de *B*, que ainda é livre para acrescentar novas características para seus propósitos específicos.

A generalização permite abstrair, a partir de um conjunto de classes, uma classe mais geral contendo todas as características comuns. A especialização é a operação inversa e, portanto, permite especializar uma classe em um número de subclasses, explicitando as diferenças entre as novas subclasses. Deste modo é possível compor uma hierarquia de classes. Esses tipos de relacionamento são conhecidos também como relacionamentos "*é um tipo de*",

[12] O termo *característica* é usado aqui para designar estrutura (atributos e associações) e comportamento (operações).

onde um objeto da subclasse também "*é um tipo de*" objeto da superclasse. Neste caso, uma instância da subclasse é dita uma *instância indireta* da superclasse. Quando uma subclasse herda características de uma única superclasse, tem-se *herança simples*. Quando uma classe é definida a partir de duas ou mais superclasses, tem-se *herança múltipla*.

Mensagens, Operações e Métodos

A abstração incorporada por um objeto é caracterizada por um conjunto de operações que podem ser requisitadas por outros objetos, ditos clientes. Métodos são implementações reais de operações. Para que um objeto realize alguma tarefa, é necessário enviar uma mensagem a ele, solicitando a realização de uma operação. Um cliente só pode acessar um objeto através da emissão de mensagens, isto é, ele não pode acessar ou manipular diretamente os dados associados ao objeto. Os objetos podem ser complexos e o cliente não precisa tomar conhecimento de sua complexidade interna. O cliente precisa saber apenas como se comunicar com o objeto e como ele reage. Assim, garante-se o encapsulamento.

As mensagens são o meio de comunicação entre objetos e são responsáveis pela ativação de todo e qualquer processamento. Dessa forma, é possível garantir que clientes não serão afetados por alterações nas implementações de um objeto que não alterem o comportamento esperado de seus serviços.

Classes e Operações Abstratas

Nem todas as classes são projetadas para instanciar objetos. Algumas são usadas simplesmente para organizar características comuns a diversas classes. Tais classes são ditas *classes abstratas*. Uma classe abstrata é desenvolvida basicamente para ser herdada por outras classes. Ela existe meramente para que um comportamento comum a um conjunto de classes possa ser colocado em uma localização comum e definido uma única vez. Assim, uma classe abstrata não possui instâncias diretas, mas suas classes descendentes *concretas*, sim. Uma classe concreta é uma classe passível de instanciação, isto é, que pode ter instâncias diretas. Uma classe abstrata pode ter subclasses também abstratas, mas as classes-folhas na árvore de herança devem ser classes concretas.

Classes abstratas podem ser projetadas de duas maneiras distintas. Primeiro, elas podem prover implementações completamente funcionais do comportamento que pretendem capturar. Alternativamente, elas podem prover apenas a definição de um protocolo para uma operação, sem apresentar um método correspondente. Tal operação é dita uma *operação abstrata*. Neste caso, a classe abstrata não é completamente implementada e todas as suas subclasses concretas são obrigadas a prover uma implementação para suas operações abstratas. Assim, diz-se que uma operação abstrata define apenas a assinatura[13] a ser usada nas implementações que as subclasses deverão prover, garantindo, assim, uma interface consistente. Métodos que implementam uma operação genérica têm de ter a mesma semântica.

Uma classe concreta não pode conter operações abstratas, porque senão seus objetos teriam operações indefinidas. Assim, toda classe que possuir uma operação abstrata não pode ter instâncias diretas e, portanto, obrigatoriamente é uma classe abstrata.

[13] nome da operação, parâmetros e retorno

6.2 – Identificação de Classes

Classificação é o meio pelo qual os seres humanos estruturam a sua percepção do mundo e seu conhecimento sobre ele. Sem ela, não é possível nem entender o mundo a nossa volta nem agir sobre ele. Classificação assume a existência de tipos e de objetos a serem classificados nesses tipos. Classificar consiste, então, em determinar se um objeto é ou não uma instância de um tipo. A classificação nos permite estruturar conhecimento sobre as coisas em dois níveis: tipos e instâncias. No nível de tipos, procuramos encontrar as propriedades comuns a todas as instâncias de um tipo. No nível de instância, procuramos identificar o tipo do qual o objeto é uma instância e os valores particulares das propriedades desse objeto (OLIVÉ, 2007).

Tipos de entidade são um dos mais importantes elementos em modelos conceituais. Definir os tipos de entidade relevantes para um particular sistema de informação é uma tarefa crucial na modelagem conceitual. Um tipo de entidade pode ser definido como um tipo cujas instâncias em um dado momento são objetos individuais identificáveis que se consideram existir no domínio naquele momento. Um objeto pode ser instância de vários tipos ao mesmo tempo (OLIVÉ, 2007). Por exemplo, seja o caso dos tipos *Estudante* e *Funcionário* em um sistema de uma universidade. Uma mesma pessoa, por exemplo João, pode ser ao mesmo tempo um estudante e um funcionário dessa universidade.

Na orientação a objetos, tipos de entidade são representados por classes, enquanto as instâncias de um tipo de entidade são objetos. Assim, uma atividade crucial da modelagem conceitual estrutural segundo o paradigma OO é a identificação de classes. Na UML, classes são representadas por um retângulo com três compartimentos: o compartimento superior é relativo ao nome da classe; o compartimento do meio é dedicado à especificação dos atributos da classe; e o compartimento inferior é dedicado à especificação das operações da classe, como mostra a Figura 6.4.

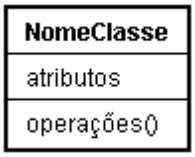


Figura 6.4 – Notação de Classes na UML.

Para nomear classes, sugere-se iniciar com um substantivo no singular, o qual pode ser combinado com complementos ou adjetivos, omitindo-se preposições. O nome da classe deve ser iniciado com letra maiúscula, bem como os nomes dos complementos, sem dar um espaço em relação à palavra anterior. Acentos não devem ser utilizados. Ex.: Cliente, PessoaFisica, ItemPedido.

Tomando por base os requisitos iniciais de cliente, relatórios de atividades de levantamento de requisitos (p.ex., atas de reunião, ou documentos em uma investigação de documentos) e, sobretudo, descrições de casos de uso, é possível iniciar a modelagem conceitual estrutural. Esse trabalho começa com a descoberta de quais classes devem ser incluídas no modelo, já que o cerne de um modelo OO é exatamente o seu conjunto de classes.

Durante a análise de requisitos, tipicamente o analista estuda, filtra e modela o domínio do problema. Dizemos que o analista "filtra" o domínio, pois apenas uma parte desse domínio fará parte das responsabilidades do sistema. Assim, um domínio de problemas pode incluir várias informações, mas as responsabilidades de um sistema nesse domínio podem incluir apenas uma pequena parcela desse conjunto.

As classes de um modelo representam a expressão inicial do sistema. As atividades subsequentes da modelagem estrutural buscam obter uma descrição cada vez mais detalhada, em termos de associações e atributos. Contudo, deve-se observar que, à medida que atributos e associações vão sendo identificados, se ganha maior entendimento a respeito do domínio e naturalmente novas classes podem surgir. Assim, as atividades da modelagem conceitual são iterativas e com alto grau de paralelismo, devendo ser realizadas concomitantemente.

Um aspecto fundamental no processo de modelagem conceitual é a interação constante com os especialistas de domínio. Técnicas de levantamento de requisitos, tais como entrevistas, análise de documentos e reuniões JAD, têm um papel fundamental nesta etapa. Assim, o levantamento de requisitos continua acontecendo paralelamente à modelagem conceitual e os relatórios produzidos nessas atividades são fontes importantíssimas de informação para a elaboração de modelos conceituais.

Uma maneira bastante prática e eficaz de trabalhar a identificação de classes consiste em olhar esses documentos à procura de classes. Diversos autores, dentre eles Jacobson (1992) e Wazlawick (2004), sugerem que uma boa estratégia para identificar classes consiste em ler esses documentos procurando por substantivos. Esses autores argumentam que uma classe é, tipicamente, descrita por um nome no domínio e, portanto, aprender sobre a terminologia do domínio do problema é um bom ponto de partida. Ainda que um bom ponto de partida, essa heurística é ainda muito vaga. Se o analista segui-la fielmente, muito provavelmente ele terá uma extensa lista de potenciais classes, sendo que muitas delas podem, na verdade, se referir a atributos de outras classes. Além disso, pode ser que importantes classes não sejam capturadas, notadamente aquelas que se referem ao registro de eventos de negócio, uma vez que esses eventos, muitas vezes, são descritos na forma de verbos. Seja o seguinte exemplo de uma descrição de um domínio de locação de automóveis: "clientes locam carros". Seriam consideradas potenciais classes: *Cliente* e *Carro*. Contudo, a locação é um evento de negócio importante que precisa ser registrado e, usando a estratégia de identificar classes a partir de substantivos, *Locação* não entraria na lista de potenciais classes.

Assim, neste texto sugere-se que, ao examinar documentos de requisitos de cliente, relatórios de atividades de levantamento de requisitos e descrições de casos de uso, os seguintes elementos sejam considerados como candidatos a classes:

- Agentes: entidades do domínio do problema que têm a capacidade de agir com intenção de atingir uma meta. Em sistemas de informação, há dois tipos principais de agentes: os agentes físicos (tipicamente pessoas) e os agentes sociais (organizações, unidades organizacionais, sociedades etc.). Em relação às pessoas, deve-se olhar para os papéis desempenhados pelas diferentes pessoas no domínio do problema.

- Objetos: entidades sem a capacidade de agir, mas que fazem parte do domínio de informação do problema. Podem ser também classificados em físicos (p.ex., carros, livros, imóveis) e sociais (p.ex., cursos, disciplinas, leis). Entretanto, há também outros tipos de objetos, tais como objetos de caráter descritivo usados para organizar e descrever outros objetos de um domínio (p.ex., modelos de carro). Objetos sociais e de descrição tendem a ser coisas menos tangíveis, mas são tão importantes para a modelagem conceitual quanto os objetos físicos.

- Eventos: representam a ocorrência de ações no domínio do problema que precisam ser registradas e lembradas pelo sistema. Eventos acontecem no tempo e, portanto, a representação de eventos normalmente envolve a necessidade de registrar, dentre outros, quando o evento ocorreu (ponto no tempo ou intervalo de tempo). Deve-se

observar que muitos eventos ocorrem no domínio do problema, mas grande parte deles não precisa ser lembrada. Para capturar os eventos que precisam ser lembrados e, portanto, registrados, devem-se focalizar os principais eventos de negócio do domínio do problema. Assim, em um sistema de locação de automóveis, são potenciais classes de eventos: *Locação*, *Devolução* e *Reserva*. Por outro lado, a ocorrência de eventos cadastrais, tais como os cadastros de clientes e carros, tende a ser de pouca importância, não sendo necessário lembrar a ocorrência desses eventos.

Seja qual for a estratégia usada para identificar classes, é sempre importante que o analista tenha em mente os objetivos do sistema durante a modelagem conceitual. Não se devem representar informações irrelevantes para o sistema e, portanto, a relevância para o sistema é o principal critério a ser adotado para decidir se um determinado elemento deve ou não ser incluído no modelo conceitual estrutural do sistema.

O resultado principal da atividade de identificação de classes é a obtenção de uma lista de potenciais classes para o sistema em estudo. Um modelo conceitual estrutural para uma aplicação complexa pode conter dezenas de classes e, portanto, pode ser necessário definir uma representação concisa capaz de orientar um leitor em um modelo dessa natureza. O agrupamento de classes em subsistemas serve basicamente a este propósito, podendo ser útil também para a organização de grupos de trabalho em projetos extensos. Conforme discutido no Capítulo 5, a base principal para a identificação de subsistemas é a complexidade do domínio do problema. Através da identificação e agrupamento de classes em subsistemas, é possível controlar a visibilidade do leitor e, assim, tornar o modelo mais compreensível. Assim, da mesma maneira que casos de uso são agrupados em pacotes, classes também devem ser.

Quando uma coleção de classes colabora entre si para realizar um conjunto coeso de responsabilidades (casos de uso), elas podem ser vistas como um subsistema. Assim, um subsistema é uma abstração que provê uma referência para mais detalhes em um modelo de análise, incluindo tanto casos de uso quanto classes. O agrupamento de classes em subsistemas permite apresentar o modelo global em uma perspectiva mais alta. Esse nível ajuda o leitor a rever o modelo, bem como constitui um bom critério para organizar a documentação.

Uma vez identificadas as potenciais classes, deve-se proceder uma avaliação para decidir o que efetivamente considerar ou rejeitar. Conforme discutido anteriormente, a relevância para o sistema deve ser o critério principal. Além desse critério, os seguintes critérios devem ser considerados nessa avaliação:

- Estrutura complexa: o sistema precisa tratar informações sobre os objetos da classe? Tipicamente, uma classe deve ter, pelo menos, dois atributos. Se uma classe apresentar apenas um atributo, avalie se não é melhor tratá-la como um atributo de uma classe existente[14].
- Atributos e associações comuns: os atributos e as associações da classe devem ser aplicáveis a todas as suas instâncias, isto é, a todos os objetos da classe.
- Existência de instâncias: toda classe deve possuir instâncias. Uma potencial classe que possua uma única instância não deve ser considerada em um modelo conceitual estrutural. Tipicamente uma classe possui várias instâncias e a população da classe varia ao longo do tempo.

[14] Uma classe que possui um único atributo, mas várias associações, também satisfaz a esse critério.

6.3 – Identificação de Atributos e Associações

Uma classe típica de um modelo conceitual estrutural apresenta estrutura complexa. A estrutura de uma classe corresponde a seus atributos e associações. Conceitualmente, não há diferença entre atributos e associações. Atributos são, na verdade, um tipo de relacionamento binário. Em um tipo de relacionamento binário, há dois participantes. Em alguns tipos de relacionamentos, esses participantes são considerados "colegas", porque eles desempenham funções análogas e nenhum deles é subordinado ao outro. Seja o caso do tipo de relacionamento "*aluno cursa um curso*". Um aluno só é aluno se cursar um curso. Por outro lado, não faz sentido existir um curso se ele não puder ser cursado por alunos. A ordem dos participantes no modelo não implica uma relação de prioridade ou subordinação entre eles (OLIVÉ, 2007). Na orientação a objetos, esse tipo de relacionamento é modelado como uma associação.

Entretanto, há alguns tipos de relacionamentos nos quais usuários e analistas consideram um participante como sendo uma característica do outro. Seja o exemplo do tipo de relacionamento "*filme possui gênero*". *Gênero* é uma característica de *filme* e, portanto, subordinado a este. Esse tipo de relacionamento é modelado como um atributo. Assim, um atributo é um tipo de relacionamento binário em que um participante é considerado uma característica de outro. Por conseguinte, um atributo é igual a uma associação, exceto pelo fato de usuários e analistas adicionarem a interpretação que um dos participantes é subordinado ao outro (OLIVÉ, 2007).

De uma perspectiva mais prática, atributos ligam classes do domínio do problema a tipos de dados. Associações, por sua vez, consistem em um tipo de informação que liga diferentes classes do domínio entre si.

Tipos de dados podem ser primitivos ou específicos de domínio. Os tipos de dados primitivos são aplicáveis aos vários domínios e sistemas, tais como strings, datas, inteiros e reais, e são considerados como sendo predefinidos. Os tipos de dados específicos de um domínio de aplicação, por outro lado, precisam ser definidos. São exemplos de tipos de dados específicos: CPF, ISBN de livros, endereço etc.

Neste texto são considerados os seguintes tipos de dados primitivos:

- *String*: cadeia de caracteres;
- *boolean*: admite apenas os valores verdadeiro e falso;
- *Integer* (ou *int*): números inteiros;
- *Float* (ou *float*): números reais;
- *Currency*: valor em moeda (reais, dólares etc.);
- *Date*: datas, com informação de dia, mês e ano;
- *Time*: horas em um dia, com informação de hora, minuto e segundo;
- *DateTime*: combinação dos dois anteriores;
- *YearMonth*: informação de tempo contendo apenas mês e ano;
- *Year*: informação de tempo contendo apenas ano.

6.3.1 – Atributos

Um atributo é uma informação de estado para a qual cada objeto em uma classe tem o seu próprio valor. Os atributos adicionam detalhes às abstrações e são apresentados na parte central do símbolo de classe. Atributos possuem um tipo de dado, que pode ser primitivo ou específico de domínio. Ao identificar um atributo como sendo relevante, deve-se definir qual o seu tipo de dado. Caso nenhum dos tipos de dados primitivos se aplique, deve-se definir, então, um tipo de dados específico. Por exemplo, em domínios que lidem com livros, é necessário definir o tipo ISBN[15], cujas instâncias são ISBNs válidos. Em domínios que lidem com pessoas físicas e jurídicas, CPF e CNPJ também devem ser definidos como tipos de dados específicos. Usar um tipo de dados primitivo nestes casos, tal como *String* ou *int*, é insuficiente, pois não são quaisquer cadeias de caracteres ou números que se caracterizam como ISBNs, CPFs ou CNPJs válidos.

Tipos de dados específicos podem apresentar propriedades. Por exemplo, CPF é um número de 11 dígitos, que pode ser dividido em duas partes: os 9 primeiros dígitos e os dois últimos, que são dígitos verificadores. Um tipo de dados especial é a enumeração. Na enumeração, os valores do tipo são enumerados explicitamente na forma de literais, como é o caso do tipo *DiaSemana*, que é tipicamente definido como um tipo de dados compreendendo sete valores: {Segunda, Terça, Quarta, Quinta, Sexta, Sábado e Domingo}. É importante observar que tipos de dados enumerados só devem ser usados quando se sabe a priori quais são os seus valores e eles são fixos. Assim, são bons candidatos a tipos enumerados informações como sexo (M/F), estado civil, etc.

Tipos de dados geralmente não são representados graficamente em um modelo conceitual estrutural, de modo a torná-lo mais simples. Na maioria das situações, basta descrever os tipos de dados específicos de domínio no Dicionário de Dados do Projeto. Contudo, se necessário, eles podem ser representados graficamente usando o símbolo de classe estereotipado com a palavra chave <<*dataType*>>. Tipos enumerados também podem ser representados usando o símbolo de classe, mas com o estereótipo <<*enumeration*>> ou <<*enum*>>, sendo que ao invés de apresentar atributos de um tipo de dados, enumeram-se os valores possíveis da enumeração. A Figura 6.5 ilustra a notação de tipos de dados na UML.

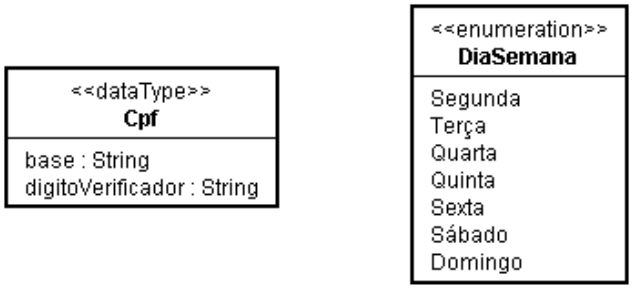


Figura 6.5 – Notação de Tipos de Dados na UML.

Uma dúvida típica e recorrente na modelagem estrutural é se um determinado item de informação deve ser modelado como uma classe ou como um atributo. Para que o item seja considerado uma classe, ele tem de passar nos critérios de inclusão no modelo discutidos na seção anterior. Entretanto, há alguns itens de informação que passam nesses critérios, mas que

[15] O ISBN - *International Standard Book Number* - é um sistema internacional padronizado que identifica numericamente os livros segundo o título, o autor, o país, a editora, individualizando-os inclusive por edição. Utilizado também para identificar software, seu sistema numérico pode ser convertido em código de barras.

ainda assim podem ser melhor modelados como atributos, tendo como tipo um tipo de dado complexo, específico de domínio. Um atributo deve capturar um conceito atômico, i.e., um único valor ou um agrupamento de valores fortemente relacionados que sirva para descrever outro objeto. Além disso, para que um item de estrutura complexa seja modelado como um atributo, ele deve ser compreensível pelos interessados simplesmente pelo seu nome.

É muito importante lembrar também que, uma vez que atributos e associações são tipos de relacionamentos, não devemos incluir na lista de atributos de uma classe, atributos representando associações (ou atributos representando "chaves estrangeiras" como se a classe fosse uma tabela de um banco de dados relacional). Associações já têm sua presença indicada pela notação de associação, ou seja, pelas linhas que conectam as classes que se relacionam.

Um aspecto bastante importante na especificação de atributos é a escolha de nomes. Deve-se procurar utilizar o vocabulário típico do domínio do problema, usando nomes significativos. Para nomear atributos, sugerem-se nomes iniciando com substantivo, o qual pode ser combinado com complementos ou adjetivos, omitindo-se preposições. O nome do atributo deve ser iniciado com letra minúscula, enquanto os nomes dos complementos devem iniciar com letras maiúsculas, sem dar um espaço em relação à palavra anterior. Acentos não devem ser utilizados. Atributos monovalorados devem iniciar com substantivo no singular (p.ex., nome, razaoSocial), enquanto atributos multivalorados devem iniciar com o substantivo no plural (p.ex., telefones).

A sintaxe de atributos na UML, em sua forma plena, é a seguinte (BOOCH; RUMBAUGH; JACOBSON, 2006):

visibilidade nome: tipo [multiplicidade] = valorInicial {propriedades}

A visibilidade de um atributo indica em que situações esse atributo é visível por outras classes. Na UML há quatro níveis de visibilidade, indicados pelos seguintes símbolos:

+ público : o atributo pode ser acessado por qualquer classe;

protegido: o atributo só é passível de acesso pela própria classe ou por uma de suas especializações;

- privado: o atributo só pode ser acessado pela própria classe;

~ pacote: o atributo só pode ser acessado por classes declaradas dentro do mesmo pacote da classe a que pertence o atributo.

A informação de visibilidade é inerente à fase de projeto e não deve ser expressa em um modelo conceitual. Assim, em um modelo conceitual, atributos devem ser especificados sem nenhum símbolo antecedendo o nome.

O *tipo* indica o tipo de dado do atributo, o qual deve ser um tipo de dado primitivo ou um tipo de dado específico de domínio. Tipos de dados específicos de domínio devem ser definidos no Dicionário de Dados do Projeto.

A multiplicidade é a especificação do intervalo permitido de itens que o atributo pode abrigar. O padrão é que cada atributo tenha um e somente um valor para o atributo. Quando um atributo for opcional ou quando puder ter mais do que uma ocorrência, a multiplicidade deve ser informada, indicando o valor mínimo e o valor máximo, da seguinte forma:

valor_mínimo .. valor_máximo

A seguir, são dados alguns exemplos:

- nome: String → instâncias da classe têm obrigatoriamente um e somente um nome.
- cpf: Cpf [0..1] → instâncias da classe têm um ou nenhum cpf.
- telefones: Telefone [0..*] → instâncias da classe têm um ou vários telefones.
- pessoasContato: String [2] → instâncias da classe têm exatamente duas pessoas de contato.

Um atributo pode ter um valor padrão inicial, ou seja, um valor que, quando não informado outro valor, será atribuído ao atributo. O campo *valorInicial* descreve exatamente este valor. O exemplo abaixo ilustra o uso de valor inicial.

- origem: Ponto = (0,0) → a origem, quando não informado outro valor, será o ponto (0,0)

Finalmente, podem ser indicadas propriedades dos atributos. Uma propriedade que pode ser interessante mostrar em um modelo conceitual é a propriedade *readonly*, a qual indica que o valor do atributo não pode ser alterado após a inicialização do objeto. No exemplo abaixo, está-se indicando que o valor do atributo *matricula* de um aluno não pode ser alterado.

- matricula: String {readonly}

Além das informações tratadas na declaração de um atributo seguindo a sintaxe da UML, outras informações de domínio, quando pertinentes, podem ser adicionadas no Dicionário de Dados do Projeto, tais como unidade de medida, intervalo de valores possíveis, limite, precisão etc.

6.3.2 – Associações

Uma associação é um tipo de relacionamento que ocorre entre instâncias de duas ou mais classes. Assim como classes, associações são tipos. Ou seja, uma associação modela um tipo de relacionamento que pode ocorrer entre instâncias das classes envolvidas. Uma instância de uma associação (dita uma ligação) conecta instâncias específicas das classes envolvidas na associação. Seja o exemplo de um domínio em que clientes efetuam pedidos. Esse tipo de relacionamento pode ser modelado como uma associação *Cliente efetua Pedido*. Seja Pedro uma instância de *Cliente* e Pedido100 uma instância de *Pedido*. Se foi Pedro quem efetuou o Pedido100, então a ligação (Pedro, Pedido100) é uma instância da associação *Cliente efetua Pedido*.

Associações podem ser nomeadas. Neste texto sugere-se o uso de verbos conjugados, indicando o sentido de leitura. Ex.: Cliente (classe) *efetua* ▸ (associação) Locação (classe). Cada classe envolvida na associação desempenha um papel, ao qual pode ser dado um nome. Cada classe envolvida na associação possui também uma multiplicidade[16] nessa associação, que indica quantos objetos podem participar de uma instância dessa associação. A notação da UML usada para representar associações em um modelo conceitual é ilustrada na Figura 6.6.

[16] Multiplicidades em uma associação são análogas às multiplicidades em atributos e especificam as quantidades mínima e máxima de objetos que podem participar da associação. Quando nada for dito, o padrão é 1..1 como no caso de atributos. Contudo, para deixar os modelos claros, recomenda-se sempre especificar explicitamente as multiplicidades das associações.

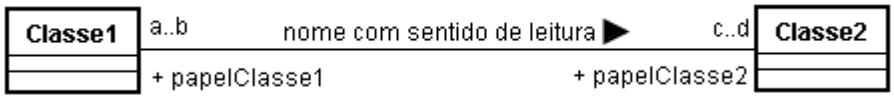


Figura 6.6 – Notação de Associações na UML.

Na ilustração da figura, um objeto da *Classe1* se relaciona com no mínimo *c* e no máximo *d* objetos da *Classe2*. Já um objeto da *Classe2* se relaciona com no mínimo *a* e no máximo *b* objetos da *Classe1*. Objetos da *Classe1* desempenham o papel de "*papelClasse1*" nesta associação, enquanto objetos da *Classe2* desempenham o papel de "*papelClasse2*" nessa mesma associação.

É importante, neste ponto, frisar a diferença entre sentido de leitura (ou direção do nome) de uma associação com a navegação da associação. O sentido de leitura diz apenas em que direção ler o nome da associação, mas nada diz sobre a navegabilidade da associação. A navegabilidade (linha de associação com seta direcionada) é usada para limitar a navegação de uma associação a uma única direção e é um recurso a ser usado apenas na fase de projeto. Em um modelo conceitual, todas as associações navegáveis nos dois sentidos e, portanto, não direcionais.

Ainda que nomes de associações e papéis sejam opcionais, recomenda-se usá-los para tornar o modelo mais claro. Além disso, há algumas situações em que fica inviável ler um modelo se não houver a especificação do nome da associação ou de algum de seus papéis.

Seja o exemplo da Figura 6.7. Em uma empresa, um empregado está lotado em um departamento e, opcionalmente, pode chefiá-lo. Um departamento, por sua vez, pode ter vários empregados nele lotados, mas apenas um chefe. Sem nomear essas associações, o modelo fica confuso. Rotulando os papéis e as associações, o modelo torna-se muito mais claro. Na Figura 6.7, um departamento exerce o papel de *departamento de lotação* do empregado e, neste caso, um empregado tem um e somente um departamento de lotação. No outro relacionamento, um empregado exerce o papel de *chefe* e, portanto, um departamento possui um e somente um chefe.

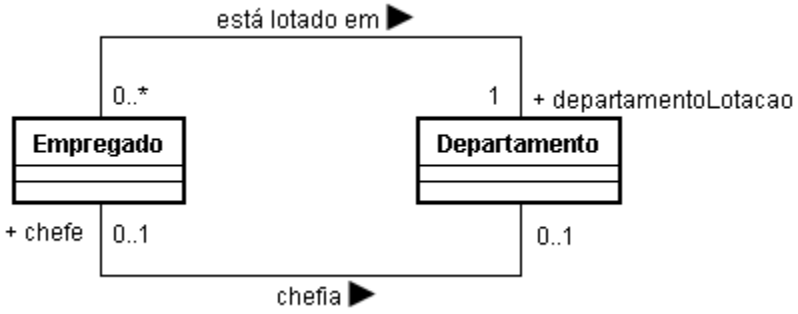


Figura 6.7 – Exemplo: Nomeando Associações.

Ao contrário das classes e dos atributos que podem ser encontrados facilmente a partir da leitura dos textos da descrição do minimundo e das descrições de casos de uso, muitas vezes, as informações sobre associações não aparecem tão explicitamente. Casos de uso descrevem ações de interação entre atores e sistema e, por isso, acabam mencionando principalmente operações. Operações transformam a informação, passando um objeto de um estado para outro, por meio da alteração dos seus valores de atributos e associações. Uma associação, por sua vez, é uma relação estática que pode existir entre duas classes. Assim, as descrições de casos de uso estão repletas de operações, mas não de associações (WAZLAWICK, 2004).

Contudo, conforme discutido na seção anterior, há alguns eventos que precisam ter sua ocorrência registrada e, portanto, são tipicamente mapeados como classes. Esses eventos estão descritos nos casos de uso e podem ter sido capturados como associações. Seja o exemplo de uma concessionária de automóveis. Neste domínio, clientes compram carros, como ilustra a parte (a) da Figura 6.8. Contudo, a compra é um evento importante para o negócio e precisa ser registrada. Neste caso, como ilustra a parte (b) da Figura 6.8, a compra deve ser tratada como uma classe e não como uma associação.

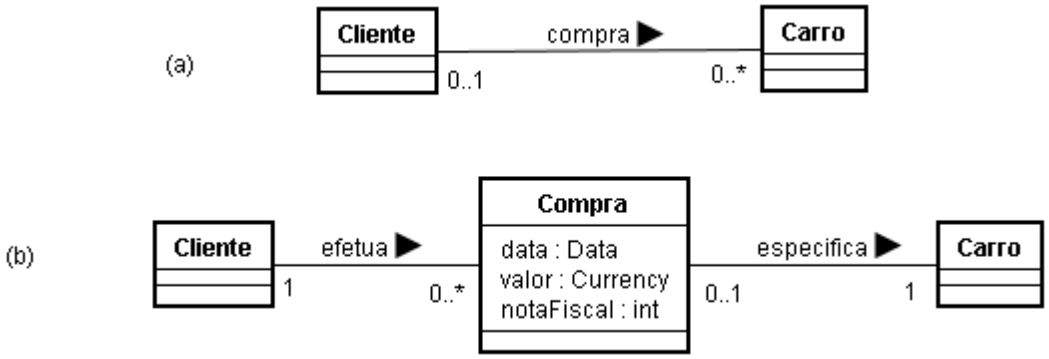


Figura 6.8 – Exemplo: Associação x Classe de Evento Lembrado.

Deve-se notar pelo exemplo acima que o evento é representado por uma classe, enquanto as associações continuam representando relacionamentos estáticos entre as classes e não operações ou transformações (WAZLAWICK, 2004). Assim, deve-se tomar cuidado com a representação de eventos como associações, questionando sempre se aquela associação é relevante para o sistema em questão.

Seja o exemplo da Figura 6.9. Nesse exemplo, o caso de uso aponta que funcionários são responsáveis por cadastrar livros em uma biblioteca. Seria necessário, pois, criar uma associação *Funcionário cadastra Livro* no modelo estrutural? A resposta, na maioria dos casos, é não. Apenas se explicitamente expresso pelo cliente em um requisito que é necessário saber exatamente qual funcionário fez o cadastro de um dado livro (o que é muito improvável de acontecer), é que tal relação deveria ser considerada. Mesmo se houver a necessidade de auditoria de uso do sistema (requisito não funcional relativo a segurança), não há a necessidade de modelar esta associação, pois requisitos não funcionais não devem ser considerados no modelo conceitual, uma vez que soluções bastante distintas à do uso dessa associação poderiam ser adotadas.

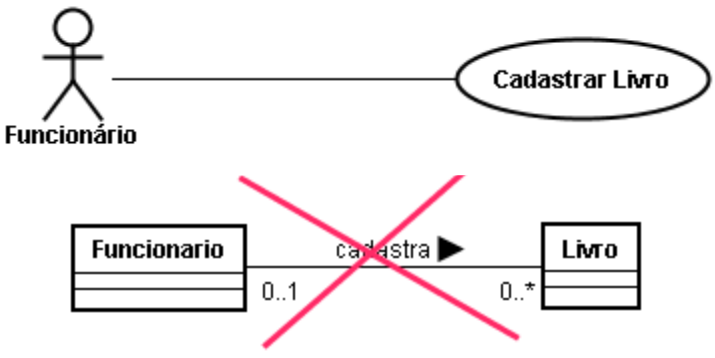


Figura 6.9 – Exemplo: Associação x Caso de Uso.

Na modelagem conceitual é fundamental saber a quantidade de objetos que uma associação admite em cada um de seus papéis, o que é capturado pelas multiplicidades da associação. Esta informação é bastante dependente da natureza do problema e do real significado da associação (o que se quer representar efetivamente), especialmente no que se refere à associação representar apenas o presente ou o histórico (WAZLAWICK, 2004).

Retomemos o exemplo da Figura 6.7, no qual se diz que um empregado está lotado em um departamento e, opcionalmente, pode chefiá-lo. Para definir precisamente as multiplicidades, é necessário investigar os seguintes aspectos: Um empregado pode mudar de lotação? Se sim, é necessário registrar apenas a lotação atual ou é necessário registrar o histórico de lotações dos empregados (ou seja, registrar o evento de lotação de um empregado em um departamento)? Um departamento pode, ao longo do tempo, mudar de chefe? Se sim, é necessário registrar o histórico de chefias do departamento (ou seja, registrar o evento de nomeação do chefe do departamento)?

O modelo da Figura 6.7 representa apenas a situação presente. Se houver mudança de chefe de um departamento ou do departamento de lotação de um empregado, perder-se-á a informação histórica. Na maioria das vezes, essa não é uma solução aceitável. Na maioria dos domínios, as pessoas querem saber a informação histórica. Assim, nota-se que é parte das responsabilidades do sistema registrar a ocorrência dos eventos de nomeação do chefe e de lotação de empregados. Assim, um modelo mais fidedigno a essa realidade é o modelo da Figura 6.10, o qual introduz as classes do tipo "evento lembrado" *NomeacaoChefia* e *Lotacao*.

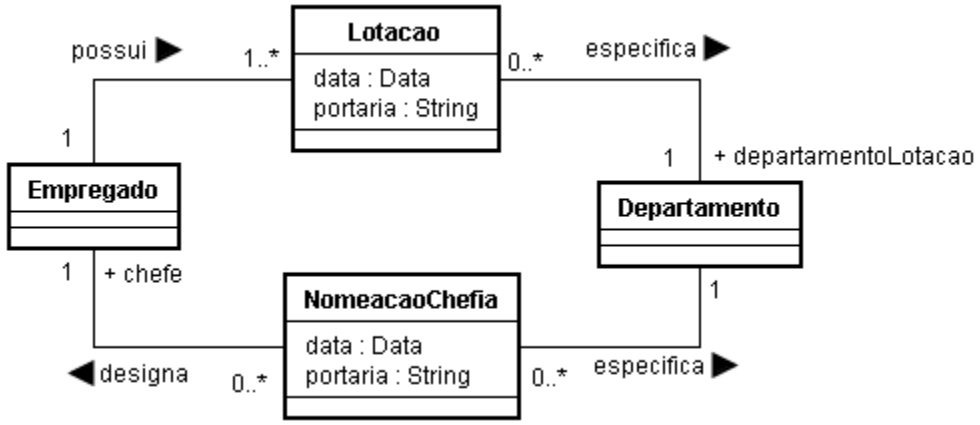


Figura 6.10– Registrando Históricos.

Ainda que este modelo seja mais fidedigno à realidade, ele ainda apresenta problemas. Por exemplo, o modelo diz que um empregado pode ter uma ou mais lotações. Mas o empregado pode ter mais de uma lotação vigente? O mesmo vale para o caso da nomeação de chefia. Um empregado pode ser chefe de mais de um departamento ao mesmo tempo? Um departamento pode ter mais do que um chefe nomeado ao mesmo tempo? Infelizmente, o modelo é incapaz de responder a essas perguntas. Para eliminar essas ambiguidades, é necessário capturar regras de negócio do tipo *restrições de integridade*. No exemplo acima, as seguintes regras se aplicam:

- Um empregado só pode estar lotado em um único departamento em um dado momento.
- Um empregado só pode estar designado como chefe de um único departamento em um dado momento.

- Um departamento só pode ter um empregado designado como chefe em um dado momento.

Observe que, como um departamento pode ter vários empregados nele lotados ao mesmo tempo, não é necessário escrever uma restrição de integridade, pois este é o caso mais geral (sem restrição). Assim, restrições de integridade devem ser escritas apenas para as associações que efetivamente envolvem restrições.

Restrições de integridade são regras de negócio e poderiam ser lançadas no Documento de Definição de Requisitos. Contudo, como elas são importantes para a compreensão e eliminação de ambiguidades do modelo conceitual, é melhor descrevê-las no próprio modelo conceitual.

Além das restrições de integridade relativas às multiplicidades *n*, diversas outras restrições podem ser importantes para tornar o modelo mais fiel à realidade. Ainda no exemplo da Figura 6.10, poder-se-ia querer dizer que um empregado só pode ser nomeado como chefe de um departamento, se ele estiver lotado nesse departamento. Restrições deste tipo são comuns em porções fechadas de um diagrama de classes. Assim, toda vez que se detectar uma porção fechada em um diagrama de classes, vale a pena analisá-la para avaliar se há ali uma restrição de integridade ou não. Havendo, deve-se escrevê-la.

Restrições de integridade também podem falar sobre atributos das classes. Por exemplo, a data da portaria de nomeação de um empregado *e* como chefe de um departamento *d* deve ser igual ou posterior à data da portaria de lotação do empregado *e* no departamento *d*.

Geralmente, restrições de integridade são escritas em linguagem natural, uma vez que não são passíveis de modelagem gráfica. Contudo, conforme já discutido anteriormente, o uso da linguagem natural pode levar a ambiguidades. Visando suprir essa lacuna na UML, o OMG[17] incorporou ao padrão uma linguagem para especificação formal de restrições, a OCL (*Object Constraint Language*). Contudo, restrições escritas em OCL dificilmente serão entendidas por clientes e usuários, o que dificulta a validação das mesmas. Assim, neste texto, sugere-se escrever as restrições de integridade em linguagem natural mesmo.

Vale ressaltar que a UML provê alguns mecanismos para representar restrições de integridade em um modelo gráfico. As próprias multiplicidades são uma forma de capturar restrições de integridade (ditas restrições de integridade de cardinalidade). Além das multiplicidades, a UML provê o recurso de restrições, as quais são representadas entre chaves (**{restrição}**). Restrições podem ser usadas, dentre outros, para restringir a ocorrência de associações. Seja o seguinte exemplo: em uma concessionária de automóveis compras podem ser financiadas ou por financeiras ou por bancos. Para capturar essa restrição, pode-se usar a restrição *xor* da UML, como ilustra a Figura 6.11.

[17] *Object Management Group* (http://www.omg.org/) é uma organização internacional que gerencia padrões abertos relativos ao desenvolvimento orientado a objetos, dentre eles a UML.

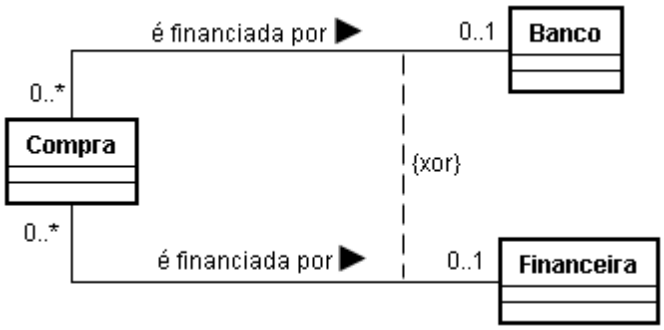


Figura 6.11 – Restrição XOR entre Associações.

Nesta figura, uma compra ou está relacionada a um banco ou a uma financeira. Não é possível que uma compra esteja associada aos dois ao mesmo tempo. Como as multiplicidades mínimas do lado de banco e financeira são zero, uma compra pode não ser financiada.

Ainda em relação às multiplicidades, vale frisar que associações muitos-para-muitos são perfeitamente legais em um modelo orientado a objetos, como ilustra o exemplo da Figura 6.12. Nesse exemplo, está-se dizendo que disciplinas podem possuir vários pré-requisitos e podem ser pré-requisitos para várias outras disciplinas.

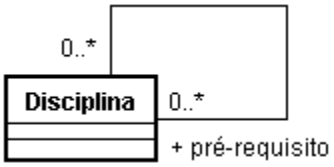


Figura 6.12– Associação Muitos-para-Muitos.

Deve-se observar, no entanto, que muitas vezes, uma associação muitos-para-muitos oculta a necessidade de uma classe do tipo evento a ser lembrado. Seja o seguinte exemplo: em uma organização, empregados são alocados a projetos. Um empregado pode ser alocado a vários projetos, enquanto um projeto pode ter vários empregados a ele alocados. Tomando por base este fato, seria natural se chegar ao modelo da Figura 6.13(a). Contudo, se quisermos registrar as datas de início e fim do período em que o empregado esteve alocado ao projeto, esse modelo é insuficiente e deve ser alterado para comportar uma classe do tipo evento lembrado *Alocacao*, como mostra a Figura 6.13(b).

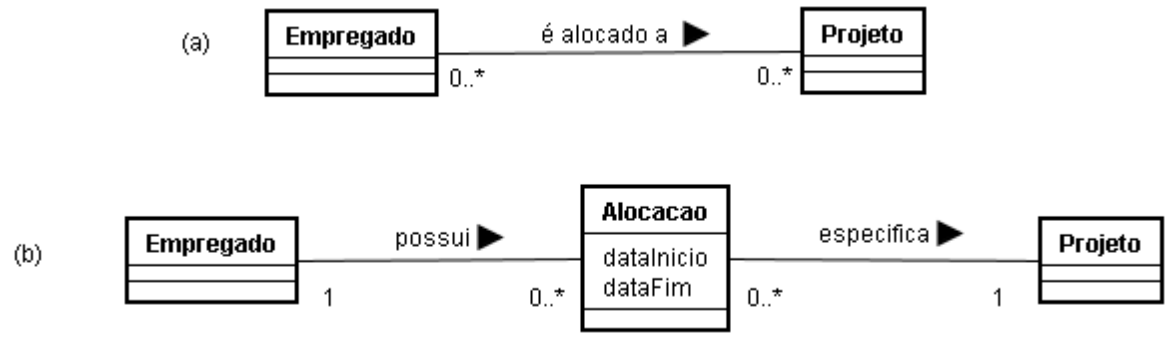


Figura 6.13– Associação Muitos-para-Muitos e Classes de Evento Lembrado.

De fato, o problema por detrás do modelo da Figura 6.13(a) é o mesmo anteriormente discutido na Figura 6.10: a necessidade ou não de se representar informação histórica. Contudo, de maneira mais abrangente, pode-se pensar que, se uma associação apresenta atributos, é melhor tratá-la como uma nova classe. Seja o seguinte exemplo: em uma loja, um cliente efetua um pedido, discriminando vários produtos, cada um deles em uma certa quantidade. O modelo da Figura 6.14(a) procura representar essa situação, mas uma questão permanece em aberto: onde representar a informação da quantidade pedida de cada produto? Essa informação não pode ficar em *Produto*, pois diferentes pedidos pedem quantidades diferentes de um mesmo produto. Também não pode ficar em *Pedido*, pois um mesmo pedido tipicamente especifica diferentes quantidades de diferentes produtos. De fato, quantidade não é nem um atributo da classe *Pedido* nem um atributo da classe *Produto*, mas sim um atributo da associação *especifica*. Assim, a solução consiste em introduzir a classe *ItemPedido*, reificando[18] essa associação, como ilustra a Figura 6.14(b).

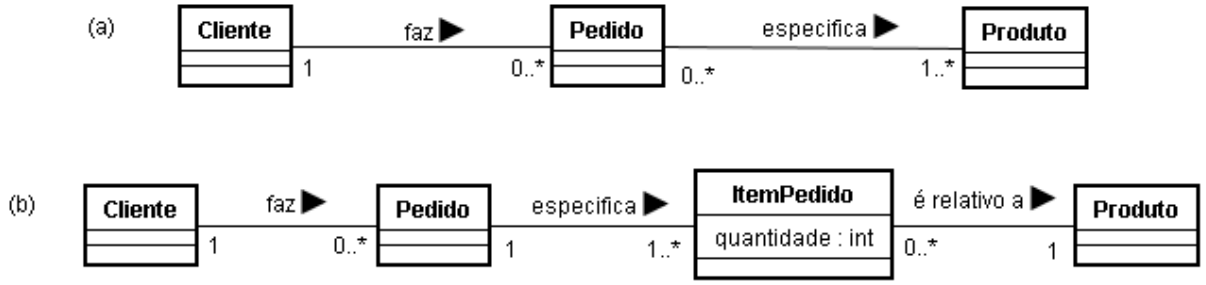


Figura 6.14 – Reificando uma Associação

A UML oferece uma primitiva de modelagem, chamada *classe de associação*, que pode ser usada para reificar associações (OLIVÉ, 2007). Uma classe de associação pode ser vista como uma associação que tem propriedades de classe (BOOCH; RUMBAUGH; JACOBSON, 2006). A Figura 6.15 mostra o exemplo anterior, sendo modelado como uma classe de associação, segundo a notação da UML.

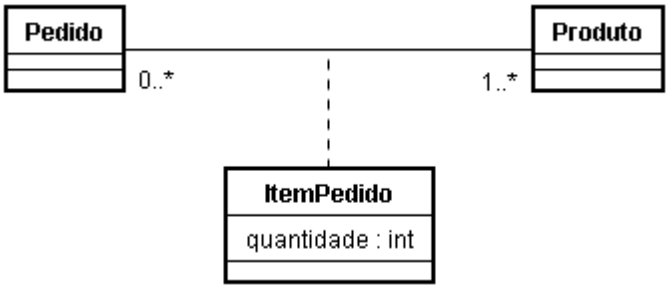


Figura 6.15 – Notação da UML para Classes Associativas.

Classes associativas são ainda representações de associações. Assim como uma instância de uma associação, uma instância de uma classe associativa é um par ordenado conectando duas instâncias das classes envolvidas na associação. Assim, se Pedido100 é uma instância de *Pedido*, Lápis é uma instância de *Produto* e o Pedido100 especifica 5 Lápis, então uma instância de *ItemPedido* é a tupla ((Pedido100, Lápis), 5).

[18] Reificar uma associação consiste em ver essa associação como uma classe. A palavra "reificação" vem da palavra do latim *res*, que significa coisa. Reificação corresponde ao que em linguagem natural se chama nominalização, que basicamente consiste em transformar um verbo em um substantivo (OLIVÉ, 2007).

Classes associativas podem ser usadas também para representar eventos cuja ocorrência precisa ser lembrada, como nos exemplos das figuras 6.10 e 6.13. Entretanto, é importante observar que o uso de classes associativas nesses casos pode levar a problemas de modelagem. Seja o seguinte contexto: em um hospital, pacientes são tratados em unidades médicas. Um paciente pode ser tratado em diversas unidades médicas diferentes, as quais podem abrigar diversos pacientes sendo tratados. A Figura 6.16(a) mostra um modelo que busca representar essa situação usando uma classe associativa. Como uma classe associativa, as instâncias de Tratamento são pares ordenados (Paciente, Unidade Médica). Assim, cada vez que um paciente é tratado em uma unidade médica diferente, tem-se um tratamento. Esta pode, contudo, não ser precisamente a concepção do problema original. Poder-se-ia imaginar que um tratamento é um tratamento de um paciente em várias unidades médicas. A classe de associação não permite representar isso. Assim, um modelo mais fiel ao domínio é aquele que representa Tratamento como uma classe do tipo evento a ser lembrado e que está relacionada com Paciente e Unidade Médica da forma mostrada na Figura 6.16(b).

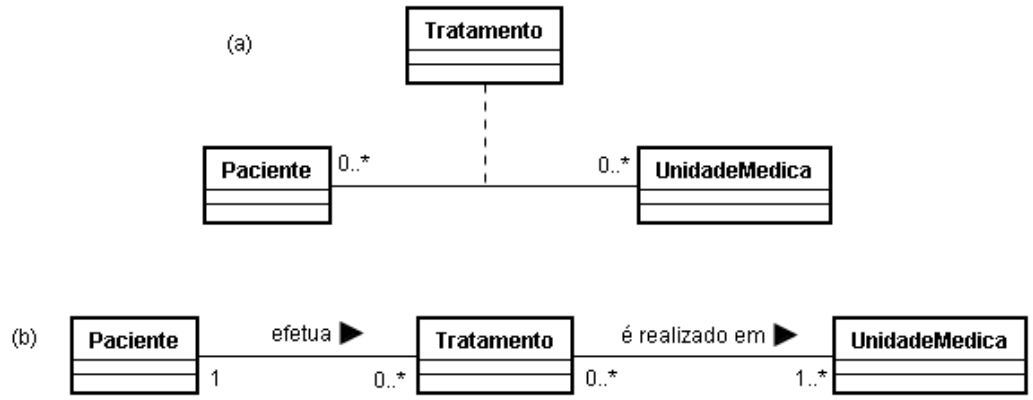


Figura 6.16 – Classes Associativas x Classes do Tipo Evento a Ser Lembrado.

Até o momento, todas as associações mostradas foram associações binárias, i.e., associações envolvendo duas classes. Mesmo o exemplo da Figura 6.12 (Disciplina tem como pré-requisito Disciplina) é ainda uma associação binária, na qual a mesma classe desempenha dois papéis diferentes (disciplina que possui pré-requisito e disciplina que é pré-requisito). Entretanto, associações *n*-árias são também possíveis, ainda que bem menos corriqueiramente encontradas. Uma associação ternária, por exemplo, envolve três classes, como ilustra o exemplo da Figura 6.17. Nesse exemplo, está-se dizendo que fornecedores podem fornecer produtos para certos clientes.

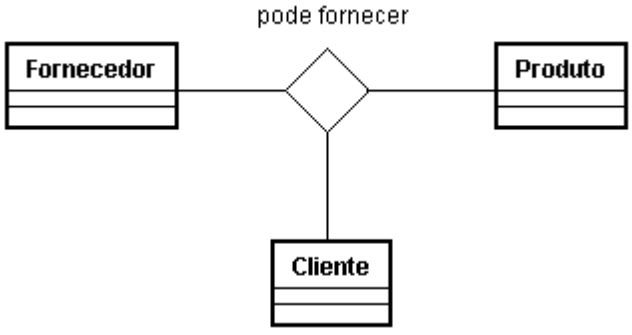


Figura 6.17 – Associação Ternária.

Na UML, associações *n*-árias são mostradas como losangos conectados às classes envolvidas na associação por meio de linhas sólidas, como mostra a Figura 6.17. O nome da

associação é colocado dentro ou em cima do losango, sem direção de leitura. Normalmente, multiplicidades não são mostradas, dada a dificuldade de interpretá-las.

Por fim, uma associação comum entre duas classes representa um relacionamento estrutural entre pares, significando que essas duas classes estão em um mesmo nível, sem que uma seja mais importante do que a outra. Algumas associações, contudo, são consideradas mais fortes, pois indicam que um objeto é composto de outros (relação todo-parte). Para esses casos, a UML considera um tipo especial de associação entre objetos: *agregação*. A agregação é uma forma especial de associação binária que especifica um relacionamento *todo-parte* entre um objeto agregado (o todo) e seus componentes (as partes). Uma agregação pode ser compartilhada ou composta. Na agregação compartilhada, a parte pode ser compartilhada por vários todos. P.ex., uma Comissão (todo) tem vários Membros (parte) e um Membro pode ser parte de mais de uma Comissão. A agregação composta (ou *composição*) é uma forma mais forte de agregação que requer que o objeto parte seja incluído em no máximo um objeto composto por vez. P.ex., um Motor só pode ser parte de um e somente um Carro por vez. Na composição, a parte tem responsabilidade na existência do todo. Se o objeto todo é excluído, todas as suas partes são também excluídas. Note, contudo, que um objeto parte pode ser removido do objeto todo antes que este último seja excluído e, portanto, o objeto parte, nesse caso, não seria excluído junto com o objeto todo (OMG, 2015).

A Figura 6.18 ilustra os casos exemplificados acima. Na Figura 6.18(a), um Carro tem como partes um Motor e quatro ou cinco Rodas. Motor e rodas, ao serem partes de um carro, não podem ser partes de outros carros simultaneamente. Assim, esta é uma relação de composição, a qual impõe que um objeto-parte só pode ser parte de um único todo. Já a agregação não implica nessa exclusividade. O exemplo da Figura 6.15(b) ilustra o caso de comissões compostas por professores. Nesse caso, um professor pode participar de mais de uma comissão simultaneamente e, portanto, trata-se uma relação de agregação. Na UML, um losango branco na extremidade da associação relativa ao todo indica uma agregação. Já um losango preto indica uma composição.

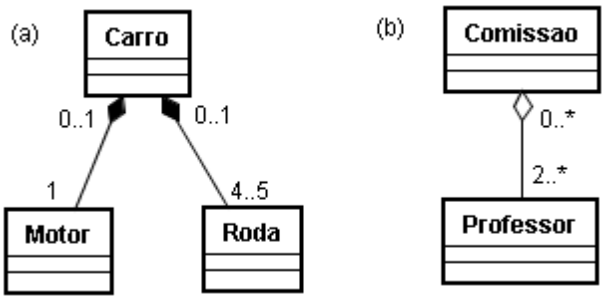


Figura 6.18 – Composição e Agregação.

Relações todo-parte podem ser empregadas em situações como:

- Quando há clareza de que um objeto complexo é composto de outros objetos (componente de). Ex.: Motor é um componente de um carro.
- Para designar membros de coleções (membro de). Ex.: Pesquisadores são membros de Grupos de Pesquisa.

Muitas vezes pode ser difícil perceber a diferença entre uma agregação e uma associação comum. Quando houver essa dúvida, é melhor representar a situação usando uma associação comum, tendo em vista que ela impõe menos restrições.

6.3 – Especificação de Hierarquias de Generalização / Especialização

Um dos principais mecanismos de estruturação de conceitos é a generalização / especialização. Com este mecanismo é possível capturar similaridades entre classes, dispondo-as em hierarquias de classes. No contexto da orientação a objetos, esse tipo de relacionamento é também conhecido como herança.

É importante notar que a herança tem uma natureza bastante diferente das associações. Associações representam possíveis ligações entre instâncias das classes envolvidas. Já a relação de herança é uma relação entre classes e não entre instâncias. Ao se considerar uma classe *B* como sendo uma subclasse de uma classe *A* está-se assumindo que todas as instâncias de *B* são também instâncias de *A*. Assim, ao se dizer que a classe *EstudanteGraduacao* herda da classe *Estudante*, está-se indicando que todos os estudantes de graduação são estudantes. Em resumo, deve-se interpretar a relação de herança como uma relação de subtipo entre classes. A Figura 6.19 mostra a notação da UML para representar herança.

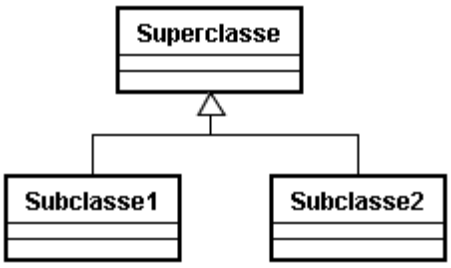


Figura 6.19 – Notação de Herança da UML.

A relação de herança é aplicável quando for necessário fatorar os elementos de informações (atributos e associações) de uma classe. Quando um conjunto de classes possuir semelhanças e diferenças, então elas podem ser organizadas em uma hierarquia de classes, de forma a agrupar em uma superclasse os elementos de informação comuns, deixando as especificidades nas subclasses.

De maneira geral, é desnecessário criar hierarquias de classes quando as especializações (subclasses) não tiverem nenhum elemento de informação diferente. Quando isso ocorrer, é normalmente suficiente criar um atributo *tipo* para indicar os possíveis subtipos da generalização. Seja o caso de um domínio em que se faz distinção entre clientes normais e clientes especiais, dos quais se quer saber exatamente as mesmas informações. Neste caso, criar uma hierarquia de classes, como ilustra a Figura 6.20(a), é desnecessário. Uma solução como a apresentada na Figura 6.20(b), em que o atributo *tipo* pode ser de um tipo enumerado com os seguintes valores {Normal, Especial}, modela satisfatoriamente o problema e é mais simples e, portanto, mais indicada.

Também não faz sentido criar uma hierarquia de classes em que a superclasse não tem nenhum atributo ou associação. Informações de estados pelos quais um objeto passa também não devem ser confundidas com subclasses. Por exemplo, um carro de uma locadora de automóveis pode estar locado, disponível ou em manutenção. Estes são estados e não subtipos de carro.

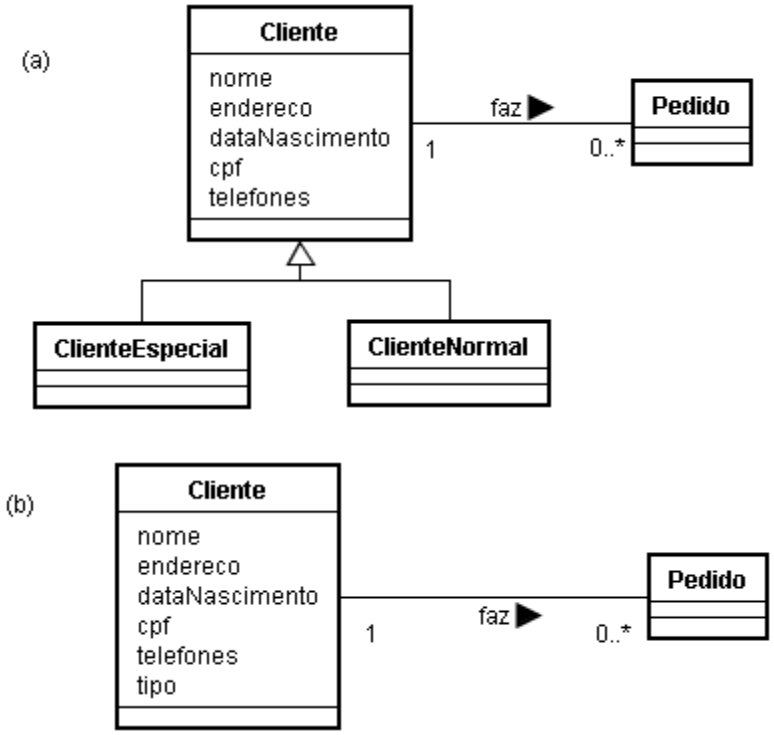


Figura 6.20 – Uso ou não de Herança.

De fato, é interessante considerar alguns critérios para incluir uma subclasse (ou superclasse) em um modelo conceitual. O principal deles é o fato da especialização (ou generalização) estar dentro do domínio de responsabilidade do sistema. Apenas subclasses (superclasses) relevantes para o sistema em questão devem ser consideradas. Além desse critério básico, os seguintes critérios devem ser usados para analisar hierarquias de herança:

- Uma hierarquia de classes deve modelar relações "é-um-tipo-de", ou seja, toda subclasse deve ser um subtipo específico de sua superclasse.
- Uma subclasse deve possuir todas as propriedades (atributos e associações) definidas por suas superclasses e adicionar mais alguma coisa (algum outro atributo, associação ou operação).
- Todas as instâncias de uma subclasse têm de ser também instâncias da superclasse.

Atenção especial deve ser dada à nomeação de classes em uma hierarquia de classes. Cada especialização deve ser nomeada de forma a ser autoexplicativa. Um nome bastante apropriado para a especialização é aquele formado pelo nome de sua superclasse, acompanhado por um qualificador que descreve a natureza da especialização. P.ex., *EstudanteGraduacao* para designar um subtipo de *Estudante*.

Hierarquias de classes não devem ser usadas de forma não criteriosa, simplesmente para compartilhar algumas propriedades. Seja o caso de uma loja de animais, em que se deseja saber as seguintes informações sobre clientes e animais: nome, data de nascimento e endereço. Não faz nenhum sentido considerar que *Cliente* é uma subclasse de *Animal* ou vice-versa, apenas para reusar um conjunto de atributos que, coincidentemente, é igual.

No que se refere à modelagem de superclasses, deve-se observar se uma superclasse é concreta ou abstrata. Se a superclasse puder ter instâncias próprias, que não são instâncias de nenhuma de suas subclasses, então ela é uma classe concreta. Por outro lado, se não for possível

instanciar diretamente a superclasse, ou seja, se todas as instâncias da superclasse são antes instâncias das suas subclasses, então a superclasse é abstrata. Classes abstratas são representadas na UML com seu nome escrito em itálico.

Quando modeladas hierarquias de classes, é necessário posicionar atributos, associações e operações adequadamente. Cada atributo, associação ou operação deve ser colocado na classe mais adequada. Atributos, associações e operações genéricos, que se aplicam a todas as subclasses, devem ser posicionados no topo da estrutura, de modo a serem aplicáveis a todas as especializações. De maneira mais geral, se um atributo, associação ou operação é aplicável a um nível inteiro de especializações, então ele deve ser posicionado na generalização correspondente. Por outro lado, se um atributo, associação ou operação não for aplicável a algumas instâncias, deve-se rever seu posicionamento ou mesmo a estrutura de generalização-especialização adotada.

Inevitavelmente, a designação de atributos e associações a classes conduz a um entendimento mais completo da hierarquia de herança do que era possível em um estágio anterior. Assim, deve-se esperar que o trabalho de reposicionamento de atributos, associações e operações conduza a uma revisão da hierarquia de classes.

Por fim vale a pena mencionar que, durante anos, o mecanismo de herança foi considerado o grande diferencial da orientação a objetos. Contudo, com o passar do tempo, essa ênfase foi perdendo força, pois se percebeu que o uso da herança nem sempre conduz à melhor solução de um problema de modelagem. Hoje a herança é considerada apenas mais uma ferramenta de modelagem, utilizada basicamente para fatorar informações, as quais, de outra forma, ficariam repetidas em diferentes classes (WAZLAWICK, 2004).

Referências do Capítulo

BOOCH, G., RUMBAUGH, J., JACOBSON, I., *UML Guia do Usuário*, 2a edição, Elsevier Editora, 2006.

GUIZZARDI, G., *Ontological Foundations for Structural Conceptual Models*, Telematics Instituut Fundamental Research Series, The Netherlands, 2005.

JACOBSON, I.; *Object-Oriented Software Engineering*, Addison-Wesley, 1992.

MYLOPOULOS, J., "*Conceptual Modeling and Telos*", In: *"Conceptual Modeling, Databases and CASE*", Wiley, 1992.

OLIVÉ, A., *Conceptual Modeling of Information Systems*, Springer, 2007.

OMG, *OMG Unified Modeling Language (OMG UML)*, Version 2.5, March 2015.

WAZLAWICK, R.S., *Análise e Projeto de Sistemas de Informação Orientados a Objetos*, Elsevier, 2004.

Capítulo 7 – Modelagem Dinâmica

Um sistema de informação realiza ações. O efeito de uma ação pode ser uma alteração em sua base de informação e/ou a comunicação de alguma informação ou comando para um ou mais destinatários. Um evento de requisição de ação (ou simplesmente uma requisição) é uma solicitação para o sistema realizar uma ação. O esquema comportamental de um sistema visa especificar essas ações (OLIVÉ, 2007).

Uma parte importante do modelo comportamental de um sistema é o modelo de casos de uso, o qual fornece uma visão das funcionalidades que o sistema deve prover. O modelo conceitual estrutural define os tipos de entidades (classes) e de relacionamentos (atributos e associações) do domínio do problema que o sistema deve representar para poder prover as funcionalidades descritas no modelo de casos de uso. Durante a realização de um caso de uso, atores geram eventos de requisição de ações para o sistema, solicitando a execução de alguma ação. O sistema realiza ações e ele próprio pode gerar outras requisições de ação. É necessário, pois, modelar essas requisições de ações, as correspondentes ações a serem realizadas pelo sistema e seus efeitos. Este é o propósito da modelagem dinâmica.

Em uma abordagem orientada a objetos, requisições de ação correspondem a mensagens trocadas entre objetos. As ações propriamente ditas e seus efeitos são tratados pelas operações das classes[19]. Assim, a modelagem dinâmica está relacionada com as trocas de mensagens entre objetos e a modelagem das operações das classes.

Os diagramas de classes gerados pela atividade de modelagem conceitual estrutural representam apenas os elementos estáticos de um modelo de análise orientada a objetos. É preciso, ainda, modelar o comportamento dinâmico da aplicação. Para tal, é necessário representar o comportamento do sistema como uma função do tempo e de eventos específicos. Um modelo de dinâmico indica como o sistema irá responder a eventos ou estímulos externos e auxilia o processo de descoberta das operações das classes do sistema.

Para apoiar a modelagem da dinâmica de sistemas, a UML oferece três tipos de diagramas (BOOCH; RUMBAUGH; JACOBSON, 2006):

- *Diagrama de Gráfico de Estados*: mostra uma máquina de estados que consiste dos estados pelos quais objetos de uma particular classe podem passar ao longo de seu ciclo de vida e as transições possíveis entre esses estados, as quais são resultados de eventos que atingem esses objetos. Diagramas de gráfico de estados (ou diagramas de transição de estados) são usados principalmente para modelar o comportamento de uma classe, dando ênfase ao comportamento específico de seus objetos.
- *Diagrama de Interação*: descreve como grupos de objetos colaboram em certo comportamento. Diagramas de interação podem ser de dois tipos: diagramas de

[19] De fato, abordagens distintas podem ser usadas, tal como representar tipos de requisições como classes, ditas classes de evento, e os seus efeitos como operações das correspondentes classes de evento, tal como faz Olivé (2007).

comunicação e diagramas de sequência. Um diagrama de sequência é um diagrama de interação que dá ênfase à ordenação temporal das mensagens trocadas por objetos. Já um diagrama de comunicação dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens. Ambos são usados para ilustrar a visão dinâmica de um sistema.

- *Diagrama de Atividades*: mostra o fluxo de uma atividade para outra em um sistema, incluindo sequências e ramificações de fluxo, subatividades e objetos que realizam e sofrem ações. Diagramas de atividades são usados principalmente para a modelagem das funções de um sistema, dando ênfase ao fluxo de controle na execução de um comportamento.

Diagramas de estados focalizam o comportamento de objetos de uma classe específica. Diagramas de interação e de atividades enfocam o fluxo de controle entre vários objetos e atividades. Enquanto os diagramas de interação dão ênfase ao fluxo de controle de um objeto para o outro, os diagramas de atividade dão ênfase ao fluxo de controle de uma etapa (atividade) para outra. Um diagrama de interação observa os objetos que passam mensagens; um diagrama de atividades focaliza as atividades e suas entradas e saídas, i.e., objetos passados de uma atividade para outra (BOOCH; RUMBAUGH; JACOBSON, 2006).

Este capítulo aborda a modelagem dinâmica, discutindo os principais aspectos dessa importante tarefa, quando realizada segundo o paradigma orientado a objetos. São abordados os diagramas de estados e os diagramas de atividades. Diagramas de interação não são discutidos neste texto. A seção 7.1 discute os diferentes tipos de requisições de ações e como os diagramas dinâmicos da UML podem ser usados para modelá-los. A seções 7.2, 7.3 e 7.4 tratam, respectivamente, dos diagramas de transição de estados, diagramas de sequência e diagramas de atividades. A seção 7.4 aborda a especificação de operações. Finalmente, a seção 7.5 explora as relações existentes entre os vários modelos elaborados durante a análise de requisitos, as quais devem ser atentamente avaliadas durante a atividade de verificação de requisitos.

7.1 – Tipos de Requisições de Ação

Um sistema de informação mantém uma representação do estado do domínio em sua base de informações. Esse estado de coisas do domínio em um dado ponto no tempo corresponde ao conjunto de instâncias dos tipos de entidades (classes) e de relacionamentos (associações) relevantes que existem no domínio naquele momento. O esquema estrutural se preocupa com essa perspectiva. Entretanto, o sistema também realiza ações. O efeito de uma ação pode ser uma alteração em sua base de informação e/ou a comunicação de alguma informação ou comando para um ou mais destinatários. Um evento de requisição de ação (ou simplesmente uma requisição) é uma solicitação para que o sistema realize uma ação. Na análise de requisitos, assume-se que a tecnologia é perfeita e, por conseguinte, que o sistema executa as ações requisitadas instantaneamente (OLIVÉ, 2007).

Dependendo de como são iniciadas, requisições podem ser explícitas, temporais ou geradas. Uma *requisição explícita* é iniciada explicitamente por um ator (*requisição externa*) ou por uma outra ação (*requisição induzida*), como parte de seu efeito. Uma *requisição temporal* é iniciada pela passagem do tempo, ocorrendo independentemente do sistema. Por fim, uma *requisição gerada* é iniciada quando uma condição de geração da requisição é satisfeita. O sistema detecta que a condição foi satisfeita e gera a correspondente requisição.

Por exemplo, em um sistema de controle de estoque, uma requisição de compra pode ser gerada quando a quantidade mínima de um produto for atingida (OLIVÉ, 2007).

A maioria das requisições é externa. Dois importantes tipos de requisições externas são notificações de eventos de domínio e consultas. Uma *consulta* é uma requisição externa que provê alguma informação para o ator que iniciou a requisição. Consultas não alteram a base de informações do sistema. Uma *notificação de evento de domínio* é uma requisição externa cujo efeito é uma mudança na base de informações do sistema, correspondendo a um evento de domínio. Nem todas as mudanças na base de informações de um sistema são admissíveis. Os fatos nessa base mudam ao longo do tempo, mas não de maneira arbitrária. Os eventos de domínio definem, exatamente, as mudanças admissíveis. Por meio de notificações de eventos de domínio, atores dizem para o sistema que um evento de domínio ocorreu (OLIVÉ, 2007). Por exemplo, quando ocorre no mundo real o evento de reserva de um carro em uma locadora de automóveis, um usuário, ao realizar o caso de uso correspondente (p.ex., Reservar Carro), está notificando o sistema que esse evento ocorreu, o que inicia uma sequência de ações (os passos do caso de uso), por meio da qual o sistema sabe que o evento ocorreu no domínio.

Um evento de domínio corresponde a um conjunto não vazio de eventos estruturais, percebido ou considerado como uma alteração única no domínio. Um evento estrutural, por sua vez, é uma ação elementar que insere ou remove um fato na base de informações do sistema. Há quatro tipos básicos de eventos estruturais: inserção de entidade, remoção de entidade, inserção de relacionamento e remoção de relacionamento. Esses eventos são ditos estruturais, porque eles são completamente determinados pelo esquema conceitual estrutural e não são explicitamente mostrados no esquema comportamental (OLIVÉ, 2007).

No desenvolvimento orientado a objetos, os eventos estruturais correspondem a operações básicas das classes. Assim, toda classe tem, implicitamente, operações para: criar objetos da classe (evento estrutural de inserção de entidade), dita operação construtora da classe; eliminar objetos (evento estrutural de remoção de entidade), dita operação destruidora da classe; estabelecer ligações e atribuir valores para atributos (eventos estruturais de inserção de relacionamentos); e remover ligações ou excluir valores de atributos (eventos estruturais de remoção de relacionamentos). Essas operações são consideradas básicas e não precisam ser mostradas explicitamente no modelo conceitual.

Seja o exemplo de um sistema de controle de produtos, cujo modelo estrutural é parcialmente apresentado na Figura 7.1. Nesse exemplo, quando a companhia começa a trabalhar com um novo produto (p.ex., *Prod1*), o estado do domínio se altera, caracterizando um evento de domínio. Esse evento de domínio corresponde a cinco eventos estruturais, a saber: (1) a criação do objeto *Prod1*, (2) a atribuição de um valor para o atributo *codigo*, (3) a atribuição de um valor para o atributo *valor*, (4) a atribuição de um valor para o atributo *quantidadeEstoque* e (5) o estabelecimento da associação *fornece* com seu fornecedor. Esse novo produto é considerado um evento de domínio, porque o conjunto de cinco eventos estruturais que o compõe é visto como um evento único. É muito mais fácil para um usuário dizer ao sistema que o evento de domínio ocorreu do que dizer explicitamente cada um dos eventos estruturais.

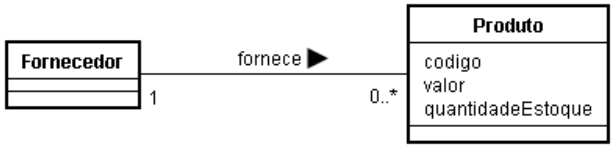


Figura 7.1 – Fragmento do Modelo Estrutural de um Sistema de Controle de Produtos.

Cada evento de domínio possui um conjunto de eventos estruturais, chamado de efeito do evento. A correspondência entre eventos e seus efeitos é dada por uma expressão de mapeamento. O efeito do evento pode ser definido segundo duas abordagens: a abordagem de pós-condição e a abordagem procedimental. Na abordagem de pós-condição, o efeito de um evento é definido por uma condição da base de informações do sistema que deve ser satisfeita após a aplicação do efeito do evento. Na abordagem procedimental, o efeito de um evento é definido por um procedimento, indicando os eventos estruturais que compõem o evento de domínio (OLIVÉ, 2007). Neste texto, enfocamos apenas a abordagem procedimental.

Na abordagem procedimental, quando o paradigma orientado a objetos é adotado, o efeito de um evento é definido como uma operação de uma classe. O corpo dessa operação deve ser tal que sua execução produza o conjunto de eventos estruturais que compõe o evento de domínio. A execução dessa operação vai deixar a base de informações em um novo estado, o qual deve satisfazer a todas as restrições estáticas (definidas no modelo estrutural). Assim, durante a definição da operação correspondente ao efeito de um evento de domínio, devem-se levar em conta as restrições capturadas no modelo estrutural e garantir que o novo estado da base de informações do sistema vai satisfazer a todas elas. Em outras palavras, os eventos de domínio do esquema comportamental devem estar consistentes com o esquema estrutural (OLIVÉ, 2007).

A ideia de efeito de um evento de domínio estende-se, na verdade, para quaisquer requisições. Ou seja, o efeito de uma requisição pode ser representado por meio de uma operação, de maneira análoga ao descrito anteriormente para eventos de domínio.

No que concerne a consultas, na modelagem conceitual define-se apenas o conteúdo de informação das respostas, abstraindo-se detalhes que dizem respeito ao formato e a características de dispositivos de saída (OLIVÉ, 2007). Para que as informações de atributos e associações possam ser recuperadas para serem mostradas como parte das respostas a consultas, as classes precisam prover operações básicas para obter essas informações. Assim como as demais operações básicas, assume-se que toda classe possui implicitamente operações para se obter os valores correntes de seus atributos e associações.

Durante a realização de um caso de uso, atores geram requisições para o sistema, solicitando a execução de ações. O sistema realiza ações e ele próprio pode gerar outras requisições de ação. O conjunto de casos de uso tem de ser consistente com o conjunto de requisições definidas no esquema comportamental do sistema.

7.2 - Diagramas de Gráfico de Estados

Todo objeto tem um tempo de vida. Na criação, o objeto nasce; na destruição, ele deixa de existir. Entre esses dois momentos, um objeto poderá interagir com outros objetos, enviando e recebendo mensagens (BOOCH; RUMBAUGH; JACOBSON, 2006). Essas interações representam o comportamento do objeto e ele pode ser variável ao longo do ciclo de vida do objeto. Ou seja, muitas vezes, o comportamento dos objetos de uma classe depende do estado em que o objeto se encontra em um dado momento. Nestes casos, é útil especificar o comportamento usando uma máquina de estados.

Classes com estados (ou classes modais) são classes cujas instâncias podem mudar de um estado para outro ao longo de sua existência, mudando possivelmente sua estrutura, seus valores de atributos ou comportamento dos métodos (WAZLAWICK, 2004).

Classes modais podem ser modeladas como máquinas de estados finitos. Uma máquina de estados finitos é uma máquina que, em um dado momento, está em um e somente um de um número finito de estados (OLIVÉ, 2007). Os estados de uma máquina de estados de uma classe modal correspondem às situações relevantes em que as instâncias dessa classe podem estar durante sua existência. Um estado é considerado relevante quando ele ajuda a definir restrições ou efeitos dos eventos.

Em qualquer estado, uma máquina de estados pode receber estímulos. Quando a máquina recebe um estímulo, ela pode realizar uma transição de seu estado corrente (dito estado origem) para outro estado (dito estado destino), sendo que se assume que as transições são instantâneas. A definição do estado destino depende do estado origem e do estímulo recebido. Além disso, os estados origem e destino em uma transição podem ser o mesmo. Neste caso, a transição é dita uma autotransição (OLIVÉ, 2007).

Diagramas de Transições de Estados são usados para modelar o comportamento de instâncias de uma classe modal na forma de uma máquina de estados. Todas as instâncias da classe comportam-se da mesma maneira. Em outras palavras, cada diagrama de estados é construído para uma única classe, com o objetivo de mostrar o comportamento ao longo do tempo de vida de seus objetos. Diagramas de estados descrevem os possíveis estados pelos quais objetos da classe podem passar e as alterações dos estados como resultado de eventos (estímulos) que atingem esses objetos. Uma máquina de estado especifica a ordem válida dos estados pelos quais os objetos da classe podem passar ao longo de seu ciclo de vida. A Figura 7.2 mostra a notação básica da UML para diagramas de gráfico de estados.

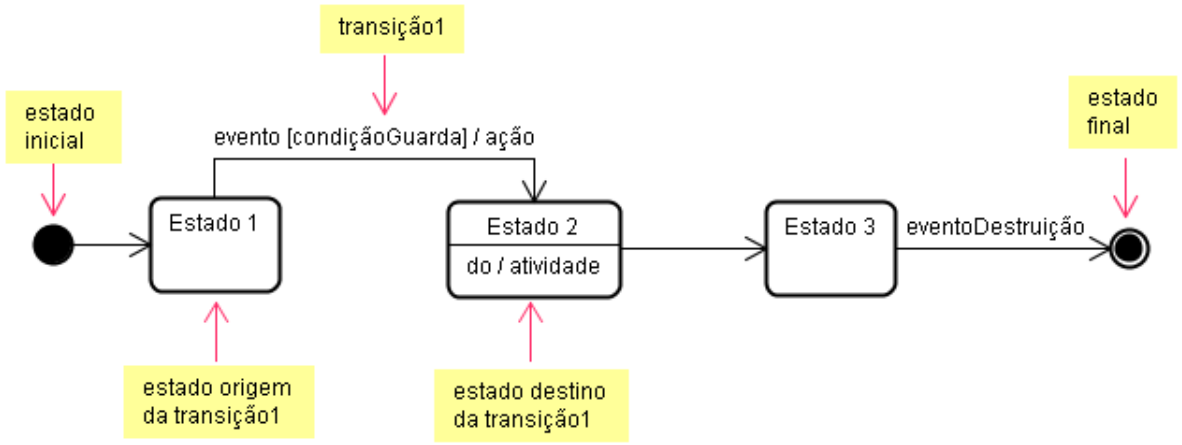


Figura 7.2 - Notação Básica da UML para Diagramas de Gráfico de Estados.

Um estado é uma situação na vida de um objeto durante a qual o objeto satisfaz alguma condição, realiza alguma atividade ou aguarda a ocorrência de um evento (BOOCH; RUMBAUGH; JACOBSON, 2006). Estados são representados por retângulos com os cantos arredondados, sendo que o nome de um estado deve ser único em uma máquina de estados. Uma regra prática para nomear estados consiste em atribuir um nome tal que sejam significativas sentenças do tipo "*o <<objeto>> está <<nome do estado>>*" ou "*o <<objeto>> está no estado <<nome do estado>>*". Por exemplo, em um sistema de locadora de automóveis, um estado possível de objetos da classe *Carro* seria "Disponível". A sentença "o carro está disponível" tem um significado claro (OLIVÉ, 2007).

Quando um objeto fica realizando uma atividade durante todo o tempo em que permanece em um estado, deve-se indicar essa atividade no compartimento de ações do

respectivo estado. É importante realçar que uma atividade tem duração significativa e, quando concluída, tipicamente a conclusão provoca uma transição para um novo estado. A notação da UML para representar atividades de um estado é: **do / <<nomeAtividade>>**.

Transições são representadas por meio de setas rotuladas. Uma transição envolve um estado origem, um estado destino e normalmente um evento, dito o gatilho da transição. Quando a máquina de estados se encontra no estado origem e recebe o evento gatilho, então o evento dispara a transição e a máquina de estados vai para o estado destino. Se uma máquina recebe um evento que não é um gatilho para nenhuma transição, então ela não é afetada pelo evento (OLIVÉ, 2007).

Uma transição pode ter uma condição de guarda associada. Às vezes, há duas ou mais transições com o mesmo estado origem e o mesmo evento gatilho, mas com condições de guarda diferentes. Neste caso, a transição é disparada somente quando o evento gatilho ocorre e a condição de guarda é verdadeira (OLIVÉ, 2007). Quando uma transição não possuir uma condição de guarda associada, então ela ocorrerá sempre que o evento ocorrer.

Por fim, quando uma transição é disparada, uma ação instantânea pode ser realizada. Assim, o rótulo de uma transição pode ter até três partes, todas elas opcionais:

evento [condiçãoGuarda] / ação

Basicamente a semântica de um diagrama de estados é a seguinte: quando o *evento* ocorre, se a *condição de guarda* é verdadeira, a *transição* dispara e a *ação* é realizada instantaneamente. O objeto passa, então, do *estado origem* para o *estado destino*. Se o estado destino possuir uma *atividade* a ser realizada, ela é iniciada.

O fato de uma transição não possuir um evento associado normalmente aponta para a existência de um evento implícito. Isso tipicamente ocorre em três situações: (i) o evento implícito é a conclusão da atividade do estado origem e a transição ocorrerá tão logo a atividade associada ao estado origem tiver sido concluída; (ii) o evento implícito é temporal, sendo disparado pela passagem do tempo; (iii) o evento implícito torna a condição de guarda verdadeira na base de informações do sistema, mas o evento em si não é modelado.

Embora ambos os termos ação e atividade denotem processos, eles não devem ser confundidos. Ações são consideradas processos instantâneos; atividades, por sua vez, estão sempre associadas a estados e têm duração no tempo. Vale a pena observar que, no mundo real, não há processos efetivamente instantâneos. Por mais rápida que seja, uma ação ocorrerá sempre em um intervalo de tempo. Esta simplificação de se considerar ações instantâneas no modelo conceitual pode ser associada à ideia de que a ação ocorre tão rapidamente que não é possível interrompê-la. Em contraste, uma atividade é passível de interrupção, sendo possível, por exemplo, que um evento ocorra, interrompa a atividade e provoque uma mudança no estado do objeto antes da conclusão da atividade.

Às vezes quer se modelar situações em que uma ação instantânea é realizada quando se entra ou sai de um estado, qualquer que seja a transição que o leve ou o retire desse estado. Seja o exemplo de um elevador. Neste contexto, ao parar em um andar, o elevador abre a porta. Suponha que a abertura da porta do elevador seja um processo que não possa ser interrompido e, portanto, que se opte por modelá-lo como uma ação. Essa ação deverá ocorrer sempre que o elevador entrar no estado "*Parado*" e deve ser indicada no compartimento de ações desse estado como sendo uma ação de entrada no estado. A notação da UML para representar ações de entrada em um estado é: **entry / <<nomeAção>>**. Para representar ações de saída de um estado a notação é: **exit / <<nomeAção>>**.

Restam ainda na Figura 7.2 dois tipos especiais de estados: os ditos estados inicial e final. Conforme citado anteriormente, um objeto está sempre em um e somente um estado. Isso implica que, ao ser instanciado, o objeto precisa estar em algum estado. O estado inicial é precisamente esse estado. Graficamente, um estado inicial é mostrado como um pequeno círculo preenchido na cor preta. Seu significado é o seguinte: quando o objeto é criado, ele é colocado no estado inicial e sua transição de saída é automaticamente disparada, movendo o objeto para um dos estados da máquina de estados (no caso da Figura 7.2, para o *Estado1*). Toda máquina de estados tem de ter um (e somente um) estado inicial. Note que o estado inicial não se comporta como um estado normal[20], uma vez que objetos não se mantêm nele por um período de tempo. Ao contrário, uma vez que eles entram no estado inicial, sua transição de saída é imediatamente disparada e o estado inicial é abandonado. A transição de saída do estado inicial tem como evento gatilho implícito o evento responsável pela criação do objeto (OLIVÉ, 2007) e, na UML, esse evento não é explicitamente representado. Estados iniciais têm apenas transições de saída. As transições de saída de um estado inicial podem ter condições de guarda e/ou ações associadas. Quando houver condições de guarda, deve-se garantir que sempre pelo menos uma das transições de saída poderá ser disparada.

Quando um objeto deixa de existir, obviamente ele deixa de estar em qualquer um dos estados. Isso pode ser dito no diagrama por meio de uma transição para o estado final. O estado final indica, na verdade, que o objeto deixou de existir. Na UML um estado final é representado como um círculo preto preenchido com outro círculo não preenchido ao seu redor, como mostra a Figura 7.2. As transições para o estado final definem os estados em que é possível excluir o objeto. Classes cujos objetos não podem ser excluídos, portanto, não possuem um estado final (OLIVÉ, 2007). Assim como o estado inicial, o estado final não se comporta como um estado normal, uma vez que o objeto também não permanece nesse estado (já que o objeto não existe mais). Ao contrário do estado inicial, contudo, uma máquina de estados pode ter vários estados finais. Além disso, deve-se representar o evento que elimina o objeto (na Figura 7.2, *eventoDestruição*).

Os eventos mostrados nas transições são os mesmos eventos de requisição de ação discutidos na Seção 7.1. Contudo, é importante indicar no diagrama de estados os eventos maiores (eventos de domínio e requisições de ações) e não os eventos estruturais que efetivamente alteram o estado do objeto. Assim, neste texto sugere-se indicar como eventos de transições de uma máquina de estados as requisições de realização de casos de uso do sistema (ou de fluxos de eventos específicos, quando um caso de uso tiver mais de um fluxo de eventos normal). Para facilitar a rastreabilidade, sugere-se usar como nome do evento exatamente o mesmo nome do caso de uso (ou do fluxo de eventos). Seja o exemplo de uma locadora de automóveis, que possua, dentre outros, os casos de uso mostrados na Figura 7.3, os quais possuem os fluxos de eventos mostrados nas notas anexadas aos casos de uso.

[20] Por não se comportar como um estado normal, o estado inicial é considerado um pseudoestado no metamodelo da UML.

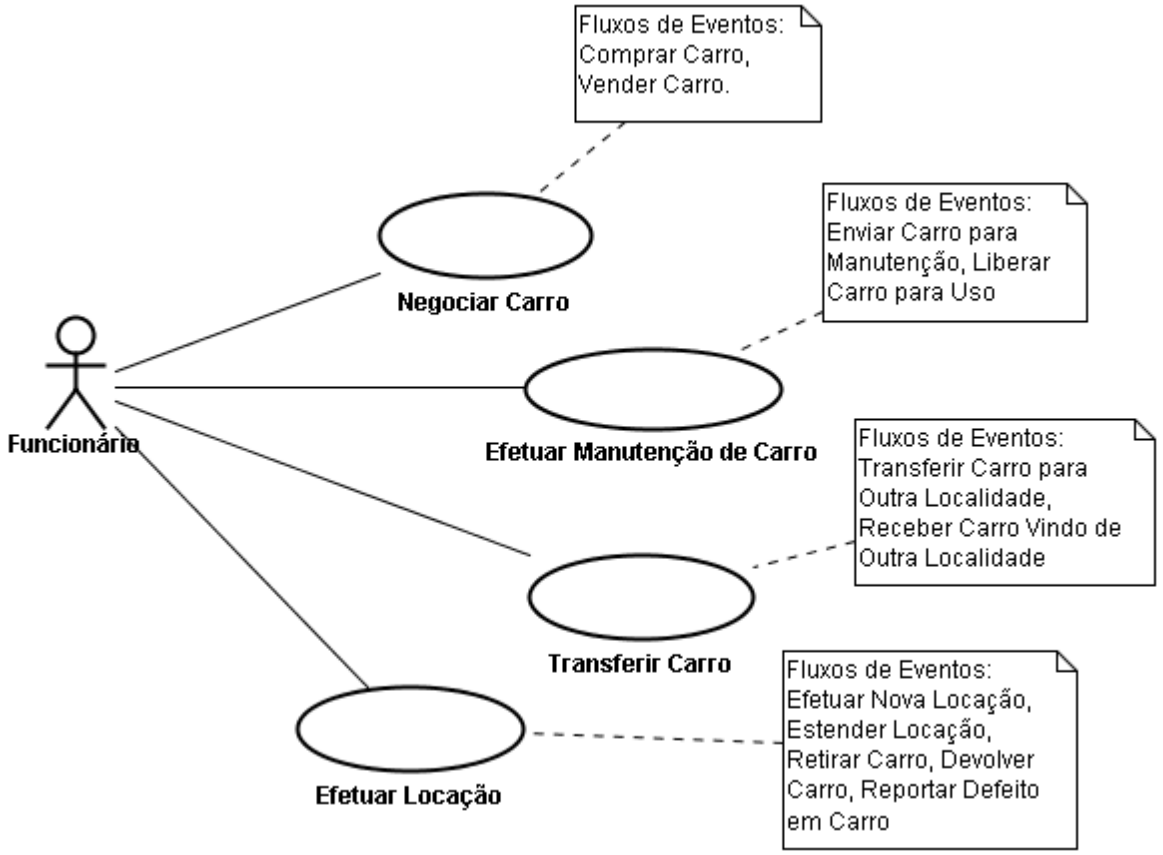


Figura 7.3 – Locadora de Automóveis - Casos de Uso e Fluxos de Eventos Associados.

A classe *Carro* tem o seu comportamento definido pela máquina de estados do diagrama de gráfico de estados da Figura 7.4. Ao ser adquirido (fluxo de eventos *Comprar Carro*, do caso de uso *Negociar Carro*), o carro é colocado *Em Preparação*. Quando liberado para uso (fluxo de eventos *Liberar Carro para Uso*, do caso de uso *Efetuar Manutenção de Carro*), o carro fica *Disponível*. Quando o cliente retira o carro (fluxo de eventos *Retirar Carro*, do caso de uso *Efetuar Locação*), este fica *Em Uso*. Quando é devolvido (fluxo de eventos *Devolver Carro*, do caso de uso *Efetuar Locação*), o carro fica novamente *Em Preparação*. Quando *Disponível*, um carro pode ser transferido de uma localidade para outra (fluxo de eventos *Transferir Carro para Outra Localidade* do caso de uso *Transferir Carro*). Durante o trânsito de uma localidade para outra, o carro está *Em Trânsito*, até ser recebido na localidade destino (fluxo de eventos *Receber Carro Vindo de Outra Localidade*, do caso de uso *Transferir Carro*), quando novamente é colocado *Em Preparação*. Finalmente, carros *Em Preparação* podem ser vendidos (fluxo de eventos *Vender Carro*, do caso de uso *Negociar Carro*), quando deixam de pertencer à locadora e são eliminados de sua base de informações.

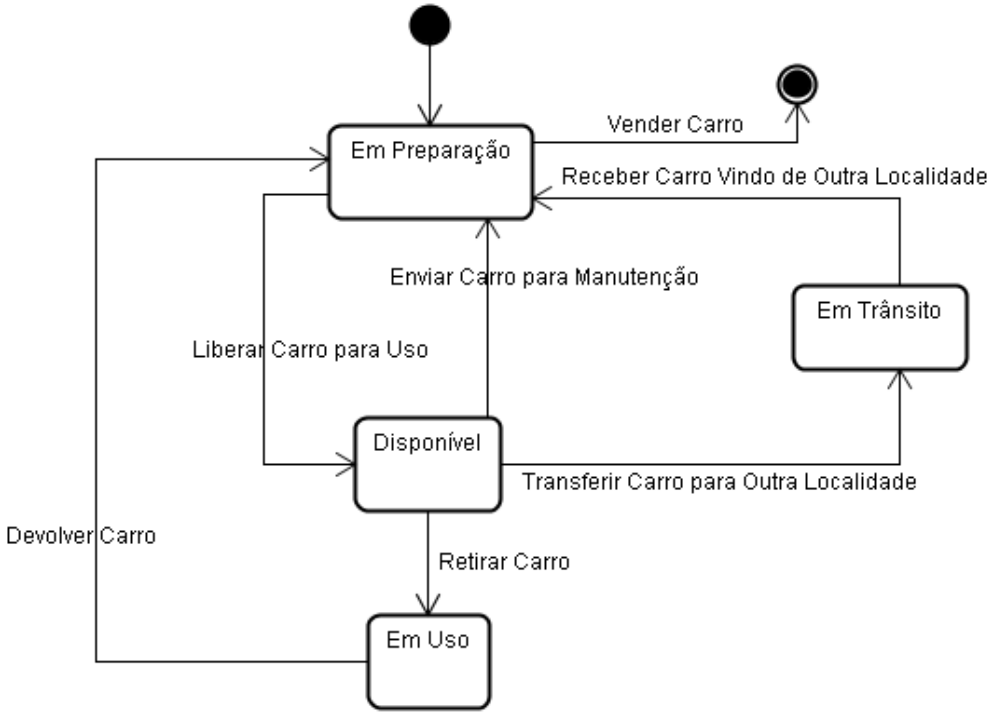


Figura 7.4 – Diagrama de Gráfico de Estados da Classe *Carro* – Disponibilidade (adaptado de (OLIVÉ, 2007)).

Nem todas as classes precisam ser modeladas como máquinas de estados. Apenas classes modais (i.e., classes que apresentam comportamento variável em função do estado de seus objetos) necessitam ser modeladas como máquinas de estados. Além disso, para os diagramas de estados serem efetivamente úteis, recomenda-se modelar uma máquina de estados somente se a classe em questão tiver três ou mais estados relevantes. Se uma classe possuir apenas dois estados relevantes, ainda cabe desenvolver uma máquina de estados. Contudo, de maneira geral, o diagrama tende a ser muito simples e a acrescentar pouca informação relevante que justifique o esforço de elaboração e manutenção do correspondente diagrama. Neste caso, os estados e transições podem ser levantados, sem, no entanto, ser elaborado um diagrama de estados.

Para algumas classes, pode ser útil desenvolver mais do que um diagrama de estados, cada um deles modelando o comportamento dos objetos da classe por uma perspectiva diferente. Em um determinado momento, um objeto está em um (e somente um) estado em cada uma de suas máquinas de estado. Cada diagrama define seu próprio conjunto de estados nos quais um objeto pode estar, a partir de diferentes pontos de vista (OLIVÉ, 2007). Seja novamente o exemplo da classe *Carro*. A Figura 7.4 mostra os possíveis estados de um carro segundo um ponto de vista de disponibilidade. Entretanto, independentemente da disponibilidade, do ponto de vista de possibilidade de negociação, um carro pode estar em dois estados (Não à Venda, À Venda), como mostra a Figura 7.5.

Vale ressaltar que os diferentes diagramas de estados de uma mesma classe não devem ter estados comuns. Cada diagrama deve ter seu próprio conjunto de estados e cada estado pertence a somente um diagrama de estados. Já os eventos podem aparecer em diferentes diagramas de estados, inclusive de classes diferentes. Quando um evento aparecer em mais de um diagrama de estados, sua ocorrência vai disparar as correspondentes transições em cada uma das máquinas de estados em que ele aparecer (OLIVÉ, 2007).

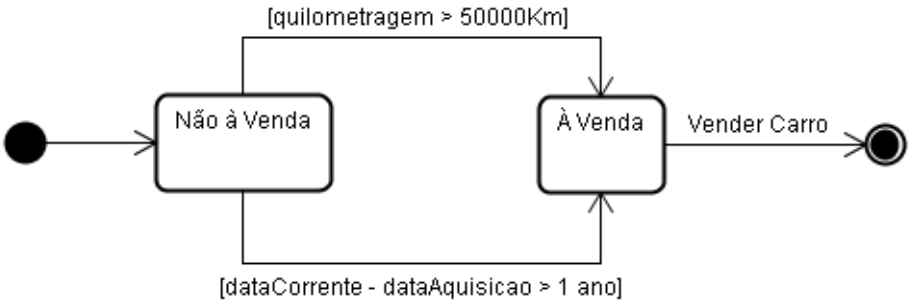


Figura 7.5 – Diagrama de Gráfico de Estados da Classe *Carro* – Possibilidade de Negociação (adaptado de (OLIVÉ, 2007)).

A Figura 7.5 mostra duas transições em que os eventos não são declarados explicitamente. No primeiro caso (quilometragem > 50.000Km), o evento implícito torna a condição de guarda verdadeira na base de informações do sistema. Esse evento corresponde ao registro no sistema de qual é a quilometragem corrente do carro. Caso esse registro ocorra sempre no ato da devolução do carro pelo cliente (fluxo de eventos *Devolver Carro*, do caso de uso *Efetuar Locação*) e/ou no ato do recebimento do carro vindo de outra localidade (fluxo de eventos *Receber Carro Vindo de Outra Localidade*, do caso de uso *Transferir Carro*), esses eventos poderiam ser explicitamente declarados. Contudo, se o registro pode ocorrer em vários eventos diferentes, é melhor deixar o evento implícito. O segundo caso (dataCorrente – dataAquisicao > 1 ano) trata-se de um evento temporal, disparado pela passagem do tempo.

Todos os estados mostrados até então são estados simples, i.e., estados que não possuem subestados. Entretanto, há também estados compostos, os quais podem ser decompostos em um conjunto de subestados disjuntos e mutuamente exclusivos e um conjunto de transições (OLIVÉ, 2007). Um subestado é um estado aninhado em outro estado. O uso de estados compostos e subestados é bastante útil para simplificar a modelagem de comportamentos complexos. Seja o exemplo da Figura 7.4, que trata da disponibilidade de um carro. Suponha que seja necessário distinguir três subestados do estado *Em Uso*, a saber: *Em Uso Normal*, quando o carro não está quebrado nem em atraso; *Quebrado*, quando o cliente reportar um defeito no carro; e *Em Atraso*, quando o carro não foi devolvido na data de devolução prevista e não está quebrado. A Figura 7.6 mostra a máquina de estados da classe *Carro* considerando, agora, que, quando um carro está em uso, ele pode estar nesses três subestados.

Nesse diagrama, não está sendo mostrado que os estados *Em Uso Normal*, *Em Atraso* e *Quebrado* são, de fato, subestados do estado *Em Uso* e, portanto, transições comuns (por exemplo, aquelas provocadas pelo evento *Devolver Carro*) são repetidas. Isso torna o modelo mais complexo e fica claro que esta solução representando diretamente os subestados (e omitindo o estado composto) não é escalável para sistemas que possuem muitos subestados, levando a diagramas confusos e desestruturados (OLIVÉ, 2007). A Figura 7.7 mostra uma solução mais indicada, em que tanto o estado composto quanto seus subestados são mostrados no mesmo diagrama. Uma outra opção é ocultar a decomposição do estado composto, mantendo o diagrama como o mostrado na Figura 7.4, e mostrar essa decomposição em um diagrama de estados separado.

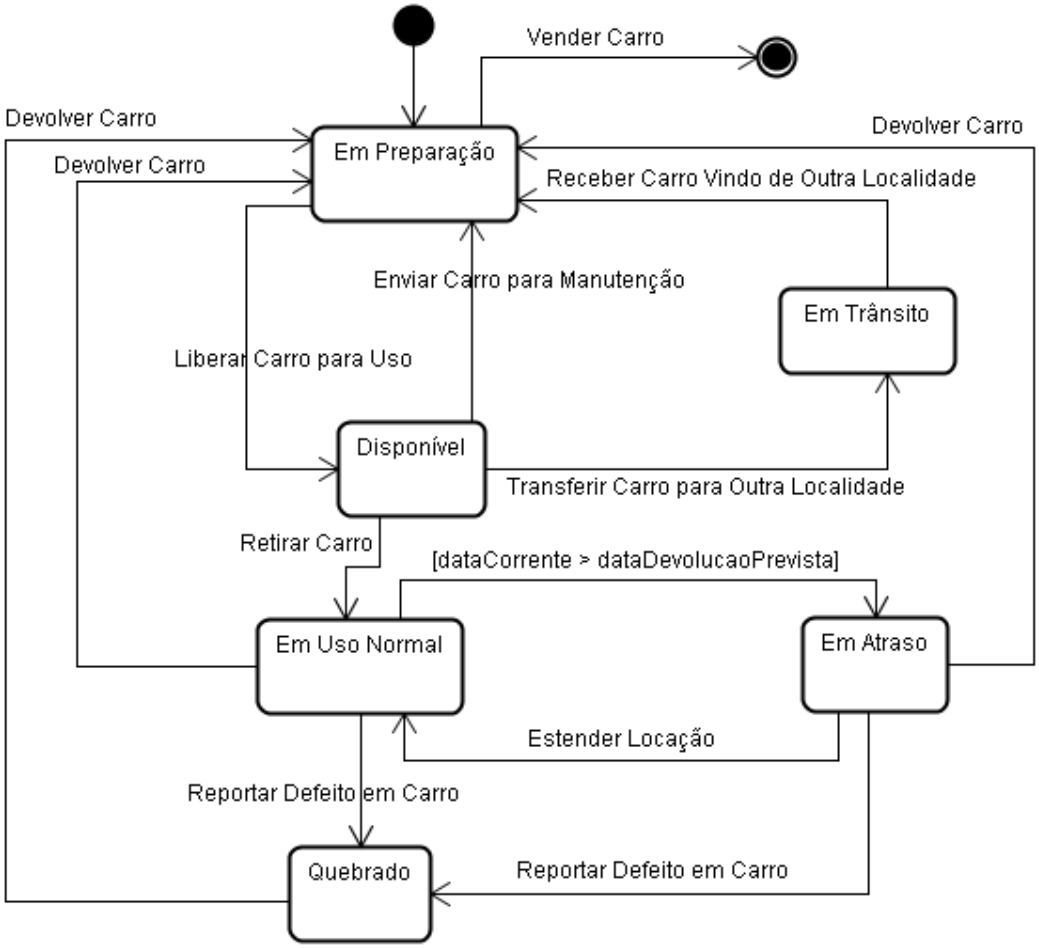


Figura 7.6 – Diagrama de Estados da Classe *Carro* (Disponibilidade) com Subestados de *Em Uso* (adaptado de (OLIVÉ, 2007)).

Se um objeto está em um estado composto, então ele deve estar também em um de seus subestados. Assim, um estado composto pode possuir um estado inicial para indicar o subestado padrão do estado composto, como representado na Figura 7.7. Entretanto, deve-se considerar que as transições podem começar e terminar em qualquer nível. Ou seja, uma transição pode ir (ou partir) diretamente de um subestado (OLIVÉ, 2007). Assim, uma outra opção para o diagrama da Figura 7.7 seria fazer a transição nomeada pelo evento *Retirar Carro* chegar diretamente ao subestado *Em Uso Normal*, ao invés de chegar ao estado composto *Em Uso*.

O estado de um objeto deve ser mapeado no modelo estrutural. De maneira geral, o estado pode ser modelado por meio de um atributo. Esse atributo deve ser monovalorado e obrigatório. O conjunto de valores possíveis do atributo é o conjunto dos estados possíveis, conforme descrito pela máquina de estados (OLIVÉ, 2007). Assim, é bastante natural que o tipo de dados desse atributo seja definido como um tipo de dados enumerado. Um nome adequado para esse atributo é "estado". Contudo, outros nomes mais significativos para o domínio podem ser atribuídos. Em especial, quando uma classe possuir mais do que uma máquina de estado e, por conseguinte, mais do que um atributo de estado for necessário, o nome do atributo de estado deve indicar a perspectiva capturada pela correspondente máquina de estados.

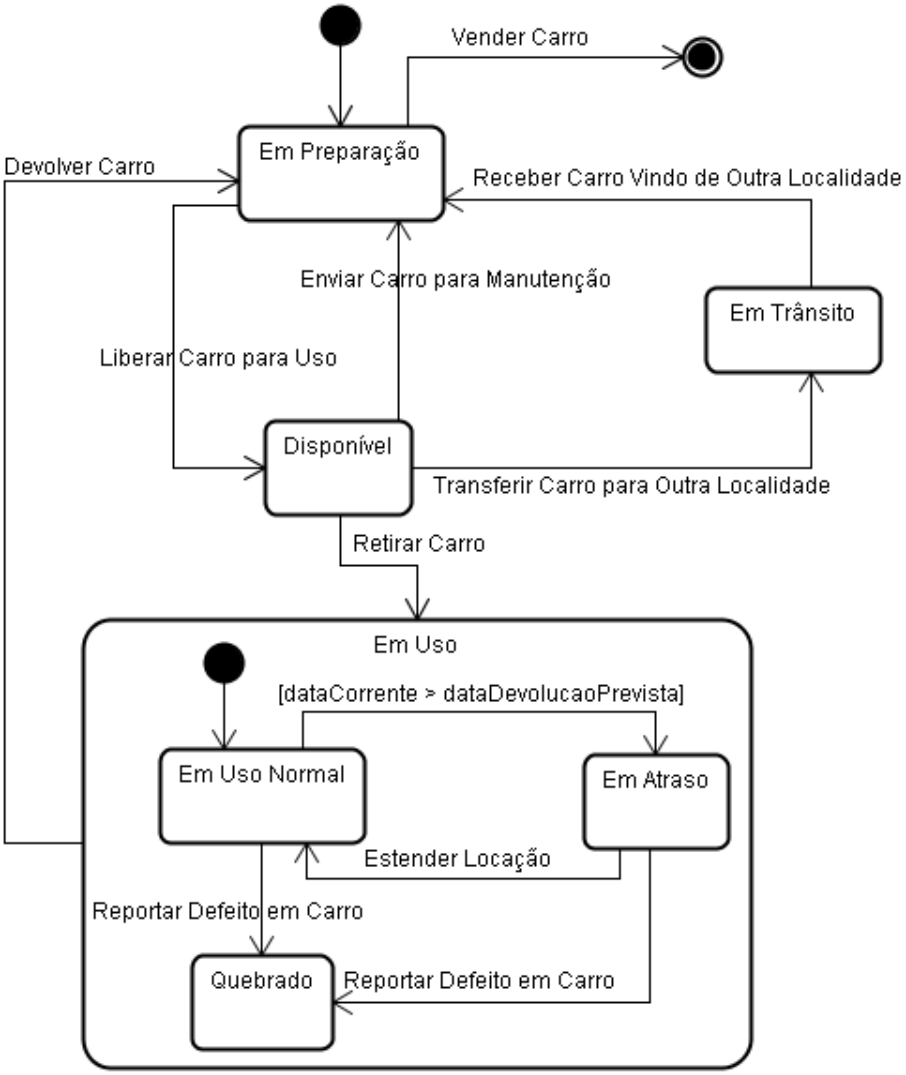


Figura 7.7 – Diagrama de Estados da Classe *Carro* (Disponibilidade) com Estado Composto *Em Uso* (adaptado de (OLIVÉ, 2007)).

É interessante observar que algumas transições podem mudar a estrutura da classe. Quando os diferentes estados de um objeto não afetam a sua estrutura, mas apenas, possivelmente, os valores de seus atributos e associações, diz-se que a transição é estável e os diferentes estados podem ser mapeados para um simples atributo (WAZLAWICK, 2004), conforme discutido anteriormente.

Entretanto, há situações em que, conforme um objeto vai passando de um estado para outro, ele vai ganhando novos atributos ou associações, ou seja, há uma mudança na estrutura da classe. Seja o exemplo de uma locação de carro. Como mostra a Figura 7.8, quando uma locação é criada, ela está ativa, em curso normal. Quando o carro não é devolvido até a data de devolução prevista, a locação passa a ativa com prazo expirado. Se a locação é estendida, ela volta a ficar em curso normal. Quando o carro é devolvido, a locação fica pendente. Finalmente, quando o pagamento é efetuado, a locação é concluída.

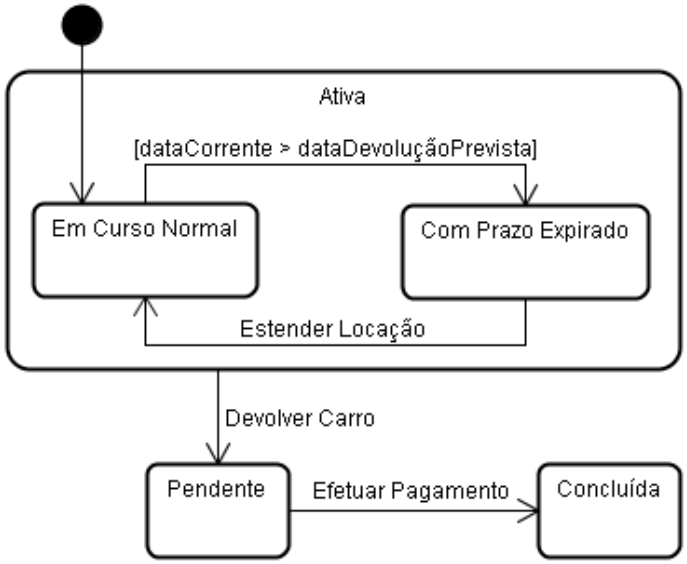


Figura 7.8 – Diagrama de Estados da Classe *Locação*.

Locações ativas (e em seus subestados, obviamente) têm como atributos: data de locação, data de devolução prevista, valor devido e caução. Quando uma locação vai para o estado pendente, é necessário registrar a data de devolução efetiva e os problemas observados no carro devolvido. Finalmente, quando o pagamento é efetuado, é preciso registrar a data do pagamento, o valor e a forma de pagamento. Diz-se que as transições dos estados de *Ativa* para *Pendente* e de *Pendente* para *Concluída* são monotônicas[21], porque a cada mudança de estado, novos relacionamentos (atributos ou associações) são acrescentados (mas nenhum é retirado).

Uma solução frequentemente usada para capturar essa situação no modelo conceitual estrutural consiste em criar uma única classe (*Locacao*) e fazer com que certos atributos sejam nulos até que o objeto mude de estado, como ilustra a Figura 7.9. Essa forma de modelagem, contudo, pode não ser uma boa opção, uma vez que gera classes complexas com regras de consistência que têm de ser verificadas muitas vezes para evitar a execução de um método que atribui um valor a um atributo específico de um estado (WAZLAWICK, 2004), tal como *dataPagamento*.

Figura 7.9 – Classe *Locação* com atributos inerentes a diferentes estados.

[21] Monotônico diz respeito a algo que ocorre de maneira contínua. Neste caso, a continuidade advém do fato de um objeto continuamente ganhar novos atributos e associações, sem perder os que já possuía.

É possível modelar essa situação desdobrando o conceito original em três: um representando a locação efetivamente, outro representando a devolução e outro representando o pagamento. Desta forma, capturam-se claramente os eventos de locação, devolução e pagamento, colocando as informações de cada evento na classe correspondente, como ilustra a Figura 7.10.

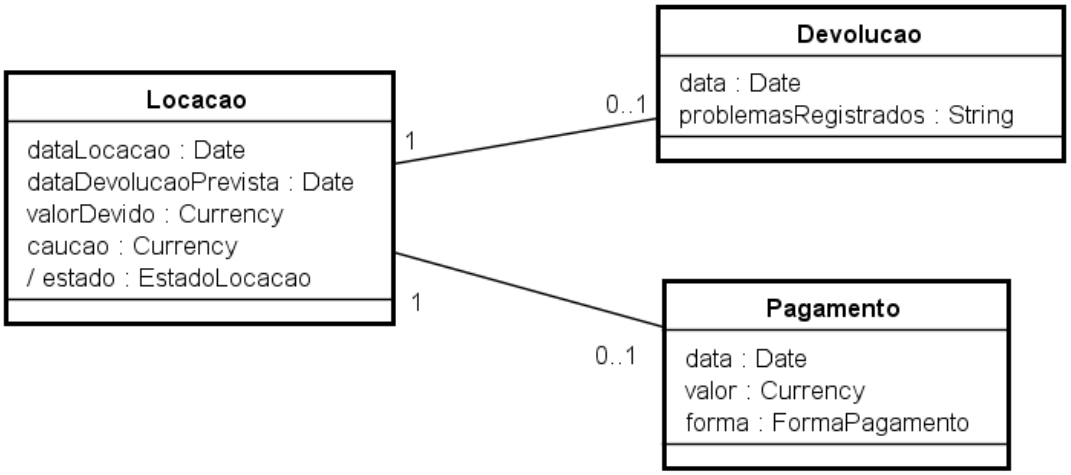


Figura 7.10 – Distribuindo as responsabilidades.

Finalmente, vale a pena comentar que estados de uma classe modal podem ser tratados por meio de operações ao invés de atributos. Seja o exemplo anterior de locações de carros (Figura 7.10). O estado de uma locação pode ser computado a partir dos atributos e associações da classe *Locacao*, sem haver a necessidade de um atributo *estado*. Se uma locação não tem uma devolução associada, então ela está ativa. Estando ativa, se a data corrente é menor ou igual à data de devolução prevista, então a locação está em curso normal; caso contrário, ela está com prazo expirado. Se uma locação possui uma devolução, mas não possui um pagamento associado, então ela está pendente. Finalmente, se a locação possui um pagamento associado, então ela está concluída. Em casos como este, pode-se optar por tratar estado como uma operação e não como um atributo. Opcionalmente, pode-se utilizar a operação para calcular o valor de um atributo derivado[22] *estado*. Atributos derivados são representados na UML precedidos por uma barra (no exemplo, */estado*).

7.3 - Diagramas de Atividades

Um diagrama de atividades é como um fluxograma, no sentido que focaliza o fluxo de controle de uma atividade para outra. Entretanto, ao contrário de um fluxograma tradicional, um diagrama de atividades mostra, além de fluxos sequenciais e ramificações de controle, fluxos concorrentes (BOOCH; RUMBAUGH; JACOBSON, 2006; BLAHA; RUMBAUGH, 2006). O propósito de um diagrama de atividades é mostrar as etapas de um processo complexo e a sequência entre elas (BLAHA; RUMBAUGH, 2006).

Diagramas de atividades são usados para representar processos, sendo utilizados tanto para modelar processos de negócio quanto para representar a realização de um caso de uso. Eles foram adicionados à UML relativamente tarde, adquirindo status de um tipo de diagrama

[22] Um atributo é derivado quando seu valor pode ser deduzido ou calculado a partir de outras informações (atributos e associações) já existentes no modelo estrutural.

somente na UML 2.0. Até a UML 1.5, diagramas de atividades eram considerados um tipo especial de diagrama de estados.

Os diagramas de atividades são muito usados para modelar processos de negócio e fluxos de trabalho em organizações (ver Seção 3.3), uma vez que esses processos / fluxos de trabalho envolvem muitas pessoas e unidades organizacionais que realizam atividades concorrentemente (BLAHA; RUMBAUGH, 2006). Principalmente no contexto de sistemas corporativos e de missão crítica, o sistema em desenvolvimento estará funcionando no contexto de processos de negócio de mais alto nível e pode ser útil usar diagramas de atividades para modelar esses processos com o intuito de investigar as formas como humanos e os vários sistemas automatizados colaboram (BOOCH; RUMBAUGH; JACOBSON, 2006). Esse importante uso dos diagramas de atividades, contudo, está fora do escopo deste texto e, portanto, não será aqui abordado.

No contexto da modelagem comportamental de sistemas (foco deste texto), os diagramas de atividades podem ser usados para modelar o fluxo de trabalho em um caso de uso. Para essa finalidade, os principais elementos de modelo dos diagramas de atividades da UML utilizados são: atividades, fluxos de controle, pontos de iniciação e conclusão, desvios (ou ramificações), bifurcação e união, fluxos de objetos e regiões de expansão. Na modelagem de processos, utilizam-se também raias para indicar quem (que pessoa ou unidade organizacional) é responsável por uma atividade.

Uma atividade é uma porção significativa de trabalho dentro de um fluxo de trabalho. Atividades podem ser atômicas ou complexas. Uma atividade atômica (i.e., que não pode ser decomposta) é dita uma ação na UML. Uma atividade complexa é composta de outras atividades (atômicas ou complexas) e na UML é representada por um nó de atividade. Assim, um nó de atividade representa um grupo de ações ou de outros nós de atividade aninhados, que possui uma subestrutura visível, representada em um outro diagrama de atividades (BOOCH; RUMBAUGH; JACOBSON, 2006). Atividades são representadas por elipses alongadas. Quando uma atividade é concluída, o fluxo de controle passa imediatamente para a atividade seguinte. O fluxo de controle é mostrado por meio de uma seta não rotulada de uma atividade para a sua sucessora.

O fluxo de controle inicia e termina em algum lugar. Os pontos de iniciação e de conclusão do fluxo de controle são mostrados em um diagrama de atividades usando a mesma notação de estados inicial e final de diagramas de gráficos de estados, respectivamente. Quando um diagrama de atividades é ativado, o fluxo de controle inicia no ponto de iniciação e avança por meio da(s) seta(s) de fluxo de controle em direção à(s) primeira(s) atividade(s) a ser(em) realizada(s). Quando o ponto de conclusão é atingido, todo o processo é encerrado e a execução do diagrama de atividades termina (BOOCH; RUMBAUGH; JACOBSON, 2006; BLAHA; RUMBAUGH, 2006). A Figura 7.11 mostra a notação da UML para atividades, fluxos de controle e pontos de iniciação e conclusão.

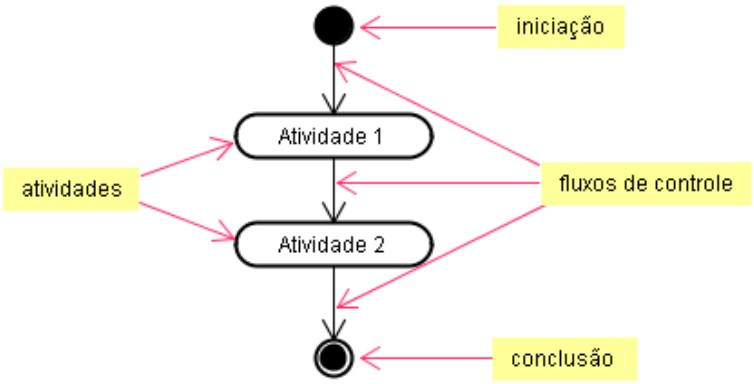


Figura 7.11 - Notação Básica da UML para Diagramas de Atividades.

Os elementos da Figura 7.11 só permitem modelar sequências de atividades. Contudo, a maioria dos fluxos de trabalho envolve fluxos alternativos, concorrentes e/ou iterativos. Para representar essas estruturas de controle, são necessários outros elementos de modelo. Em diagramas de atividades da UML, fluxos alternativos, concorrentes e/ou iterativos podem ser representados por meio de desvios (ou ramificações), bifurcações e uniões, e regiões de expansão, respectivamente.

Um desvio ou ramificação permite especificar caminhos alternativos a serem seguidos, tomando por base alguma expressão booleana. Uma ramificação possui um fluxo de entrada e dois ou mais de saída. Em cada fluxo de saída é colocada uma expressão booleana, a qual é avaliada quando o controle atinge a ramificação. As condições não podem se sobrepor, pois senão o fluxo de controle poderia seguir por mais de um caminho, o que não é admissível em uma ramificação. Além disso, elas têm de cobrir todas as possibilidades, pois, caso contrário o fluxo de controle pode ficar sem ter para onde seguir em alguma situação. Para evitar esse problema, pode-se utilizar a condição *else*, a qual é satisfeita caso nenhuma outra condição seja satisfeita (BOOCH; RUMBAUGH; JACOBSON, 2006; BLAHA; RUMBAUGH, 2006).

Uma ramificação é representada na UML por um losango. Quando dois caminhos de controle alternativos se fundem novamente, pode-se utilizar o mesmo símbolo para mesclá-los. Na fusão, contudo, há duas ou mais setas de fluxo de controle de entrada e somente uma de saída. A Figura 7.12 ilustra a notação de desvios e fusões.

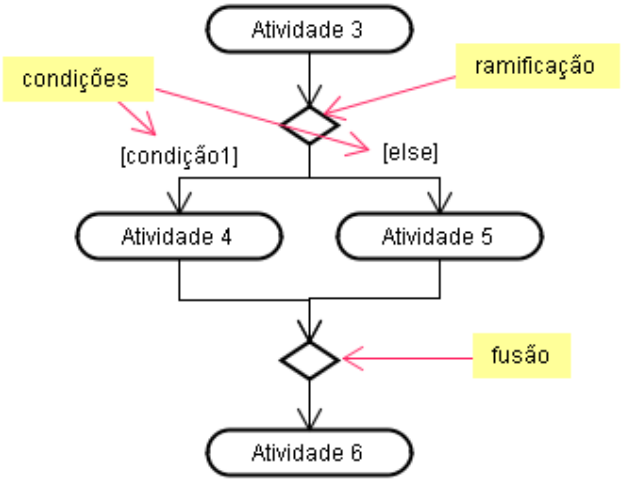


Figura 7.12 – Ramificações em Diagramas de Atividades da UML.

Para mostrar atividades realizadas concorrentemente, podem-se utilizar bifurcações e uniões, as quais são representadas por barras de sincronização. Uma barra de sincronização é representada por uma linha grossa horizontal ou vertical. Uma bifurcação tem de ter um único fluxo de controle de entrada e dois ou mais fluxos de saída, cada um deles representando um fluxo de controle independente. As atividades associadas a cada um desses caminhos prosseguem paralelamente, indicando que as atividades ocorrem concorrentemente. Já uma união representa a sincronização de um ou mais fluxos de controle concorrentes. Uma união tem de ter dois ou mais fluxos de entrada e apenas um fluxo de controle de saída. Na união, os fluxos concorrentes de entrada são sincronizados, significando que cada um deles aguarda até que todos os fluxos de entrada tenham atingido a união, a partir da qual o fluxo de controle de saída prossegue. De maneira geral, deve haver um equilíbrio entre bifurcações e uniões, indicando que o número de fluxos de controle que deixam uma bifurcação deve ser equivalente ao número de fluxos que chegam a uma união correspondente (BOOCH; RUMBAUGH; JACOBSON, 2006). A Figura 7.13 ilustra a notação de bifurcações e uniões.

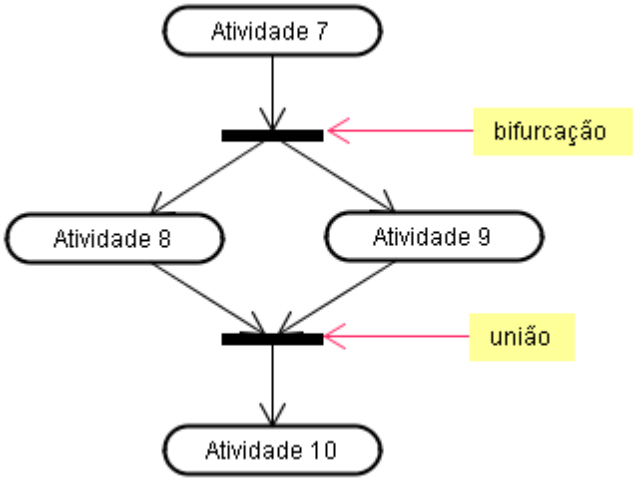


Figura 7.13 – Bifurcações e Uniões em Diagramas de Atividades da UML.

Para representar atividades que ocorrem várias vezes, operando sobre elementos de um conjunto, podem-se utilizar regiões de expansão. Uma região de expansão representa um fragmento do diagrama de atividades que é realizado operando sobre os elementos de uma lista ou conjunto. Ela é representada por uma linha tracejada em torno da região do diagrama que envolve as atividades iterativas. A região é executada uma vez para cada elemento do conjunto de entrada (BOOCH; RUMBAUGH; JACOBSON, 2006).

Muitas vezes é útil representar os objetos requeridos (entradas) e produzidos (saídas) por uma atividade em um diagrama de atividades. É possível especificar os objetos envolvidos nas atividades, conectando-os às atividades que os produzem ou consomem. Além de representar objetos e o seu fluxo nas atividades, pode-se representar, ainda, o estado em que se encontra o objeto. A Figura 7.14 mostra a notação de objetos, fluxos de objetos e estado do objeto.

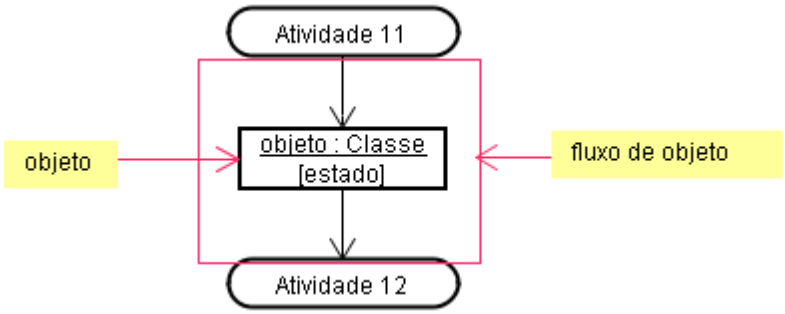


Figura 7.14 – Objetos e Fluxos de Objetos em Diagramas de Atividades da UML.

Um fluxo de objetos implica em um fluxo de controle, pois representa o fluxo de um objeto de uma atividade para outra. Portanto, é desnecessário desenhar um fluxo de controle entre as atividades conectadas por fluxos de objetos (BOOCH; RUMBAUGH; JACOBSON, 2006).

Neste texto, defendemos o uso de diagramas de atividades para complementar a visão comportamental de casos de uso complexos. Sugere-se elaborar um diagrama de atividades para cada fluxo de eventos normal complexo, mostrando no mesmo diagrama o fluxo de eventos normal e os correspondentes fluxos variantes e de exceção. Elaborar diagramas de atividade para casos de uso simples, tais como casos de uso cadastrais e de consulta, dificilmente adicionará algum valor ao projeto e, portanto, tais diagramas são dispensáveis.

Para ajudar os usuários a entender um diagrama de atividades, pode ser útil ilustrar a execução do mesmo usando fichas de atividade. Uma ficha é colocada no ponto de iniciação do diagrama e ela vai sendo deslocada pelas atividades do diagrama. Quando houver uma bifurcação, múltiplas fichas vão surgir, uma para cada fluxo de saída. De maneira análoga, uma união do controle reduz o conjunto de fichas de entrada para uma única ficha de saída (BLAHA; RUMBAUGH, 2006).

7.4 – Especificação das Operações

Uma vez estudado o comportamento do sistema, tem-se uma base para a definição das operações das classes que compõem o sistema. Operações correspondem a serviços que podem ser solicitados aos objetos de uma classe e são apresentadas na seção inferior do símbolo de classe, com a seguinte sintaxe[23] para a sua assinatura (BOOCH; RUMBAUGH; JACOBSON, 2006):

visibilidade nome(lista_de_parâmetros): tipo_de_retorno

A visibilidade de uma operação indica em que situações ela é visível por outras classes, podendo uma operação ser pública, protegida, privada e de pacote, da mesma forma que atributos (ver Seção 6.2.1). A informação de visibilidade é inerente à fase de projeto e não deve ser expressa em um modelo conceitual.

Para nomear uma operação, sugere-se o uso de um verbo no infinitivo, o qual pode ser combinado com complementos, omitindo-se preposições. A exceção fica por conta de operações com retorno booleano, para as quais se sugere usar um nome que dê uma noção de

[23] A UML admite outras informações na assinatura de uma operação. Entretanto, essas são as notações mais comumente utilizadas.

uma pergunta sendo feita. O nome da operação deve ser iniciado com letra minúscula, enquanto os nomes dos complementos devem iniciar com letras maiúsculas, sem dar um espaço em relação à palavra anterior. Acentos não devem ser utilizados. Ex.: calcularValor, emAtraso.

Na assinatura de uma operação, podem ser indicados zero ou mais parâmetros separados por vírgula, cada um deles com a sintaxe abaixo[24], onde *nome_parâmetro* é o nome para referenciar o parâmetro, *tipo* é seu tipo de dados ou classe, e *valor_padrão* é o valor que será assumido se um valor não for informado.

nome_parâmetro: tipo [= valor_padrão]

O *tipo_de_retorno* indica a classe ou o tipo de dado do valor retornado pela operação, o qual pode ser uma classe, um tipo de dado primitivo ou um tipo de dado específico de domínio. Caso uma operação não tenha retorno, nada é especificado.

Conforme discutido anteriormente, há operações, ditas básicas, que simplesmente manipulam atributos e associações, criam ou destroem objetos. Essas operações não são representadas nos diagramas de classes, nem especificadas e documentadas no Dicionário de Projeto, já que podem ser deduzidas do modelo conceitual estrutural. As demais operações devem ser descritas no Dicionário de Projeto, dando uma descrição sucinta de seu propósito.

Referências do Capítulo

BLAHA, M., RUMBAUGH, J., *Modelagem e Projetos Baseados em Objetos com UML 2*, Elsevier, 2006.

BOOCH, G., RUMBAUGH, J., JACOBSON, I., *UML Guia do Usuário*, 2a edição, Elsevier Editora, 2006.

OLIVÉ, A., *Conceptual Modeling of Information Systems*, Springer, 2007.

WAZLAWICK, R.S., *Análise e Projeto de Sistemas de Informação Orientados a Objetos*, Elsevier, 2004.

[24] Idem comentário anterior.

Capítulo 8 – Qualidade e Agilidade em Requisitos

Durante o processo de Engenharia de Requisitos, diversos documentos e modelos podem ser elaborados. Durante o levantamento de requisitos, requisitos de cliente são capturados em um Documento de Requisitos. Na fase de análise, esses requisitos são especificados usando diagramas diversos organizados em dois modelos principais: o modelo estrutural e o modelo comportamental. O modelo conceitual estrutural de um sistema tem por objetivo descrever as informações que esse sistema deve representar e gerenciar. O modelo comportamental, por sua vez, provê uma visão ampla do comportamento do sistema.

No que tange à modelagem do comportamento de um sistema, em um nível superior, os diagramas de casos de uso proveem uma visão externa da funcionalidade do sistema e dos atores nela envolvidos. O comportamento dos casos de uso é elaborado por meio de descrições de casos de uso. Essas descrições podem ser refinadas por meio de diagramas de atividade ou de sequência. Os diagramas de sequência tipicamente mostram visões parciais. Eles, geralmente, são desenvolvidos apenas para fluxos de eventos principais ou são desenvolvidos um diagrama de sequência para o fluxo principal de eventos e diagramas adicionais para os fluxos de exceção e variantes. Diagramas de atividade permitem consolidar todo esse comportamento em um único diagrama, documentando ramificações, bifurcações e uniões do fluxo de controle. Diagramas de sequência são bons para mostrar colaborações entre objetos em um caso de uso. Já os diagramas de atividade são úteis para focalizar as atividades de um processo complexo. Contudo, esses dois tipos de diagramas não são apropriados para observar o comportamento de um objeto isolado ao longo de vários casos de uso. Para tal, são utilizados diagramas de gráfico de estados.

Neste contexto, uma questão importante precisa ser tratada: como fazer a engenharia dos requisitos de um sistema com qualidade, mas ao mesmo tempo de maneira ágil, sem despender tempo elaborando diagramas que acrescentam pouco valor ao projeto? Não se deve perder de vista que o propósito da modelagem é ajudar a entender o problema de modo que se possa produzir um sistema que atenda às necessidades do usuário. Produzir artefatos desnecessários transforma o trabalho de Engenharia de Software em "burocracia de software".

Para tratar essa questão, primeiro deve-se considerar a ótica da qualidade. Sob esse ponto de vista, é fundamental garantir consistência entre os diversos artefatos produzidos. Os diversos documentos e modelos proveem diferentes visões de um mesmo sistema e, portanto, devem estar compatíveis entre si. É importante observar que os modelos produzidos envolvem conceitos comuns, dentre eles classes, objetos e operações, e que é essencial manter as informações rastreáveis. Ambos os modelos, estrutural e comportamental, são necessários para um completo entendimento de um problema, embora a importância de cada um dos diagramas envolvidos varie de projeto para projeto e em função do tipo de aplicação a ser desenvolvida. Esses modelos se unem para dar forma às classes com atributos, associações e operações, que serão a base para o projeto e a implementação. Em especial, as operações envolvem dados (objeto destino, argumentos, variáveis e retorno), controle (sequência) e interações (mensagens e chamadas). Assim, para uma completa especificação das classes, o que envolve a

especificação de suas operações, são necessários os modelos estrutural e comportamental. Além disso, é necessário compará-los para garantir que não há inconsistências entre eles (BLAHA; RUMBAUGH, 2006). Para apoiar a verificação da consistência, técnicas de leitura de modelos de análise podem ser aplicadas. Algumas dessas técnicas são discutidas na Seção 8.1.

Do ponto de vista de agilidade no processo de Engenharia de Requisitos, merece destaque a modelagem ágil. A modelagem ágil provê uma série de princípios e valores que podem ser aplicados para nortear a decisão de quais diagramas produzir, de modo que o esforço despendido na análise de requisitos seja compensador. A modelagem ágil é discutida na Seção 8.2.

Por fim, visando tanto à qualidade quanto a agilidade, podem ser aplicadas técnicas de reutilização de requisitos. A reutilização tende a proporcionar qualidade, uma vez que os requisitos, modelos ou outros artefatos reutilizados já foram avaliados em outros contextos e, por conseguinte, tendem a prover soluções já avaliadas. Do ponto de vista da agilidade, ao se reutilizar algum artefato da Engenharia de Requisitos, espera-se que haja um aumento da produtividade, uma vez que não se está partindo do zero. A reutilização na Engenharia de Requisitos é discutida na Seção 8.3.

8.1 – Técnicas de Leitura de Modelos da Análise de Requisitos

Na verificação da consistência entre os diversos modelos e diagramas produzidos durante um processo de software orientado a objetos, devem ser consideradas técnicas de leitura de modelos orientados a objetos, as quais podem ser de dois tipos: técnicas para leitura vertical e para leitura horizontal. As técnicas para leitura horizontal dizem respeito à consistência entre artefatos elaborados em uma mesma fase, procurando verificar se esses artefatos estão descrevendo consistentemente diferentes aspectos de um mesmo sistema, no nível de abstração relacionado à fase em questão. Técnicas verticais referem-se à consistência entre artefatos elaborados em diferentes fases.

Neste texto, o foco recai sobre técnicas de leitura horizontal, mais especificamente entre os modelos produzidos durante a análise de requisitos. A Figura 8.1 mostra as relações existentes entre os diferentes artefatos elaborados na fase de análise de requisitos, indicando as técnicas de leituras associadas. Essa figura destaca a importância de dois artefatos produzidos na análise de requisitos: as descrições de casos de uso e os diagramas de classes. Como se pode notar pela figura, esses dois artefatos têm relações com quase todos os demais, o que mostra o papel central que eles desempenham no processo de desenvolvimento orientado a objetos. Na sequência, cada uma das técnicas referenciadas na Figura 8.1 é descrita.

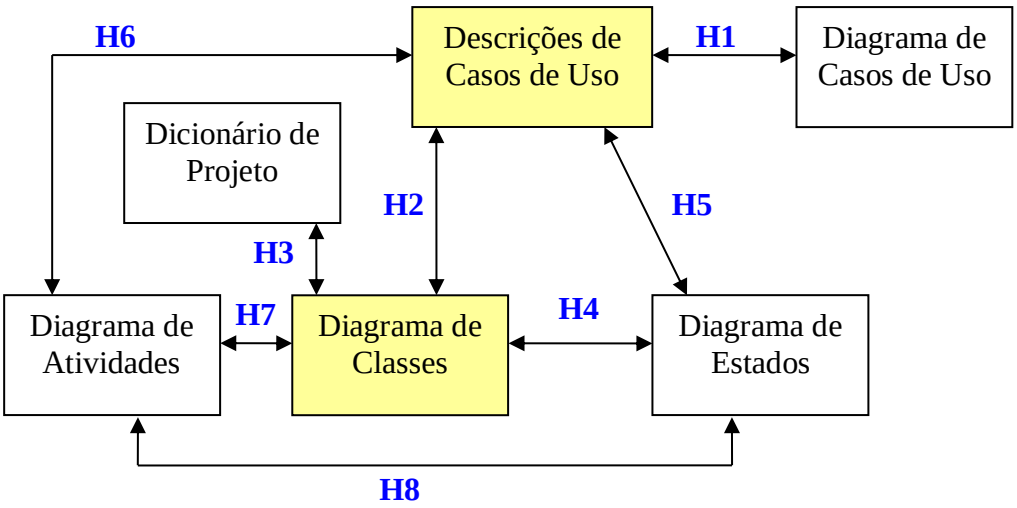


Figura 8.1 – Técnicas de Leitura Horizontal de Modelos Orientados a Objetos – Fase de Análise de Requisitos.

H1 – Descrições de Casos de Uso x Diagrama de Casos de Usos

Os seguintes aspectos devem ser verificados nas relações entre diagramas de casos de uso e descrições de casos de uso:

a) Para cada caso de uso identificado em um diagrama de casos de uso, deve haver uma descrição de caso de uso associada.

b) Os nomes dos casos de uso nos dois artefatos devem ser os mesmos.

c) As descrições dos casos de uso devem fazer menção aos atores envolvidos nos casos de uso e os atores identificados nos dois artefatos devem ser consistentes.

d) Quando um diagrama de casos de uso apontar uma associação entre casos de uso (inclusão ou extensão), a descrição correspondente deve fazer menção explicitamente à realização do caso de uso associado.

H2 – Descrições de Casos de Uso x Diagrama de Classes

Os seguintes aspectos devem ser verificados nas relações entre descrições de casos de uso e diagramas de classes:

a) As classes, associações, atributos e operações modelados no diagrama de classes devem ser necessários e suficientes para realizar cada um dos fluxos de eventos apontados nas descrições de casos de uso.

b) Quando uma descrição de caso de uso fizer menção a dados de uma classe no diagrama de classes, então a descrição do caso de uso deve ser consistente com os atributos modelados na correspondente classe do diagrama de classes (e vice-versa).

c) Para manter os modelos rastreáveis, a descrição de caso de uso deve enumerar as classes envolvidas em sua realização.

H3 – Dicionário de Projeto x Diagrama de Classes

Os seguintes aspectos devem ser verificados nas relações entre dicionários de projeto e diagramas de classes:

a) Para cada classe existente no diagrama de classes deve haver uma entrada no dicionário de projeto, associada a uma descrição sucinta da mesma.

b) Todos os atributos e operações apresentados no diagrama de classes devem estar enumerados e sucintamente descritos na entrada do dicionário de projeto referente à correspondente classe.

c) Restrições de integridade não passíveis de registro pela notação da UML devem ser explicitamente declaradas no modelo de classes. As classes, associações e atributos envolvidos em uma restrição de integridade devem estar consistentes com o diagrama de classes e suas descrições no dicionário de projeto.

H4 – Diagrama de Estados x Diagrama de Classes

Os seguintes aspectos devem ser verificados nas relações entre diagramas de estados e diagramas de classes:

a) Um diagrama de estados deve estar relacionado a uma classe modelada no diagrama de classes.

b) A classe correspondente no diagrama de classes deve conter atributos, associações e/ou operações capazes de indicar todos os estados pelos quais um objeto da mesma pode passar.

H5 – Diagrama de Estados x Descrições de Casos de Uso

Os seguintes aspectos devem ser verificados nas relações entre descrições de casos de uso e diagramas de estados:

a) Os eventos associados a transições entre estados em um diagrama de estados devem corresponder a fluxos de eventos ou a casos de uso em uma descrição de casos de uso.

b) Quando pertinente, a descrição de caso de uso deve fazer uma menção explícita à transição para o novo estado.

H6 – Diagrama de Atividades x Descrições de Casos de Uso

Os seguintes aspectos devem ser verificados nas relações entre descrições de casos de uso e diagramas de atividades:

a) Um diagrama de atividades deve mostrar as atividades no contexto de um fluxo de eventos descrito em uma descrição de casos de uso. Cursos alternativos (de exceção ou variantes) devem ser mostrados no mesmo diagrama de atividades.

b) A sequência de atividades em um diagrama de atividades deve estar consistente com a sequência de passos da descrição do caso de uso correspondente.

H7 – Diagrama de Atividade x Diagrama de Classes

Os seguintes aspectos devem ser verificados nas relações entre diagramas de atividades e diagramas de classes:

- Todos os objetos em um diagrama de atividades devem ter a indicação de a qual classe pertencem. Essas classes devem estar modeladas no diagrama de classes.
- Algumas atividades em um diagrama de atividades, tipicamente aquelas que apontam um processamento a ser feito pelo sistema, podem indicar a necessidade de uma operação na classe de um dos objetos que são entrada para essa atividade. Quando este for o caso, a operação deve ser mostrada na classe correspondente do diagrama de classes.

H8 – Diagrama de Atividade x Diagrama de Estados

Os seguintes aspectos devem ser verificados nas relações entre diagramas de atividades e diagramas de estados:

a) Quando um objeto em um diagrama de atividades tiver o seu estado indicado e houver um diagrama de estados para a classe desse objeto, então os estados devem estar consistentes, inclusive em relação ao evento que altera o estado do objeto.

8.2 – Modelagem Ágil

Ao contrário da modelagem conceitual estrutural em que o modelo de classes é basicamente o modelo a ser construído, na modelagem comportamental há diversos modelos e diagramas que podem ser empregados (modelos de casos de uso, diagramas de estados, diagramas de sequência e diagramas de atividades). Cada um desses diagramas trabalha uma perspectiva diferente e, portanto, diferentes diagramas podem e devem ser elaborados para prover uma visão abrangente do comportamento de um sistema. Entretanto, é fundamental considerar a relação custo-benefício do uso desses modelos. Neste contexto, é importante levar em consideração princípios da Modelagem Ágil.

A Modelagem Ágil (AMBLER, 2004) é uma coleção de valores, princípios e práticas de modelagem de software que pode ser aplicada a projetos de desenvolvimento de software de modo a torná-los mais leves e efetivos. Dentre os princípios de modelagem propostos por Ambler, destacamos os seguintes:

- *O sistema de software é seu objetivo principal; possibilitar o próximo trabalho é seu objetivo secundário*: o principal objetivo de um projeto de desenvolvimento de software é produzir um sistema de software de qualidade e não criar uma documentação. Contudo, durante o desenvolvimento, é necessário preparar o terreno para a próxima atividade, que no caso da modelagem de análise pode ser a fase de projeto ou mesmo de manutenção. Para apoiar atividades subsequentes do processo de software, será necessário ter também uma documentação suficiente e de qualidade.
- *Modele com um propósito*: para que um certo diagrama seja elaborado, deve-se ter uma finalidade em mente. Um diagrama deve ser útil para comunicar informações para clientes e usuários, ajudando a se obter um entendimento comum dos requisitos, ou deve ajudar desenvolvedores a compreender melhor algum aspecto de software. Com base na

meta estabelecida, deve-se escolher o tipo de diagrama mais apropriado para atingir a meta e o nível de detalhe a ser utilizado.

- *Use múltiplos modelos*: há muitos modelos / diagramas e níveis de descrição (notação) que podem ser usados descrever sistemas de software. Contudo, normalmente apenas um pequeno subconjunto deles é essencial para a maioria dos projetos. Assim, para que a elaboração de um certo diagrama se justifique, ele deve adicionar valor ao projeto. Ou seja, apenas aqueles diagramas que ofereçam valor à sua audiência alvo devem ser usados.
- *Diminua a carga de trabalho (Viaje leve)*: cada um dos modelos e diagramas elaborados, se conservado ao longo do desenvolvimento, precisa ser mantido à medida que mudanças nos requisitos acontecem. Isso representa trabalho para a equipe e, portanto, toda vez que se decide conservar um modelo, está-se comprometendo a agilidade em prol da conveniência de se ter aquela informação disponível para a equipe. A diretriz, portanto, é conservar apenas aqueles modelos que fornecerão valor em longo prazo e descartar o restante. Esse princípio tem várias consequências:
    - Deve-se selecionar apenas um pequeno conjunto de modelos e diagramas para serem elaborados, mantendo-se em linha com o princípio anterior;
    - Caso se opte por elaborar um certo diagrama para compreender melhor um certo aspecto do software, ele pode ser elaborado, mas não precisa ser necessariamente incorporado à documentação. Às vezes, um esboço em uma folha de papel cumpre o propósito e o diagrama pode ser descartado em seguida;
    - Praticamente todos os modelos e diagramas elaborados na fase de análise podem ser refinados na fase de projeto, incorporando detalhes relativos à solução específica a ser adotada. Modelos de classes, de casos de uso, diagramas de estados, de sequência e de atividades, por exemplo, podem ter versões de análise e projeto. Contudo, nem sempre vale à pena refinar todos esses diagramas. Deve-se avaliar criteriosamente quais deles refinar na fase de projeto.
- *Adote a simplicidade*: pressuponha que a solução mais simples é a melhor e mantenha os modelos tão simples quanto puder. Não modele seu sistema em excesso hoje, mostrando características adicionais não requeridas.
- *Conheça os modelos e as ferramentas usadas para criá-los*: Entenda o propósito de cada modelo / diagrama, seus pontos fortes e fracos e as principais notações a serem empregadas em cada fase do processo de desenvolvimento. Conheça também as ferramentas usadas para criá-los e suas limitações. Isso dará agilidade ao desenvolvimento de modelos.

Ao considerar esses princípios na modelagem segundo o paradigma orientado a objetos, chega-se ao seguinte conjunto de orientações:

a) Modelos de casos de uso e de classes são essenciais para o desenvolvimento e devem ser elaborados durante a análise de requisitos. Descrições de casos de uso completas só devem ser elaboradas para casos de uso mais complexos, tipicamente aqueles relacionados aos principais processos de negócio sendo apoiados pelo sistema em desenvolvimento.

b) Diagramas de estados só devem ser elaborados para classes com comportamento dependente do estado. Recomenda-se elaborar diagramas de estados apenas para as

classes que possuírem três ou mais estados relevantes. Se julgado útil para a compreensão, pode ser elaborado um diagrama de estados para uma classe com apenas dois estados relevantes. Entretanto, deve-se avaliar se o mesmo não pode ser descartado após cumprir seu propósito, não sendo incorporado à documentação do projeto.

c) Diagramas de atividades só devem ser elaborados quando forem úteis para refinar o entendimento provido pela descrição de um caso de uso complexo, normalmente com várias atividades paralelas, iterativas, condicionais ou alternativas. Diagramas de atividades também são uma boa opção para a modelagem de processos de negócio (ver Seção 3.3).

d) Em relação ao refinamento de modelos e diagramas na fase de projeto, de maneira geral, modelos de classe devem ser refinados, uma vez que eles são a base para a implementação. O mesmo não ocorre com os modelos comportamentais (modelos de casos de uso, diagramas de estados, sequência e atividades), os quais, na maioria das vezes, basta manter a versão de análise.

8.3 – Reutilização na Engenharia de Requisitos

A reutilização no desenvolvimento de software tem como objetivos melhorar o cumprimento de prazos, diminuir custos e obter produtos de maior qualidade (GIMENES; HUZITA, 2005). Artefatos de software são reutilizados com o intuito de diminuir o tempo de desenvolvimento, investindo-se esforço na sua adaptação ao invés de se investir esforço na sua construção a partir do zero. Ao longo do processo de software, diversos tipos de artefatos podem ser reutilizados, dentre eles modelos, especificações, planos, código-fonte etc. Nesta seção, o foco é na reutilização como apoio à Engenharia de Requisitos.

Analisando o processo de Engenharia de Requisitos, é possível notar que a reutilização pode ser útil, sobretudo no reúso de requisitos de sistemas similares e de modelos conceituais. Contudo, na prática, na grande maioria das vezes, requisitos e modelos são construídos a partir do zero. Ou seja, requisitos de cliente são levantados junto aos interessados e posteriormente refinados em requisitos de sistema, quando modelos são construídos levando-se em conta os requisitos inicialmente capturados. Entretanto, essa abordagem tem se mostrado insuficiente, pois desconsidera a reutilização do conhecimento já existente na organização acerca do domínio do problema (FALBO et al., 2007).

Ao especificar requisitos para um sistema é interessante ter em mente que muito provavelmente alguém já desenvolveu algum sistema muito parecido ou até mesmo idêntico ao que está sendo desenvolvido. Assim, caso se tenha acesso aos requisitos de um sistema similar ao que será desenvolvido, certamente o esforço para se obter requisitos para o mesmo será bem menor (ROBERTSON; ROBERTSON, 2006).

O reúso na Engenharia de Requisitos tem por objetivo a utilização de requisitos (e outros artefatos relativos às fases da Engenharia de Requisitos) de projetos anteriores com o intuito de auxiliar a obtenção dos requisitos para um novo sistema a ser desenvolvido. Requisitos podem ser reutilizados de diversas maneiras, dentre elas pelo reúso de especificações de projetos similares ao projeto atual, informalmente através da experiência das pessoas ou pela reutilização de artefatos desenvolvidos previamente com vistas ao reúso. Por exemplo, ao se analisar um projeto similar, desenvolvido para um mesmo domínio de aplicação, pode-se concluir que diversos requisitos funcionais, não funcionais e até regras de negócio podem se

aplicar ao novo projeto em desenvolvimento. Neste caso, os requisitos podem ser copiados, e adaptados, quando necessário, de um projeto para o outro. Outra abordagem bastante utilizada pelas organizações consiste em definir equipes que atuam sistematicamente em um certo domínio de aplicações, de modo que seus membros possam reutilizar o conhecimento que têm sobre esse domínio no desenvolvimento de novos projetos.

Ainda que essas abordagens mais informais e oportunistas tragam alguns benefícios, elas não exploram adequadamente as potencialidades de uma reutilização sistemática. Em uma abordagem sistemática de desenvolvimento *para* e *com* reutilização, primeiramente artefatos são concebidos explicitamente para serem reutilizados. Depois, os projetos de desenvolvimento reutilizam esses artefatos, fazendo as adaptações necessárias. Uma vez que esses itens tenham sido desenvolvidos para reúso, o esforço de adaptação e reutilização tende a ser menor. Assim, para se obter os reais benefícios da reutilização, é importante que os artefatos já sejam desenvolvidos pensando no reúso posterior. Neste contexto, merecem destaque três abordagens, discutidas na sequência: Engenharia de Domínio, Ontologias e Padrões de Análise.

8.3.1 – Engenharia de Domínio

A Engenharia de Domínio representa um enfoque sistemático para a produção de componentes reutilizáveis que engloba atividades de análise, projeto e implementação de domínio, as quais objetivam, respectivamente, representar requisitos comuns de uma família de aplicações por meio de modelos de domínio, disponibilizar modelos arquiteturais para aplicações a partir de um único modelo de domínio e disponibilizar implementações de componentes que representam funcionalidades básicas de aplicações relacionadas a um domínio (GIMENES; HUZITA, 2005).

A Análise de Domínio é a atividade diretamente ligada à reutilização na Engenharia de Requisitos. A Análise de Domínio visa capturar os elementos relevantes de um domínio de aplicações e disponibilizá-los para serem utilizados no desenvolvimento de diferentes sistemas de apoio a negócios neste domínio. Assim, a Análise de Domínio busca explicitar e modelar aspectos de domínio, produzindo artefatos (tipicamente modelos) que contêm informações sobre o domínio e que podem ser reutilizados no desenvolvimento de sistemas.

Pode-se fazer um paralelo entre a Análise de Domínio e a Análise de Requisitos: ambas enfocam a modelagem conceitual de um domínio de aplicações. Entretanto, enquanto a análise de requisitos convencional enfoca a modelagem dos aspectos do domínio que são relevantes para um sistema específico, a análise de domínio é mais abrangente e visa capturar elementos de informação do domínio potencialmente relevantes para o desenvolvimento de diversos sistemas neste domínio, estando, portanto, em nível mais alto de abstração. Em outras palavras, ao invés de explorar requisitos de uma aplicação específica, na análise de domínio, os requisitos explorados dizem respeito a uma família de aplicações de uma determinada área (ARANGO; PRIETO-DÍAZ, 1994). Neste sentido, pode-se considerar que a Análise de Domínio direciona a Engenharia de Requisitos, pois seus modelos de domínio, mais abstratos, fornecem uma base para o trato com requisitos no contexto de um projeto específico.

Para se fazer uma análise de domínio, pode-se trabalhar de maneira análoga à análise de requisitos convencional, consultando-se especialistas do domínio e modelando o conhecimento capturado, de modo que esse conhecimento possa ser reutilizado nos vários projetos que façam parte desse domínio (ROBERTSON; ROBERTSON, 2006). Há, ainda, diversos métodos de análise de domínio, os quais procuram explorar as especificidades dessa atividade, dentre eles

FODA, ODM, EDLC, FORM, RSEB e Catalysis (GIMENES; HUZITA, 2005). Contudo, qualquer que seja a abordagem de análise de domínio empregada, deve-se ter em mente que um modelo de domínio deve conter os elementos comuns às várias aplicações do domínio e não detalhes específicos de uma ou de todas as aplicações. Assim, para ser realmente reutilizável, um modelo de domínio deve ser mais geral e abstrato, contendo o conhecimento de senso comum a respeito do domínio, sem incluir detalhes que podem ser relevantes apenas para uma ou para poucas aplicações.

Deve-se apontar, ainda, que a análise de domínio apenas será uma abordagem interessante, caso haja uma real necessidade de reúso no domínio em questão. A análise de domínio é uma atividade complexa que consome tempo e recursos, podendo ser anterior ou paralela a um processo de desenvolvimento de software. Assim, se não houver uma real possibilidade de reúso nesse domínio, a análise de domínio torna-se inviável.

A Figura 8.2 ilustra o processo de Engenharia de Domínio e como ele pode apoiar o processo de desenvolvimento de um sistema de software.

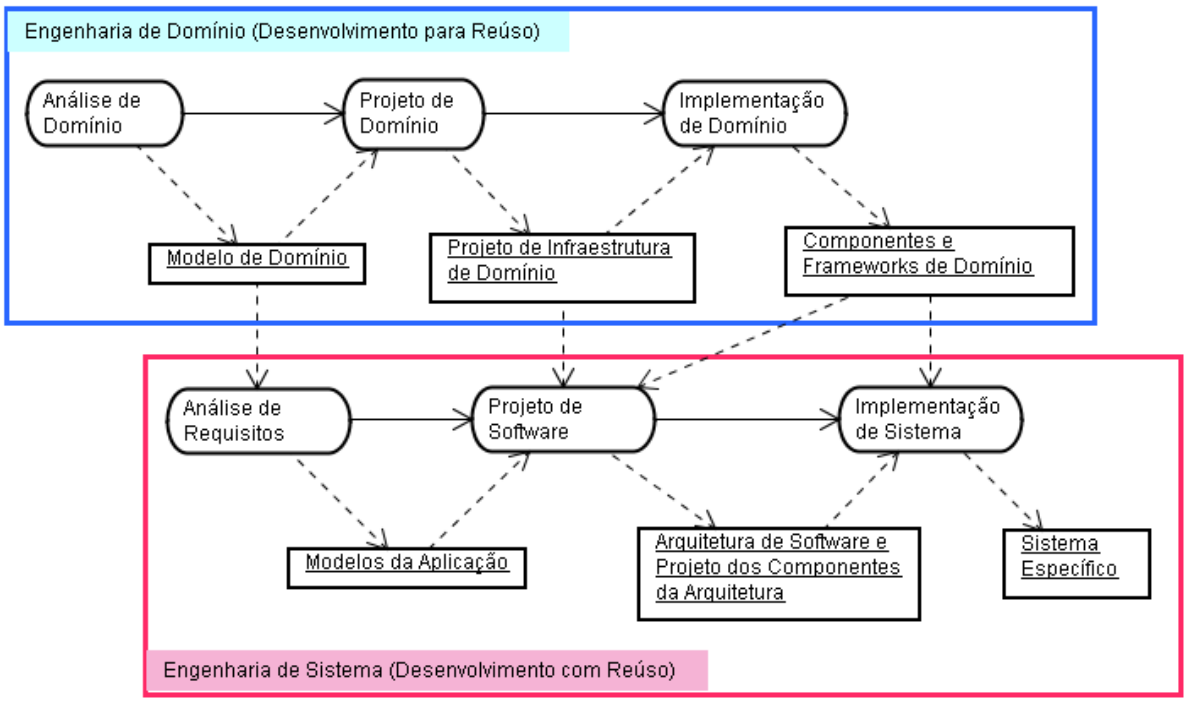


Figura 8.2 – Abordagens *para* e *com* Reúso na Engenharia de Domínio.

Nuseibeh e Easterbrook (2000), em seu mapa para o futuro da Engenharia de Requisitos, colocaram o reúso de modelos como um dos maiores desafios da Engenharia de Requisitos para os anos 2000. Eles acreditavam que, para muitos domínios de aplicação, modelos de referência para especificação de requisitos seriam desenvolvidos, de modo que o esforço de desenvolver modelos de requisitos a partir do zero fosse reduzido. De alguma forma, isso, de fato, vem ocorrendo, ainda que não propriamente na forma de modelos de referência, mas por meio da construção de diversas ontologias de domínio e pelo crescente número de padrões de análise disponíveis (FALBO et al., 2007), assuntos discutidos a seguir.

8.3.2 – Ontologias de Domínio

Uma grande dificuldade atual no desenvolvimento de sistemas para apoiar processos de negócio em um mesmo domínio, sobretudo quando os mesmos precisam trabalhar de maneira integrada, é a existência de diversas visões parciais sobre esse domínio. Cada visão parcial carrega consigo um vocabulário e determinados valores próprios, dificultando a integração entre os diversos profissionais pela ausência de padronização. Esse problema pode ser minimizado se a conceituação envolvida no domínio for, pelo menos parcialmente, explicitada, o que pode ser feito por meio de uma ontologia de domínio.

Uma ontologia de domínio é um artefato de engenharia que busca explicitar a conceituação de um domínio particular. Uma conceituação, por sua vez, corresponde ao conjunto de conceitos usados para interligar abstrações de entidades de um dado domínio. Assim, uma ontologia de domínio tem como objetivo explicitar e formalmente definir os conceitos, relações, propriedades e restrições em um domínio particular (GUIZZARDI, 2005).

Ontologias de domínio têm diversos usos, dentre eles (JASPER; USCHOLD, 1999): (i) apoio à comunicação entre pessoas envolvidas no domínio; (ii) integração de dados e interoperabilidade de sistemas desenvolvidos para o domínio e (iii) especificação reutilizável para a construção de sistemas no domínio.

Ontologias de domínio podem ser usadas como modelos de domínio em uma abordagem de Engenharia de Domínio. Assim, como discutido na seção anterior, pode-se pensar que o processo de Engenharia de Domínio, neste caso, envolveria atividades de Modelagem da Ontologia, Projeto da Ontologia e Implementação da Ontologia. Do ponto de vista da Engenharia de Requisitos, está-se mais interessado em ontologias como modelos conceituais. Assim, o foco está nas ditas ontologias de referência, as quais representam um modelo de consenso dentro de uma comunidade. Uma ontologia de referência é uma especificação independente de solução, com o objetivo de fazer uma descrição clara e precisa de entidades do domínio, para propósitos de comunicação, aprendizado e resolução de problemas (ZAMBORLINI; GONÇALVEZ; GUIZZARDI, 2008).

Vale a pena destacar similaridades e diferenças entre modelos de domínio em uma abordagem tradicional de Engenharia de Domínio e em uma abordagem baseada em ontologias. Ambos têm como propósito capturar a conceituação de um certo domínio, estando em um nível de abstração mais elevado do que um modelo conceitual de um sistema. Entretanto, ontologias são desenvolvidas com vários propósitos em mente (e não somente de servir como uma especificação base para o desenvolvimento de sistemas), o que faz com que tenha de ser especificada com mais rigor, normalmente exigindo o consenso em uma comunidade, tornando-a mais geral do que modelos de domínio.

Uma vez que uma ontologia é um artefato complexo de engenharia, ela deve ser construída usando métodos apropriados. Há diversos métodos existentes para o desenvolvimento de ontologias, dentre eles SABiO (FALBO, 2004), Neon (SUÁREZ-FIGUEROA et al., 2007) e Methontology (GOMEZ-PEREZ; CORCHO; FERNANDEZ-LOPEZ, 2007).

8.3.3 – Padrões de Análise

Padrões de software formalizam soluções para problemas recorrentes no desenvolvimento de software, com base na experiência de especialistas, permitindo que essas soluções possam ser reutilizadas em várias outras situações, por outros especialistas. Na engenharia de software, padrões são usados para descrever soluções de sucesso para problemas de software comuns. Esses padrões refletem a estrutura conceitual dessas soluções e podem ser aplicados várias vezes na análise, projeto e produção de aplicações, em um contexto particular (DEVEDZIC, 1999). O uso pelos engenheiros de software de soluções já conhecidas e testadas ajuda a aumentar o nível de abstração, a produtividade e a qualidade dos projetos nas várias fases do desenvolvimento de sistemas (COTA, 2004).

No contexto de desenvolvimento de software, padrões procuram representar a experiência de muitos esforços de reengenharia de desenvolvedores que tentaram adquirir maior reusabilidade e flexibilidade em seus produtos de software. Representam também, a formalização de uma solução para determinada situação que desenvolvedores sempre se depararam na sua experiência em desenvolver sistemas. Diante de uma situação recorrente, observa-se que pode ser aplicada uma solução que já foi adotada em vários contextos e, com isso, estabelece-se um padrão.

Os desenvolvedores não inventam padrões de software, descobrem-nos da experiência em construir sistemas na prática (DEVEDZIC, 1999). Em outras palavras, os padrões são descobertos e não inventados. Por exemplo, modelos se tornam padrões somente quando se descobre que eles podem ter uma utilidade comum. Ou seja, padrões são coisas que os desenvolvedores conhecem e reconhecem que podem ser úteis em diferentes contextos. Ou seja, a partir da análise de diversos eventos de negócio do mesmo tipo, um padrão de análise pode ser derivado por meio da abstração de elementos comuns (ROBERTSON; ROBERTSON, 2006). Com um padrão, é apresentada uma aproximação genérica para resolver um problema. Assim, padrões têm que ser trabalhados e adaptados para casos específicos.

Padrões existem em várias fases do desenvolvimento de software. A comunidade de padrões de software primeiro descobriu, descreveu e classificou um certo número de padrões de projeto (GAMMA et al.,1995). Mais recentemente foram identificados outros padrões relacionados a outras fases e aspectos do desenvolvimento de software, como os padrões de análise (FOWLER, 1997) e padrões para arquiteturas de software (FOWLER, 2003). Para a Engenharia de Requisitos, os padrões de interesse são os padrões de análise.

Padrões de análise são definidos a partir de modelos conceituais de aplicações. Ao se criar um modelo conceitual, os analistas percebem que muitos aspectos de projetos anteriores se repetem. Se ideias usadas em projetos anteriores são úteis atualmente, então, eles melhoram essas ideias e as adaptam para atender a uma nova demanda. Para reutilizar um padrão de análise em outra aplicação, é preciso reinterpretar cada classe no padrão como uma classe correspondente no novo sistema. A estratégia é a abstração do modelo inicial, de onde novos modelos podem ser desenvolvidos.

Fowler (1997) se refere a padrões de análise como grupos de conceitos que representam uma construção comum na modelagem de negócios. Eles podem ser relevantes para somente um domínio ou podem ser expandidos para vários domínios. Os padrões de análise lidam com problemas gerais de modelagem, sendo apresentados através de um problema particular em um domínio, onde é mais fácil de entendê-los. Conhecendo-os, são identificadas inúmeras situações onde aplicá-los. No citado livro, são descritos padrões para vários domínios de negócios, de

acordo com a experiência pessoal do autor em aplicar modelagem de objetos a sistemas de informação de grandes corporações.

Padrões de maneira geral são registrados em catálogos. Um catálogo de padrões é uma coleção de padrões com uma estrutura e uma organização. Os padrões são subdivididos em categorias e há relações entre eles. A classificação em um catálogo facilita o aprendizado e direciona os esforços para encontrar novos padrões, torna-os úteis para os leitores, que podem lembrar dos padrões, identificá-los em diferentes situações e reutilizá-los no desenvolvimento de algum sistema. Para tal, a cada padrão de software é dado um nome, que serve como um guia para facilitar a discussão do padrão e a informação que ele representa.

Um dos catálogos de padrões de análise mais conhecidos é o descrito por Fowler em seu livro *Analysis Patterns: Reusable Objects Models* (Padrões de Análise: Modelos de Objetos Reutilizáveis) (FOWLER, 1997), onde há mais de 40 padrões de análise, contendo modelos já aplicados em projetos nos quais o autor trabalhou. Os padrões neste catálogo são organizados em dez categorias principais, a saber: Responsabilidades, Observações e Medições, Observações para Finanças Coorporativas, Referência a Objetos, Inventário e Contabilidade, Uso de Modelos de Contabilidade, Planejamento, Negociação, Contratos Derivados e Pacotes de Negociação.

Os padrões de análise relativos a responsabilidades, por exemplo, buscam capturar situações em que uma pessoa ou organização é responsável por outra. É uma noção abstrata que pode empregada em várias situações, incluindo estrutura organizacional, contratos e relações de emprego. Dentre os padrões dessa categoria podem ser citados: Parte (*Party*), Hierarquia Organizacional *(Organization Hierarquies*) e Estrutura Organizacional (*Organization Structure*).

Os padrões de análise de negociação, por sua vez, tratam o comércio de mercadorias sob uma perspectiva de vendedor / comprador. Esses padrões tratam compra e venda de mercadorias e o valor dessas mercadorias com respeito às mudanças nas condições de mercado. Nesta categoria há, dentre outros, os seguintes padrões: Contrato (*Contract*) e Carteira de Negócios (*Portfolio*).

As figuras 8.3, 8.4 e 8.5 mostram alguns exemplos de modelos estruturais propostos em padrões de análise, adaptados de (FOWLER, 1997). Na Figura 8.3, é apresentado o padrão Pessoa, cujo propósito é tratar situações em que pessoas físicas e jurídicas têm responsabilidades similares. Neste caso, a solução adotada é criar um tipo Pessoa, como um supertipo de Pessoa Física e Pessoa Jurídica, reunindo propriedades comuns às duas.

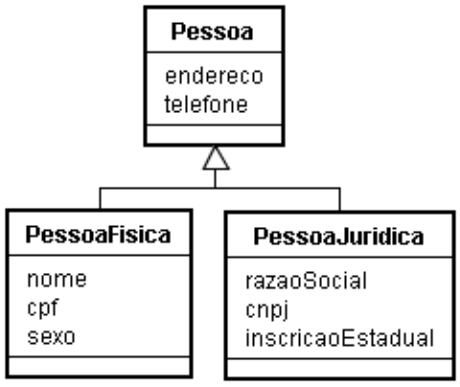


Figura 8.3 – Padrão de Análise Pessoa.

As figuras 8.4 e 8.5 mostram diferentes versões de um mesmo padrão: o padrão Plano. O problema envolvido neste caso é o planejamento de ações. Na versão da Figura 8.4, considera-se que uma ação é definida exclusivamente para um plano. Na versão da Figura 8.5, considera-se que uma ação proposta pode fazer parte de diversos planos.

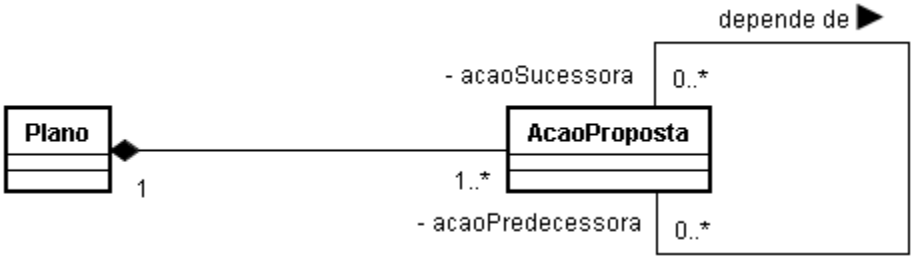


Figura 8.4 – Padrão de Análise Plano – Versão 1.

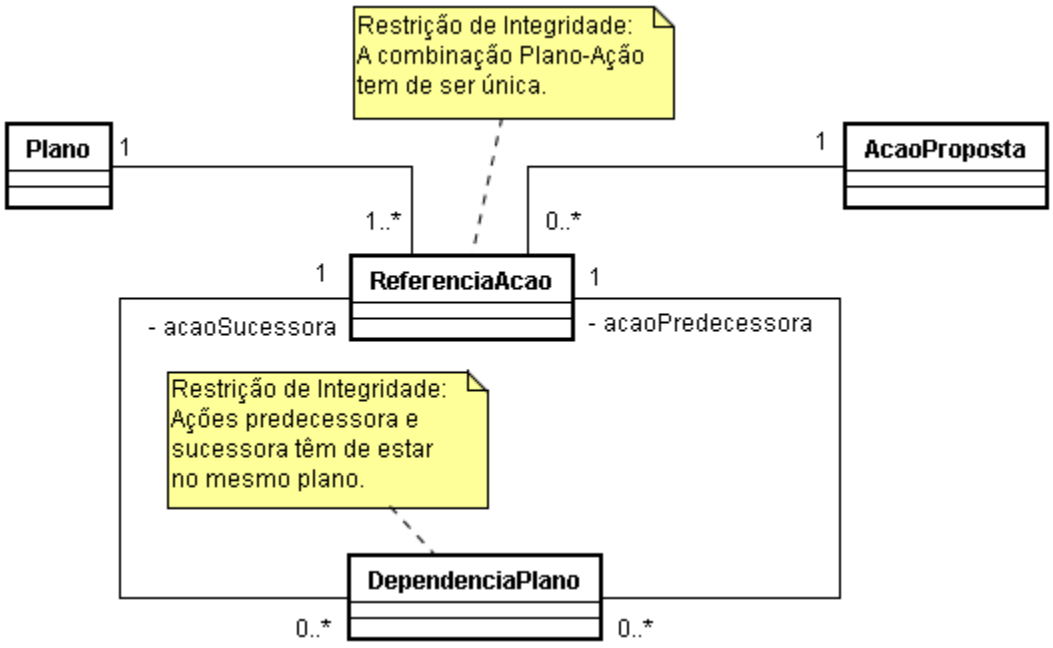


Figura 8.5 – Padrão de Análise Plano – Versão 2.

Utilizar ontologias e padrões de análise para a modelagem conceitual de sistemas abre espaço para uma visão mais abrangente do domínio tratado, levando a um melhor entendimento do mesmo. Isso pode diminuir o tempo gasto na especificação de requisitos, pois os analistas já têm uma fonte preliminar para aprender sobre o domínio, que estabelece uma terminologia comum aos especialistas do negócio (COTA, 2004).

Padrões de Análise, assim como ontologias, descrevem aspectos no nível de conhecimento. Assim, uma vez que eles proveem conhecimento sobre soluções bem sucedidas a problemas recorrentes no desenvolvimento de software, eles favorecem a reutilização. No entanto, ontologias capturam a estrutura conceitual intrínseca de um domínio, enquanto padrões de análise focalizam a estrutura de uma aplicação (DEVEDZIC, 1999).

Em (FALBO et al., 2007) é proposto um processo de Engenharia de Requisitos baseado em reutilização de ontologias e padrões de análise, voltado para o desenvolvimento de sistemas

de informação. O processo proposto visa contribuir para a institucionalização de práticas de reúso na Engenharia de Requisitos. O processo começa com um levantamento preliminar de requisitos, cujo objetivo é definir o escopo do projeto e enumerar os principais requisitos funcionais e não-funcionais. Paralelamente, um modelo conceitual é elaborado, tomando por base ontologias e padrões de análise relacionados ao domínio do problema. Os requisitos são então detalhados e os modelos detalhados. Como produto, um documento de especificação de requisitos é produzido.

Referências do Capítulo

AMBLER, S.W., *Modelagem Ágil: Práticas Eficazes para a Programação eXtrema e o Processo Unificado*, Porto Alegre: Bookman, 2004.

ARANGO, G., PRIETO-DÍAZ, R., *Domain Analysis Concepts and Research Directions*, Workshop on Software Architecture, USC Center for Software Engineering, Los Angeles, EUA, 1994.

BLAHA, M., RUMBAUGH, J., *Modelagem e Projetos Baseados em Objetos com UML 2*, Elsevier, 2006.

COTA, R.I.; *Um Estudo sobre o Uso de Ontologias e Padrões de Análise na Modelagem de Sistemas de Gestão Empresarial*. Dissertação de Mestrado, Mestrado em Informática, UFES, 2004.

DEVEDZIC, V.; "Ontologies: Borrowing from Software Patterns", Intelligence, vol. 10, issue 3 (Fall 1999), 1999, p. 14-24.

FALBO, R. A.; "Experiences in Using a Method for Building Domain Ontologies" Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering, International Workshop on Ontology In Action, Banff, Canada, 2004.

FALBO, R.A., MARTINS, A.F., SEGRINI, B.M., BAIÔCO, G., DAL MORO, R., NARDI, J.C.; "Um Processo de Engenharia de Requisitos Baseado em Reutilização e Padrões de Análise", VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'07), Lima, Peru, 2007.

FOWLER, M.; *Analysis Patterns: Reusable Object Models*. Addison-Wesley Professional Computing Series, 1997.

FOWLER, M., *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J.M., *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

GIMENES, I.M.S., HUZITA, E.H.M.; *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*. Editora Ciência Moderna, 2005.

GÓMEZ-PÉREZ, A., CORCHO, O., FERNANDEZ-LOPEZ, M.; *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, 2nd edition, Springer, 2007.

GUIZZARDI, G., *Ontological Foundations for Structural Conceptual Models*, Telematics Instituut Fundamental Research Series, The Netherlands, 2005.

JASPER, R., USCHOLD, M.; "A Framework for Understanding and Classifying Ontology Applications", Proceedings of the IJCAI99 Workshop on Ontologies and Problem-Solving Methods, Stockholm, Sweden, 1999.

NUSEIBEH, B., EASTERBROOK, S., "Requirements Engineering: A Roadmap", In: Procedings of the Future of Software Engineering, ICSE'2000, Ireland, 2000, p. 37-46.

ROBERTSON, S., ROBERTSON, J. *Mastering the Requirements Process***.** 2nd Edition. Addison Wesley, 2006.

SUÁREZ-FIGUEROA, M.A., et al., *NeOn Development Process and Ontology Life Cycle*, 2007. Disponível em http://www.neon-project.org/web-content/index.php?option=com_weblinks&view=category&id=17&Itemid=73.

ZAMBORLINI, V., GONÇALVEZ, b., GUIZZARDI, G.; "Codification and Application of a Well-Founded Heart-ECG Ontology", Proceedings of the 3rd Workshop on Ontologies and Metamodels in Software and Data Engineering, Campinas, Brazil, 2008.