Linguagem de Modelagem Unificada (UML)

1. Introdução

Sob o ponto de vista de quem desenvolve software, seu principal papel deve ser a construção de um produto de software capaz de satisfazer às necessidades de seus usuários e respectivos negócios, a partir de uma verificação detalhada dos problemas que devem ser resolvidos, aliada aos desejos do usuário sobre a questão. Nem sempre todas as solicitações oriundas do contexto para informatização parecerão lógicas sob a ótica de um técnico, mas serão motivadas por questões políticas no cerne de uma cultura empresarial. Considerar todos os requisitos para desenvolvimento de um software deve ser o principal objetivo para um desenvolvedor e modelar um sistema é uma atividade de primeira grandeza, sem a qual, o objetivo principal não será devidamente alcançado.

1.1 Modelagem Visual

Para conseguir desenvolver um software capaz de satisfazer às necessidades de seus usuários, com qualidade duradoura, por intermédio de uma arquitetura sólida que aceite modificações, de forma rápida, eficiente, com o mínimo de desperdício e retrabalho, é necessário o emprego de modelagem. Modelagem é uma parte central de todas as atividades que levam à implantação de um bom software. Construir o modelo de um sistema não é uma atividade simples ou fácil, são várias abordagens a serem consideradas: a organização da empresa, os processos existentes ou requeridos, as informações existentes (ou requeridas) e os recursos envolvidos.

Pode-se fazer uma representação dos recursos organizacionais dispondo-os em hierarquia, conforme a figura a seguir, de maneira que o primeiro nível hierárquico poderia ser visualizado em um organograma que define responsabilidades a serem exercidas por pessoas. As pessoas no desempenho de algum papel dentro da organização são, em geral, as responsáveis pela execução de processos. Assim, do ponto de vista da confrontação ou cruzamento entre um organograma e uma modelagem funcional da organização, encontra-se um forte impacto de natureza política, que promove, segundo sua conveniência, várias modificações na condução dos processos existentes, muitas vezes sem uma preocupação com os aspectos técnicos envolvidos, o que torna a modelagem de processos uma tarefa difícil.

Não é possível desenvolver uma modelagem de processos sem interferência de pessoas e da cultura estabelecida em uma organização; portanto, quem modela deve desenvolver habilidades que sobrepujam às técnicas.

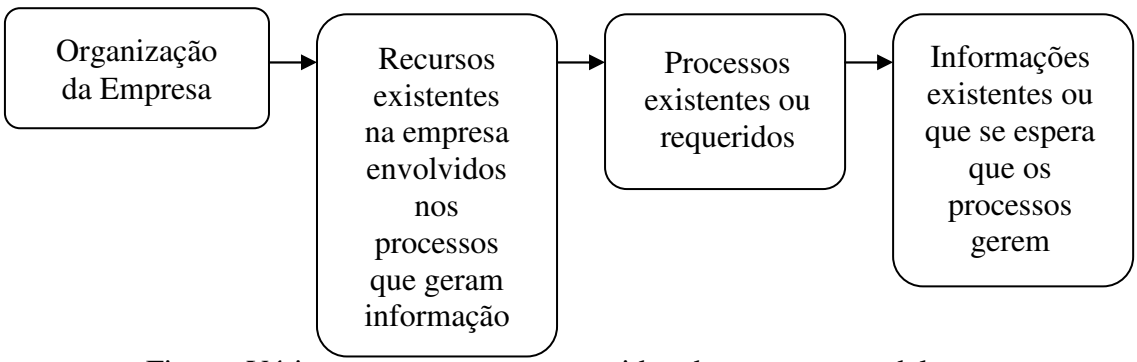


Figura: Vários aspectos a serem considerados em uma modelagem

Modelos são construídos para que o sistema que será desenvolvido seja melhor compreendido. Com a modelagem, pode-se alcançar quatro objetivos:

- Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que seja.
- Os modelos permitem especificara estrutura ou o comportamento de um sistema.
- Os modelos proporcionam um guia para a construção do sistema.
- Os modelos documentam as decisões tomadas

Existem limites para a capacidade humana de compreender complexidades, mais especificamente, reter todos os detalhes que envolvem uma realidade complexa, os relacionamentos existem, as possíveis situações que possam ocorrer dependendo da combinação de cada aspecto envolvido. Com a ajuda da modelagem, delimitamos o problema que estamos estudando, restringindo nosso foco a um único aspecto por vez. Quanto mais complexo for o sistema, maior será a probabilidade de ocorrência de erros ou de construção de itens errados, caso não haja nenhuma modelagem, além disso, pode-se 'esquecer' de detalhes que surpreendentemente irão comprometer o produto quando estiver sendo utilizado.

Os diferentes aspectos do sistema que está sendo modelado são chamados de *visões*. Um sistema que se planeja construir poderá vir a ter um número ilimitado de visões; quanto maior a complexidade do sistema maior tende a ser a quantidade de visões que se avaliará, cada uma mostrará aspectos particulares do sistema propiciando ângulos e níveis de abstração diferentes; desse modo, um molde completo do sistema poderá ser construído. As visões também podem servir de ligação entre a linguagem de modelagem e o método/processo de desenvolvimento escolhido. Qualquer sistema deve ser considerado a partir de três macros aspectos básicos:

- Funcionais (sua estrutura estática e suas interações dinâmicas).
- Não Funcionais (requisitos de tempo, confiabilidade, desenvolvimento etc.).
- Organizacionais (organização do trabalho, mapeamento dos módulos de código, distribuição física do hardware etc.).

De acordo com o método de desenvolvimento a ser utilizado, cada visão é descrita por um ou mais conjuntos de diagramas que contemplam os elementos daquela porção da realidade. Todos os sistemas bem desenvolvidos, que se mostram como recursos úteis a seus usuários, apresentam uma tendência natural para se transformarem em algo mais complexo ao longo do tempo, dados que os requisitos são mutáveis no tempo; assim, o panorama que originou o software em algum momento, vai se transformando de maneira que, se o software não sofrer adequações para atender aos novos requisitos, acaba se tornando obsoleto. Portanto, ainda que considere não ser necessário fazer modelagem hoje, à medida que o sistema construído evoluir, tornando-se algo mais complexo, você certamente se arrependerá dessa decisão.

1.2 SÍNTESE HISTÓRICA DA UML

A linguagem de modelagem unificada (UML) *não* é um método de desenvolvimento de sistemas, é uma linguagem gráfica que poder ser aplicada para descrever e documentar um projeto de software. Ela simplifica o complexo processo de análise, projeto e construção de software criando visões do sistema que está sendo construído.

Um método pressupõe um modelo de linguagem para as especificações técnicas e um processo. O modelo de linguagem é a notação que o método usa para descrever o

projeto. O processo é constituído de passos que devem ser seguidos na construção de um projeto. O modelo de linguagem é uma parte muito importante do método. Corresponde ao ponto principal da comunicação. Se uma pessoa que conversar com outra sobre o projeto, é por meio do modelo de linguagem que elas se entendem, na medida em que um projeto avança, por meio do modelo de linguagem documenta-se tudo o que foi definido nas fases. Se em algum momento deseja-se recuperar detalhes sobre um aspecto que foi analisado anteriormente, essa atividade é facilitada com o modelo de uma linguagem, ainda que a atividade seja desenvolvida por uma pessoa diferente daquela que fez a referida especificação.

A UML é uma linguagem padrão para visualizar, especificar, construir e documentar artefatos de um sistema baseado em software. Os autores da UML preocupam-se em incorporar recursos que permitissem a abordagem de diversos tipos de sistemas, dos mais simples até os sistemas concorrentes e distribuídos. Os esforços são concentrados em um metamodelo comum, que unifica as semânticas, e em uma notação comum que fornece uma interpretação humana dessas semânticas. A UML reuniu vários recursos existentes em diversos métodos orientados a objetos.

São várias as menções acerca da grande quantidade de métodos orientados a objetos que foram originados no início dos anos 90. A 'novidade' de aplicar-se o modelo orientado a objetos no processo de desenvolvimento do software, bem como algumas características que demonstravam ser um meio eficaz para a produção do software, levaram a um grande entusiasmo quanto à aplicação do recurso, muito embora tenha se estabelecido o inconveniente de que, dado ao grande numero de metodologias existentes, não se tinha a noção de qual deles seria o ideal. A indústria não tinha como lançar produtos que dessem resguardo a este ou aquele método, pois não tinha a perspectiva sobre qual deles haveria uma convergência de mercado.

Em 1997, por iniciativa da OMG (*Object Management Group*), foi aberta a proposta para apresentação de trabalhos de padronização de um modelo para desenvolvimento de sistemas que atendesse ao modelo orientado a objetos. A UML foi a proposta vencedora, apresentada pela empresa *Rational Software Corporation* e se tornou um padrão a ser seguido pelo mercado, com relação às especificações orientadas a objetos. A OMG é uma organização sem fins lucrativos que cuida das padronizações vinculadas ao modelo orientado a objetos, possui mais de 800 filiados, incluindo empresas de renome no mercado internacional, implicando, portanto, que o mercado como um todo utilizará softwares que irão considerar a UML como referência.

Essa proposta de padronização foi um esforço liderado por *Grady Booch*, *James Rumbaugh* e *Ivar Jacobson* que resultou na versão 1.0 da UML publicada em 13 de janeiro de 1997 e adotada como padrão pela OMG no mesmo ano. Aglutinou o que havia de melhor em vários métodos existentes, tendo recebido a colaboração de vários metodologistas.

1.3 CONCEITOS DA UML

A UML tem como objetivo prover as necessidades de desenvolvedores de software com uma linguagem de modelagem visual completa, buscando atingir os seguintes aspectos:

- Disponibilização de mecanismos de especificações que possam expressar os níveis conceituais.

- Independência de processos de desenvolvimento e linguagens de programação.
- Incentivo do crescimento das aplicações desenvolvidas no conceito da orientação a objetos.
- Permissão de suporte a conceitos de desenvolvimento de alto nível, tais como *frameworks*, padrão e componentes.

O processo de desenvolvimento do software não está previsto na UML, o que a torna, portanto, uma linguagem de modelagem e não um método; no entanto, pode-se eleger em termos genéricos cinco etapas para o desenvolvimento de software em que a UML pode ser aplicada: análise de requisitos, análise sistêmica, projeto, implementação e testes/implantação.

1.3.1 ANÁLISE DE REQUISITOS

O levantamento de requisitos deve ser a primeira etapa a ser desenvolvida, uma vez que reunirá os subsídios necessários para as etapas seguintes. Na análise de requisitos se verificam quais são os problemas e desejos do usuário com relação ao software que será desenvolvido. À medida que o levantamento de requisitos é realizado, pode-se fazer uma modelagem das atividades encontradas, empregando-se para isso o diagrama *use-case*. Esse diagrama permite a representação da relação do software com o ambiente externo a ele, demonstrando tudo aquilo que terá alguma responsabilidade frente ao software (pessoas, departamento e outros sistemas). As pessoas, departamentos e outros sistemas são considerados entidades externas diante do software que será desenvolvido e, em geral, têm alguma relação com ele. A UML, a título de generalização de conjunto de 'coisas', utiliza um estereótipo chamado 'ator' (*actor*). Dessa maneira, as pessoas, os departamentos e outros sistemas são denotados como atores externos. Os atores externos têm alguma relação com o sistema. Essa relação sempre denota uma responsabilidade que pode ser modelada no diagrama 'use-case'. A relação entre atores e o sistema tem vínculo a uma funcionalidade do software que será desenvolvido, de maneira que se pode antecipadamente conhecer o *que* deverá existir no software, sem a preocupação de *como* isso será implementado.

1.3.2 ANÁLISE SISTÊMICA

Durante a análise sistêmica será feito um estudo de todos os dados e processos verificados na fase anterior (levantamento de requisitos), de maneira que se façam abstrações para identificação de classes, seus atributos e métodos. As classes deverão ser apresentadas em um modelo de maneira que se visualize a estrutura e a forma em que elas deverão interoperar, para tanto, poderá ser empregado o diagrama de classes. Na análise sistêmica só serão modeladas classes que pertençam ao domínio principal do problema, ou seja, classes técnicas que gerenciem banco de dados, interface, comunicação, concorrência e outros que não estarão presentes nesse diagrama.

1.3.3 PROJETO

Nesta etapa extrapola-se o domínio principal do problema do software. Outras classes podem ser adicionadas ao modelo existente para propiciar uma infra-estrutura tecnológica, como a interface do usuário e dos periféricos, o gerenciamento de banco de dados, a comunicação com outros sistemas etc. Trata-se de um aprimoramento da etapa

anterior, cujo resultado será um detalhamento das especificações para que seja possível a programação do software.

1.3.4 IMPLEMENTAÇÃO

Nesta fase ocorre a codificação dos programas de computador, naturalmente empregando uma linguagem orientada a objetos. Essa codificação deve inicialmente estar ocorrendo automaticamente, convertendo-se o modelo de classe para o código da linguagem escolhida. Essa conversão automática será possível dependendo do software CASE que esteja sendo utilizado. No momento, a conversão realizada pelos softwares CASE, do modelo de classes para uma linguagem, gera apenas 'a espinha dorsal' do código. Ainda há a necessidade de intervenção manual para a criação do software. O que se realiza nas etapas anteriores a esta é apenas a criação de modelos que traduzem tecnicamente o significado do entendimento e da estrutura do sistema. A programação é o desfecho onde os modelos criados 'ganham vida'.

1.3.5 TESTES E IMPLANTAÇÃO

Todo software codificado deve sofrer rigoroso e exaustivos testes na busca de erros e conseqüente eliminação dos mesmos. São quatro aspectos que devem ser abordados nesta etapa. O primeiro aspecto são os testes de unidade, onde cada programa, individualmente, é testado. Posteriormente, quando todos os programas tiverem sido testados, faz-se um teste de conjunto. Nada garante que, apesar de terem funcionado individualmente, irão se comportar bem quando executados em conjunto (pois nessa situação outros fatores estão relacionados, performance, compartilhamento etc). Se tudo estiver certo, deve-se partir para testes de integração, quando o software criado estiver algum mecanismo de interface com outros sistemas. Por último será o teste de adequação aos requisitos, com o envolvimento direto do usuário, o qual dará a aprovação final, quando então o software poderá ser implantado.

2. NOTAÇÃO DA UML

Como é impossível representar um sistema na sua completude por meio de um único diagrama, é necessário um conjunto de recursos que expresse os diversos aspectos que compõem o sistema. Pensando neste contexto, a UML possibilita empregar várias notações gráficas que buscam caracterizar o sistema na sua totalidade. O sistema é descrito em facetas (visões), em cada qual observa-se um aspecto particular do sistema, a junção dessas visões mostram o sistema na sua totalidade. Cada visão está composta por um conjunto de diagramas que retratam a particularidade enfatizada pela visão.

2.1 DIAGRAMA DE CASOS DE USO (USE CASES)

O diagrama é usado para descrever o que um novo sistema deverá fazer ou para descrição de um sistema já existente, podendo mostrar como o sistema se comporta em várias situações que podem ocorrer durante sua operação. Deve prever todas as operações que vier a disponibilizar. Naturalmente, por operação, subentende-se macro como procedimentos têm um objetivo completo dentro do contexto geral, por exemplo, em sistema de vendas, ‘cadastrar pedidos’ seria uma operação, ‘cadastrar cliente’ outra.

Originalmente o diagrama foi criado por Ivar Jacobson, baseado em suas experiências no desenvolvimento de um sistema para a Ericsson com a utilização dos métodos OOSE e Objectory. Considerando as cinco fases de desenvolvimento de sistemas mencionadas (análise de requisitos, análise sistêmica, projeto, implementação e implantação), o diagrama *Use Case* está relacionado com a primeira delas, pois os casos de usos são aplicados para capturar os requisitos solicitados pelo cliente. Por meio dessa modelagem pode-se ter um contexto de como será o funcionamento do sistema, sem se preocupar com a implementação do mesmo. Trata-se de um primeiro nível de abstração acerca do sistema.

Um diagrama um diagrama de caso de uso representa uma coleção de *use* e *ator*, permite a representação da relação existente entre eles e, com isso, especifica ou caracteriza a funcionalidade e o comportamento de um sistema. A construção de modelos de casos de uso é feita a partir de várias discussões entre as pessoas envolvidas com o sistema a ser modelado: desenvolvedores, clientes e usuários finais. É necessário um esforço muito grande do analista de sistemas no sentido de reunir todas as informações necessárias sobre cada aspecto que está sendo entendido e avaliado do sistema.

Os clientes e os usuários finais têm interesse nesse tipo de modelagem, pois ela representa toda a funcionalidade do sistema e descreve como ele será usado; sua participação durante a modelagem é fundamental, uma vez que um analista de sistema será um agente que transcreverá aquilo que entender sobre a realidade exposta pelos clientes/usuários e suas necessidades, em especificações técnicas que devem retratar tal realidade. Naturalmente, na medida em que a modelagem vai sendo construída, será constantemente adaptada de forma a refletir em detalhes as necessidades de clientes/usuário. Mais uma vez, embora enfadonha ressaltar, o processo de *feedback* é imprescindível. O analista de sistemas deve sempre repassar o que entendeu com os usuários envolvidos no problema.

É importante que na modelagem a ser construída seja especificado os limites do sistema, definidos pela funcionalidade que é exigida do software. A funcionalidade de todo o sistema é representada por um conjunto de casos de uso que retratam a funcionalidade completa esperada para o sistema. Cada caso de uso por sua vez deve ser extensivamente avaliado, para encontrar todas as possíveis situações de uso daquela funcionalidade que está sendo modelada.

Pode-se dizer que, no diagrama de caso de uso, o sistema se parece com uma 'caixa preta' que oferece funcionalidades. No momento da construção do diagrama, não se deve ter a preocupação quanto aos aspectos de implementação, visto que os principais objetivos da representação são:

- Captar (entender) a funcionalidade necessária para resolução dos problemas existentes, sob a ótica do cliente ou usuários.
- Mostrar uma visão funcional coesa sobre tudo o que o software deverá fazer, pois esse diagrama será a base para todo o processo de desenvolvimento.
- Deverá ser aplicado para testes de validação (O software quando pronto realmente possui a funcionalidade inicialmente planejada).
- Propiciar facilidades para a transformação dos requisitos funcionais em classes e operações reais do software.

Para a criação do diagrama de *use case*, pode-se utilizar um caminho constituído das seguintes etapas:

- Definição do sistema e entendimento macro de seus objetivos.
- Identificar os possíveis atores (quem exerce alguma atividade pertinente ao sistema) e os casos de uso existentes (atividades que envolvem os atores identificados).
- Detalhar várias situações de funcionalidade para os casos de uso.
- Estabelecer os relacionamentos entre os elementos.
- Checar o modelo com usuários e clientes.

O diagrama de casos de uso deve descrever o sistema, seu ambiente e a relação entre os dois. Os componentes desse diagrama são os 'atores' e os 'casos de uso' propriamente dito, conforme mostra a Figura abaixo.

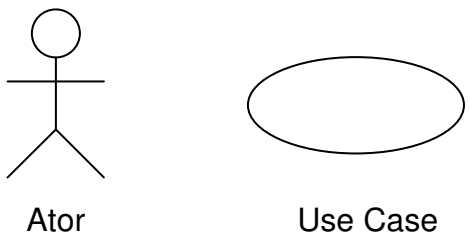


Figura 1. Ator e caso de uso

Pessoas, departamentos e mesmo equipamentos, que possam de alguma forma interagir com o sistema que está sendo modelado, são considerados uma entidade externa ao sistema, constituindo-se o que é chamado de ator (*actor*). Visto que os atores representam as entidades externas do sistema, eles ajudam a delimitá-lo e fornecem uma visão clara do que será realizado. Os *use case* são desenvolvidos de acordo com os eventos que ocorrem entre as entidades externas e o sistema.

Um **ator** representa um tipo de objeto (pessoas, departamentos, máquinas) que interage diretamente com o sistema. Cada *ator* deve ter um nome, como, por exemplo AtorCliente, mostrado na Figura abaixo.

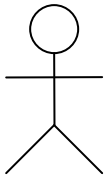


Figura 2. Estereótipo de ator

Actor na UML é a representação de um esteriótipo (*stereotype*). Um *stereotype* tem a capacidade de criar um tipo de elemento de modelagem. Um *stereotype* representa a metaclassificação de um elemento, ou seja, mostra uma classe dentro do metamodelo da UML (isto é, um tipo de elemento de modelagem). Embora existam *stereotypes* já definidos, novos tipos podem ser adicionados.

Qualquer entidade externa ao sistema é representado pelo esteriótipo ator, de maneira que apresenta as seguintes características:

- Ator é externo ao sistema, pode operá-lo; porém, não é parte dele. Representa os papéis que alguém pode desempenhar interagindo com o sistema.
- Ator pode interagir ativamente com o sistema ou com outros atores.
- Ator pode receber informações do sistema.
- Ator pode representar departamento, uma empresa, um ser humano, uma máquina ou outro sistema.

Um ator é tipo de objeto (uma possível classe) e não uma instância. O ator retrata um papel e não um usuário em particular que tenha alguma relação com o sistema. Em uma biblioteca universitária, caso Joaquim queira emprestar um livro, estamos diante de uma atividade desenvolvida pelo papel de usuário da biblioteca. É sobre o papel de usuário que se está interessado conhecer quando modela-se um ator. Modela-se o comportamento diante do sistema, e não a pessoa propriamente dita, como no caso Joaquim, pois, assim como ele, também a Maria no papel de usuária pode fazer a mesma coisa. Dependendo do seu papel em um sistema, uma pessoa pode ser vários atores. Os papéis que alguém pode ter no sistema também podem ser restringidos. Por exemplo, na mesma biblioteca mencionada, uma mesma pessoa pode ser proibida de ter um livro emprestado se ao mesmo tempo ela também exercer o papel (for o ator) atendente do balcão.

Os nomes atribuídos aos atores devem refletir a generalidade dos papéis que desempenham e não uma instância específica.

Os atores devem ser investigados em todos os seus atributos que, de alguma maneira, se exige que o sistema venha a conhecer. No caso de um usuário de uma biblioteca, pode ser necessário que se conheça sobre o ator usuário atributos como: RG, nome, endereço,

telefone. Esses atributos deverão compor parte do encapsulamento de uma classe (conforme será cisto adiante).

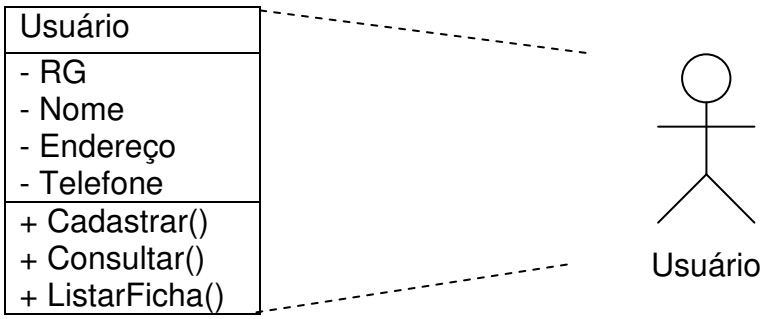


Figura 3. Classe derivada de um ator

Como os atores podem dar origem a classes, eles podem ter os mesmos relacionamentos existentes entre as classes (conforme será visto adiante no diagrama de classes). Nos diagramas de casos de uso, apenas os relacionamentos de *generalização* são usados para descrever um comportamento comum entre um número de atores.

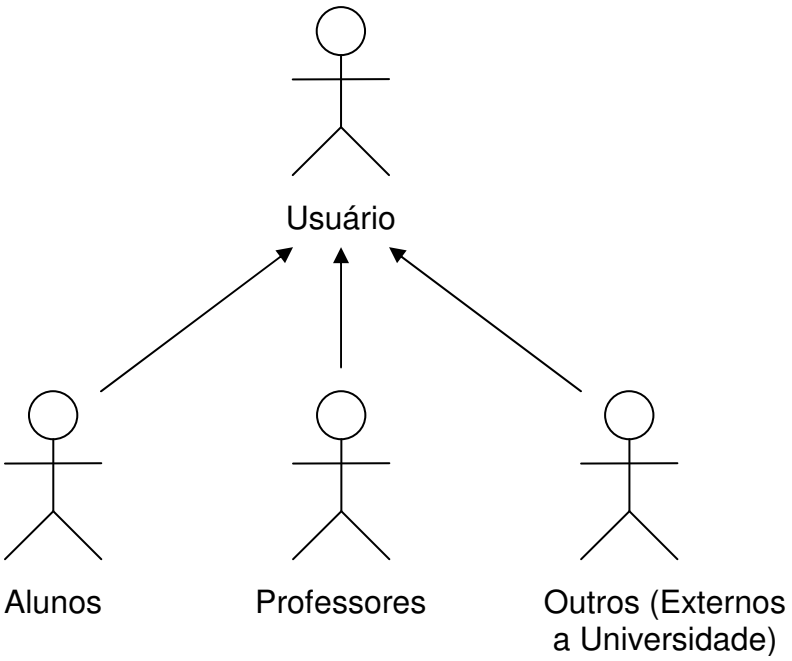


Figura 4. Notação de herança entre atores

De acordo com a definição dada pela UML, um caso de uso é 'um conjunto de seqüências de ações que um sistema desempenha para produzir um resultado observável de valor a um ator específico'. Portanto, um caso de uso representa uma funcionalidade completa na percepção de um ator. Emprestar um livro, por exemplo, trata-se de uma funcionalidade do sistema de biblioteca, sob a ótica do ator usuário.

Um caso de uso é uma atividade ou procedimento composto por uma seqüência de ações que o sistema executa, revela um padrão de comportamento, acionado em geral

por um ator e produz um resultado que contribui para os objetivos do sistema. Algumas características de casos de uso são descritas a seguir:

- Um caso de uso modela a interação entre os atores e o sistema, ou mesmo entre casos de uso.
- Um caso de uso é ativado por um ator ou por um outro caso de uso para acionar uma certa função do sistema, como por exemplo 'cadastrar fornecedor'.
- Um caso de uso é um fluxo de eventos completo e consistente (conjunto de operações que se completam, atingindo um objeto).
- Todos os casos de uso juntos representam todas as situações possíveis de utilização do sistema e mostram toda a funcionalidade existente disponível no sistema.

Após a definição dos atores (ainda que nem todos tenham sido descobertos a priori), os casos de uso podem ser identificados seguindo-se um roteiro que compreende as seguintes questões:

- O software precisará ter quais funções para satisfazer as necessidades de um ator? O que o ator precisa fazer?
- Um ator precisará ter acesso ou informar dados do software? O ator precisa ser notificado sobre os eventos no sistema ou é o ator que precisa notificar o sistema sobre algo?
- É possível simplificar ou melhorar o trabalho do ator mediante a inclusão de novas funções ao sistema, principalmente funções não automatizadas?

As questões acima pressupõem como ponto de partida a existência dos atores já identificados. Deve-se acrescer as seguintes questões para completude da visão de casos de uso, o que pode levar à identificação de atores ainda não identificados:

- De que entrada ou saída o sistema precisa? De onde elas vêm e para onde vão?
- Quais são os principais problemas com a implementação já existente do sistema?

Um *use case* é visualmente mostrado como uma elipse com um nome, que poder ser colocado acima, dentro ou abaixo do símbolo, como por exemplo, CadstrarCliente, mostrado na Figura abaixo.

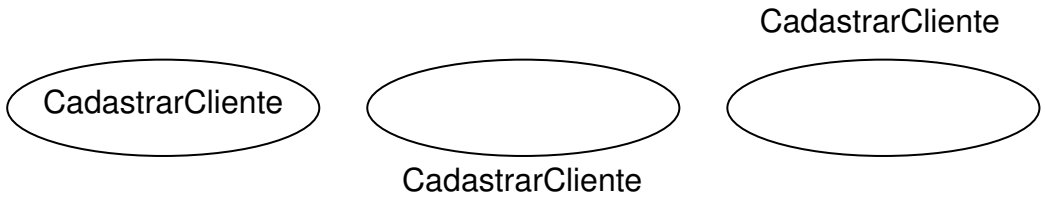


Figura 5. Exemplos de use case

Para que um diagrama de caso de uso seja rigorosamente avaliado, emprega-se o conceito de cenário. Um caso de uso deve ser avaliado sob a ótica de vários cenários, o que permitirá avaliar sua completude da funcionalidade. Deve-se criar tanto cenários quantos forem necessários. Os cenários são situações de uso informal para validação do sistema com relação ao caso de uso em particular.

A criação de cenários é decorrente da atividade de análise e especificação dos requisitos, os quais agora são modelados. Antes de descrever os cenários, o analista deve ter entrevistado o cliente e os usuários, bem como feito as observações *in loco*

necessárias, de maneira que se tenha certeza quanto ao entendimento daquela situação em particular, a qual irá retratar. As entrevistas feitas propiciaram aos usuários falarem sobre suas tarefas e os problemas associados a cada uma delas. A observação direta *in loco* realizada deve ter permitido que o analista tenha entendido a situação de uso como ela realmente vem ocorrendo na prática.

Os procedimentos do usuário (usuário *versus* atividades) pode ser melhor entendida quando o analista procura descrever situações do processo de trabalho, que consistem de uma coleção de narrativas de situações no domínio que favorecem o levantamento de informações, a identificação de problemas e a antecipação de soluções. As atividades realizadas pelas pessoas é o foco dos cenários a serem criados, uma vez que tal procedimento possibilita uma perspectiva ampla quanto a visualização dos problemas atuais no domínio descrito.

A criação dos cenários, além do entendimento do domínio do problema, se presta a estimular novos questionamentos, possibilitando que se encontre alternativas para o desenvolvimento do software, o que implica que os cenários, via de regra, não precisam apresentar uma visão absolutamente precisa sobre a realidade. Novas características que estejam sendo planejadas aos procedimentos existentes podem ser vislumbradas na exploração dos cenários. Como fica determinada a situação se for acrescido determinado procedimento? Como fica a situação se o procedimento atual for realizado de tal forma? Quais as conseqüências se for extinto determinado procedimento, ou se pular-se determinada etapa, ou ainda forem incorporadas novas funcionalidades?

Após a elaboração dos possíveis cenários, os quais certamente possuem certo grau de imprecisão, o analista deve reunir-se com os usuários envolvidos e validar sua modelagem discutindo cada cenário desenhado (para cada cenário um diagrama de caso de uso). Os diagramas podem ser afixados em quadros, na parede ou projetados por recursos de *datashow* onde os participantes possam analisá-los e fazer comentários, possivelmente redesenhando possíveis trechos na medida em que o debate de idéias se realiza, modificando-se os cenários existentes. Somente depois dessa discussão é que realmente o analista terá uma definição quanto aos possíveis cenários de uso de uma determinada atividade no sistema, e só então conseguirá construir 'definitivamente' os diagramas de casos de uso para tais abordagens. Contudo, não se deve esquecer que o cenário obtido irá mudar com o tempo, pois os requisitos que deram origem a ele, mudam; o que se espera é que o software possa ir evoluindo (sem muitos traumas) acompanhando os novos requisitos que virão a existir.

Necessariamente, não é obrigatório que o analista construa logo de início todos os diagramas de caso de uso para cada situação constatada, ou diante das sugestões de funcionalidades ainda inexistentes, mas que deverão estar disponíveis no software a ser construído. Pode-se em um primeiro momento constuir cenários empregando-se narrativas textuais ou por meio de *storyboards*, muito embora, aparentemente, desenhar os casos de uso é menos dispendioso. As narrativas textuais podem ser descritas livremente, identificando os agentes e as ações que eles participam, o problema neste caso é tentar evitar possíveis ambigüidades, uma vez que o texto é livre.

O *storyboarding* é um roteiro (textual), podendo-se fazer acompanhar de um quadro ilustrativo da cena. Emprega-se na estruturação de propagandas de televisão e mesmo no cinema. É uma forma muito natural de lidar com a descrição de cenário, porque

apresenta uma cena que foca uma situação, na qual são descritas as ações que os atores desempenham. A título de exemplo, segue uma forma de representação textual para descrever cenário e na seqüência a representação dos mesmos cenários com aplicação do diagrama de caso de uso.

Cena 1	Usuário solicita um livro com um certo conteúdo
Agentes Envolvidos	Usuário, Atendente e Bibliotecária
	Usuário entra na biblioteca e dirige-se ao balcão onde está a atendente:
Usuário	Eu gostaria de emprestar um livro sobre modelagem orientada a objetos.
Atendente	Algum título específico?
Usuário	Não, mas gostaria que abordasse o padrão estabelecido pela OMG.
Atendente	Você se recorda do nome desse padrão?
Usuário	Não, esqueci.
	A atendente pergunta à bibliotecária:
Atendente	Você sabe o nome do padrão que a PMG estabeleceu para a modelagem orientada a objetos?
Bibliotecária	Hummm. Me lembro que é uma sigla curta. Acho que começa com U. UXL, não, não...UML acho que é isso...
Usuário	É isso mesmo.
	Em seguida, a atendente faz a consulta por palavra-chave e descobre todos os livros disponíveis que têm UML como conteúdo. O cliente escolhe um e o leva emprestado.

Cena 2	Usuário procura livros sobre análise e projeto de sistemas, conhecendo alguns autores
Agentes Envolvidos	Usuário, Atendente e Bibliotecária
	Usuário entra na biblioteca e dirige-se ao balcão onde está a atendente:
Usuário	Eu gostaria de emprestar livros sobre análise e projeto de sistemas.
Atendente	Algum específico? Algum autor?
Usuário	Bem, pode ser do Chris Gane, Yourdon ou Coad.
	Atendente procura para ver disponibilidades:
Atendente	Temos esses aqui.
	A atendente mostra quatro títulos disponíveis. O usuário escolhe dois. A atendente encaminha o usuário à bibliotecária para os procedimentos de retirada dos livros. Depois de preencher a ficha do usuário, registrando a retirada, a bibliotecária passa os livros para o usuário.

Esses mesmos cenários podem ser representados empregando-se o diagrama de casos de uso. Deve-se escolher uma outra forma, desde que estejamos falando de um momento inicial da análise, antes da validação da modelagem, pelos envolvidos no cenário. Após

a validação, necessariamente deve ter-se o diagrama de casos de uso que representa o processo.

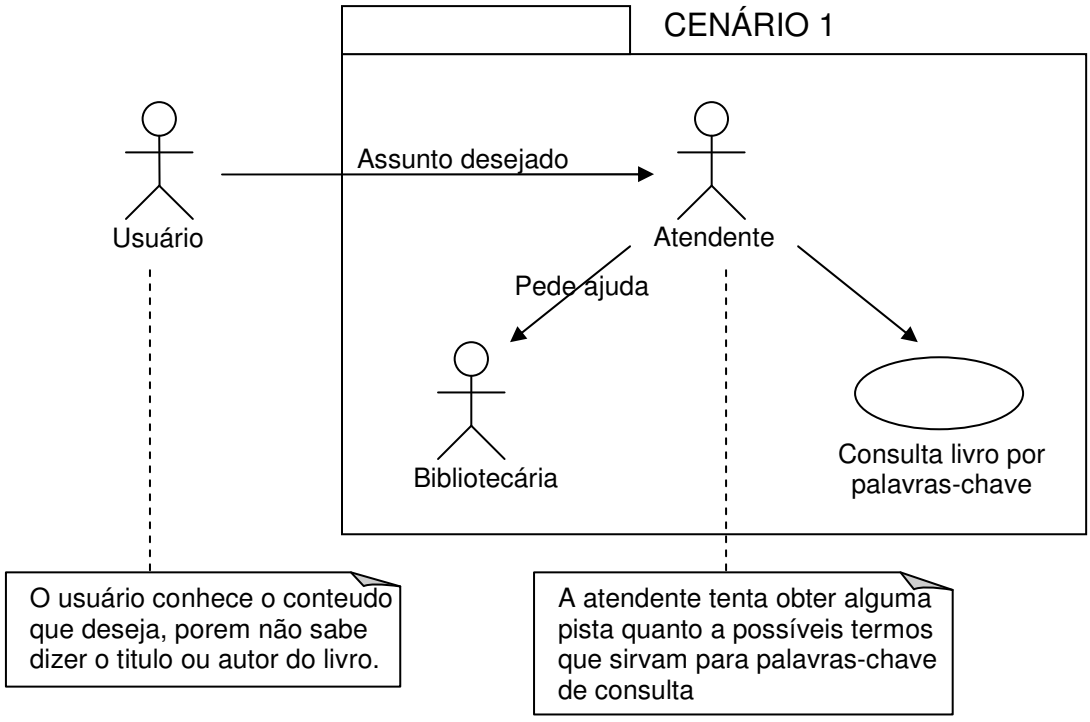


Figura 6. Primeiro cenário

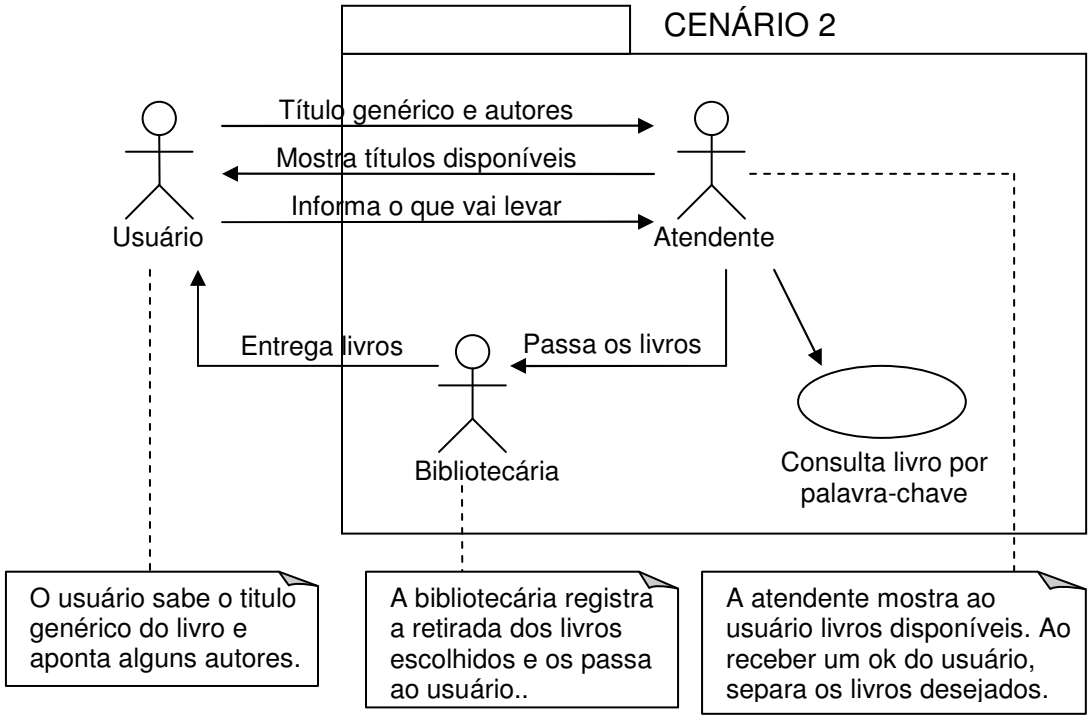


Figura 7. Segundo cenário

Após os cenários estarem desenvolvidos, o analista deve trabalhar em conjunto com os usuários para avaliá-los e refiná-los, pois tanto a forma descritiva textual quanto o diagrama de casos de uso são amigáveis. O analista pode explicar os cenários aos usuários para checar se a modelagem realmente retrata o que ocorre na realidade, ou se retrata a nova funcionalidade ainda existente.

Quando se chegar a um modelo que os usuários tenham aprovado, deve-se desenhar um diagrama de caso de uso que integre os cenários; assim, considerando-se como exemplo o caso de uso 'empréstimo do acervo' se conseguiria a representação de um modelo onde estaria previsto todas as possibilidades em termos de formas de empréstimo.

O resultado da modelagem por meio de cenários é uma base para a compreensão de quem são os agentes (atores) envolvidos e quais as atividades que devem ser previstas quanto a um aspecto particular do software que será desenvolvido. Um software é o resultado da união de vários casos de usos, e cada um deles possui diversos cenários a serem investigados.

Nos exemplos de casos de uso apresentados, que representam cenários de uma situação de empréstimo de livros, verifica-se a existência de relacionamento entre casos de uso e atores. Esse relacionamento é chamado de *associação*, que é representado por uma conexão semântica entre o caso de uso e o ator. As associações são unidirecionais. O nome da associação é usado para identificar o propósito do relacionamento.

A Figura a seguir apresenta um diagrama de caso de uso com relacionamento de associação: o ator Usuário relaciona-se por intermédio da associação "Título, Autor ou Editora", com o *use case* Consultar Obras. O use case Consultar Obras relaciona-se com o ator Usuário por meio da associação Obras Existentes.

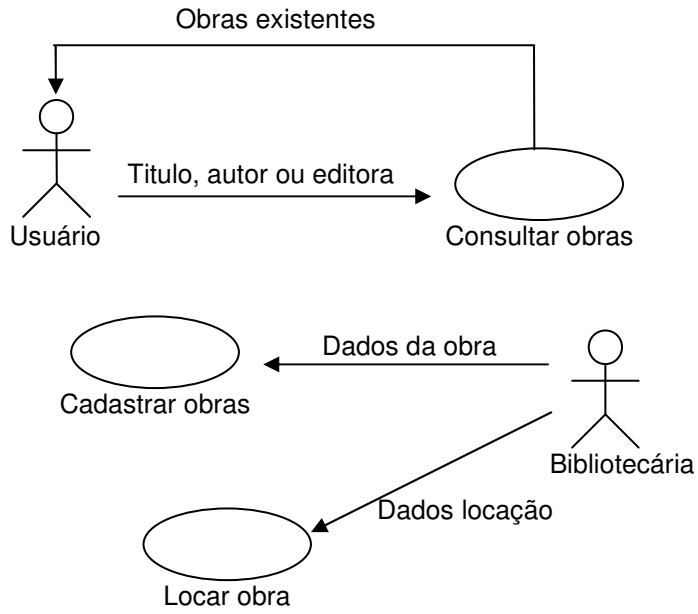


Figura 8. Exemplo relacionamento de associação

Casos de uso podem compartilhar ações executadas por outros casos de uso. Essa situação expressa um relacionamento chamado de *generalização*. A generalização em casos de uso é empregada para descrever o comportamento comum entre dois ou mais *use cases*; portanto, é um dos mecanismos utilizados para identificar comportamentos reutilizáveis.

A Figura apresenta um diagrama de *use case* com relacionamento de generalização: o ator aluno conecta-se através do relacionamento de associação 'dados consulta', como o *use case*' "Consulta Notas". O *use case* "Consultar Notas" conecta-se com o *use case* "Identificar Aluno", por meio do relacionamento de generalização, o qual faz a identificação do aluno. O caso de uso "Consultar Notas" relaciona-se com o aluno por intermédio da associação 'notas'.

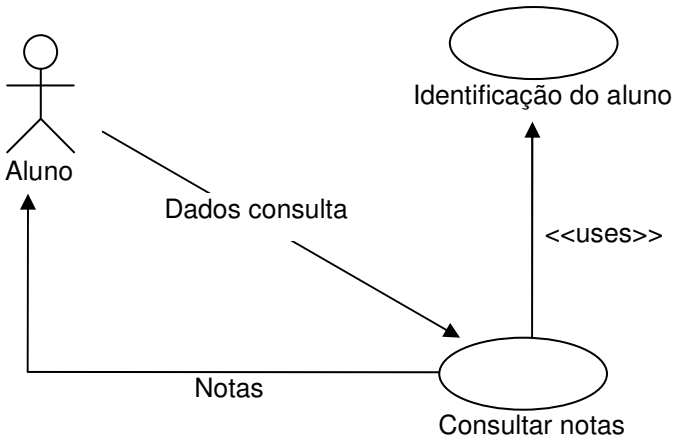


Figura 9. Exemplo de um relacionamento de generalização

O relacionamento de generalização da Figura está empregando o esteriótipo << *uses*>>. Esse esteriótipo no relacionamento de generalização indica que será obrigatório que o caso de uso 'consultar notas' acione o comportamento expresso pelo caso de uso 'identificação do aluno'. Outro esteriótipo possível de ser empregado em relacionamentos de generalização nos casos de uso é o <<*extends*>>. Ele é empregado para expressar comportamento opcional por um *use case*. Por exemplo, a Figura mostra uma generalização <<*extends*>> do *use case* "Consultar Notas" para o *use case* "Atualizar dados cadastrais aluno". A generalizacao <<*extends*>> indica que o use case "Consultar Notas"poderá ou não utilizar o use case "Atualizar dados cadastrais aluno".

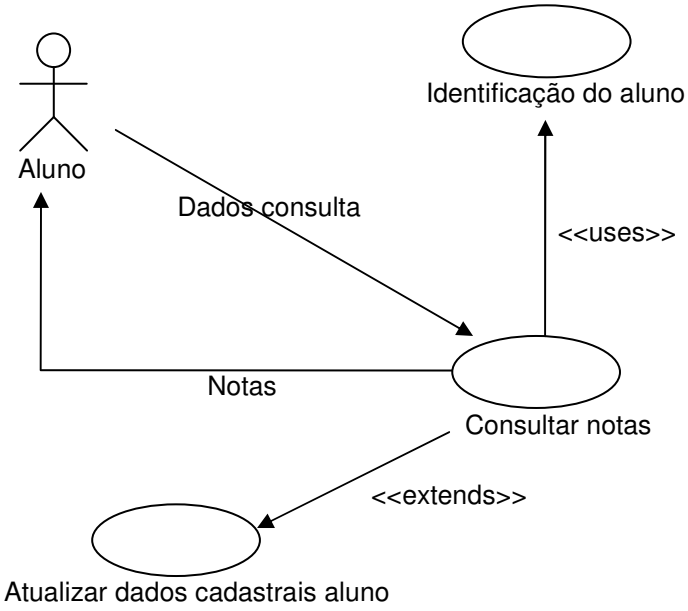


Figura 10. Exemplo de generalização <<extends>>

Diagrama use case pode ser empregado para:	**Fase**
Capturar os requisitos do sistema e compreender o que o sistema faz.	Analise
Especificar o comportamento do sistema que será implementado e/ou especificar as semânticas do mecanismo do use case.	Projeto

Tabela 1. Uso do diagrama use case

2.2 DIAGRAMA DE CLASSES

O diagrama de classes expressa a estrutura estática de um sistema, pois aquilo que é descrito é sempre válido em qualquer ponto no ciclo de vida do sistema. No diagrama de classes também é mostrada a possibilidade de interações entre classes, pois não se constituem elementos isolados e com absoluta autonomia; na verdade, muitas tarefas somente são possíveis de serem realizadas pela colaboração existente entre as classes. A notação padrão usada pela UML para representar uma Classe de Objetos é:

NomeClasse
<atributos>
<métodos>

A classe de objeto é representada por um retângulo, subdividido em três áreas. A primeira contém o nome da classe, a segunda contém seus atributos e a terceira contém seus métodos (funções/procedimentos). Uma classe representa um conjunto de objetos que tenham a mesma estrutura e comportamento. É uma abstração de objetos do mundo real. Os objetos são instâncias da classe. As classes são utilizadas para classificar os objetos identificados no mundo real; sendo assim, elas devem ser retiradas do domínio do problema e serem nomeadas pelo que elas representam no sistema. Essa atividade deve ser feita por alguém com muito conhecimento do domínio do problema. Ao

empregar-se o símbolo de classe, a UML prevê uma sintaxe de desenho e escrita dos elementos que a constituem, conforme a Figura abaixo.

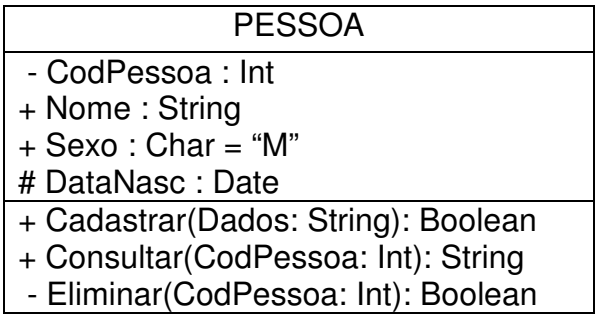


Figura 11. Exemplo de uma classe

Ao escrever o nome de um atributo ou de um método, a UML sugere uma sintaxe a ser seguida.

Com relação aos atributos a proposta geral é:

Visibilidade NomeAtributo: TipoDoAtributo = ValorDefault {propriedade}

- Visibilidade

  Trata-se de uma marcação que pode ser realizada pelos símbolos (+, #, -), ou ainda pela aplicação de ícones. O elemento visual a ser empregado deve indicar uma das possibilidades abaixo:

  (+) Visibilidade pública – é acessível por todas as classes (trata-se do valor *default*).

  (#) Visibilidade protegida – pode ser visto pela classe e pelo pacote no qual a classe é definida.

  (-) Visibilidade privada – somente acessível pela própria classe.

- NomeAtributo

  Seqüência de caracteres que devem formar um nome auto-explicativo criado pelo analista que denota o conteúdo que se pretende armazenar. Tipicamente inicia-se com uma letra.

- TipoDoAtributo

  Expressa o tipo de conteúdo que se pretende armazenar para o atributo. Como está intimamente ligado à linguagem de programação, a UML deixa definida a sintaxe para esse elemento.

- ValorDefault

  Refere-se ao conteúdo inicial do atributo, de acordo com o seu tipo.

- {propriedade}

  Trata-se de um elemento opcional, que complementa informações a respeito do atributo, acomodando uma descrição sobre o atributo, representando claramente seu propósito e domínio de valores.

Com relação aos métodos, a sintaxe geral sugerida é:

- Visibilidade

  Equivalente à mesma representação utilizada para os atributos.

- NomeDoMétodo

  Palavra criada pelo analista que representa a operação que será processada. Pode ser formada pela concatenação de duas ou mais palavras: obterSaldo, consultarDepositoBancário, atualizarDados etc.

- Parâmetro

  Tara-se de uma lista de valores devidamente separados por vírgula, pois, para cada um, o método tem uma necessidade claramente definida.

- TipoDeRetorno

  Equivalente à definição existente para os atributos.

- {propriedade}

  Equivalente à definição existente para os atributos.

Com base nos elementos padronizados para descrição visual de uma classe, um software CASE poderá ser capaz de 'entender' a especificação visual e gerar um correspondente trecho em código-fonte, de acordo com uma linguagem de programação previamente escolhida, conforme exemplo na Figura abaixo.

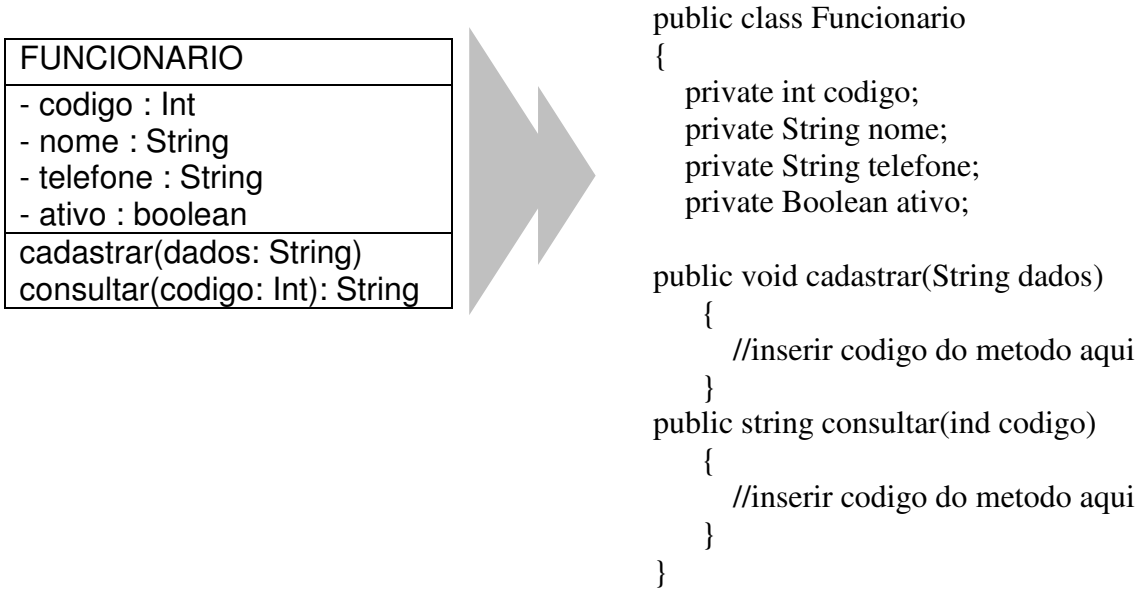


Figura 12. Transformação automática de classe em código-fonte (no caso, JAVA)

2.2.1 RELAÇÕES ENTRE CLASSES

As classes podem apresentar quatro tipos de relações: *herança, dependência, associação e agregação*.

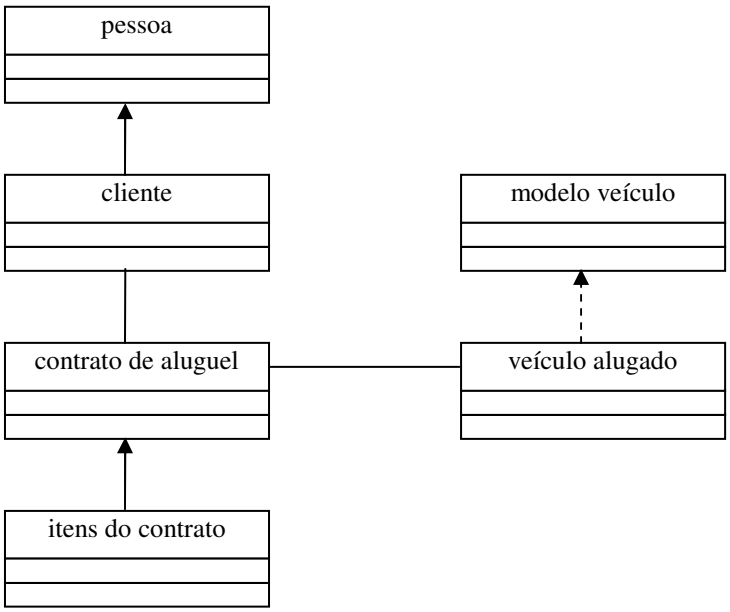


Figura 13. Exemplo com relacionamento de associação e herança

Por herança, como relacionamento entre classes, subentende-se que a subclasse compartilha toda a estrutura e o comportamento da superclasse (classe mãe ou metaclasse). Na figura 13, o cliente herda a estrutura e o comportamento da pessoa, lê-

se cliente " é-uma" pessoa. A notação de herança é uma forma de representar hierarquias entre classes, mostrando uma estrutura do mais geral (generalização) para algo mais específico (especialização).

Na Figura 13, a classe "Veículo alugado" depende da existência do ' modelo veículo'. Essa notação indica que, se houver uma mudança em algum objeto da classe independente (modelo de veículo, no exemplo), pode afetar outro objeto da classe dependente (Veículo alugado).

A classe 'contrato de aluguel' é um agregado de 'itens do contrato'. A Figura 13 também mostra uma relação de associação entre 'contrato de aluguel' e 'cliente', bem como entre 'contrato de aluguel' e 'veículo alugado'. Todos esses tipos de relações são explicados em maiores detalhes a seguir.

- Relacionamento de herança

O relacionamento de herança induz a dois conceitos básicos, conforme mostra a Figura 14. Nas laterais da figura estão dispostos graficamente os conceitos inerentes ao relacionamento entre as classes pessoa, cliente e fornecedor. Pessoa é uma generalização de cliente e fornecedor, ou seja, tanto cliente quanto fornecedor possui atributos e métodos comuns. Cliente ou Fornecedor é um tipo de pessoa (sua especialização). A classe raiz (genérica) é chamada de classe mãe, superclasse ou metaclasse. A classe que herda as características da classe mãe é chamada de subclasse ou classe filha.

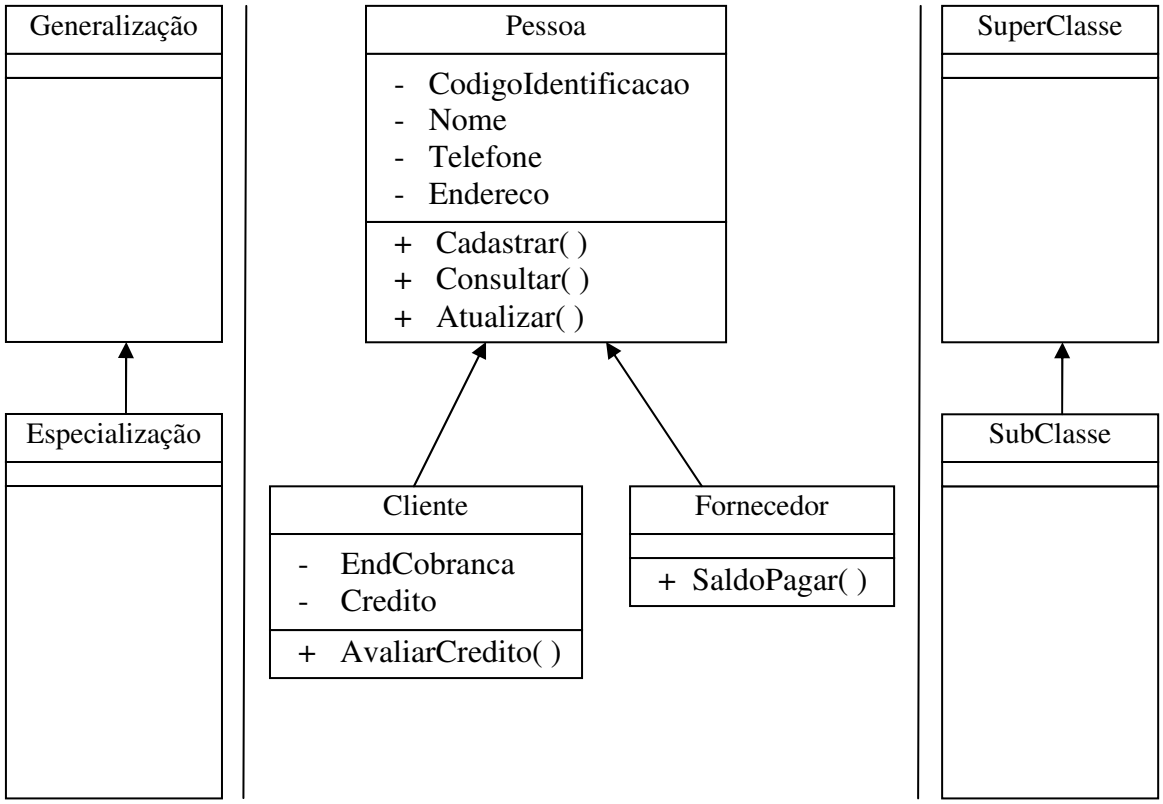


Figura 14. Exemplo de relacionamento de herança entre classes (generalização/especialização)

- Relacionamento de dependência

Um relacionamento de dependência entre duas classes mostra que uma instância de uma classe depende da instância de outra classe, normalmente chamada cliente/servidora respectivamente. Por exemplo, a Figura 13 mostra que a classe "Veículo Alugado" depende da classe " Modelo Veículo" para existir; denota-se no caso específico que um veículo para ser alugado só existirá se houver um modelo daquele veículo previamente existente. Um outro exemplo de dependência é representado por uma lista de presença em aulas, conforme a Figura 15.

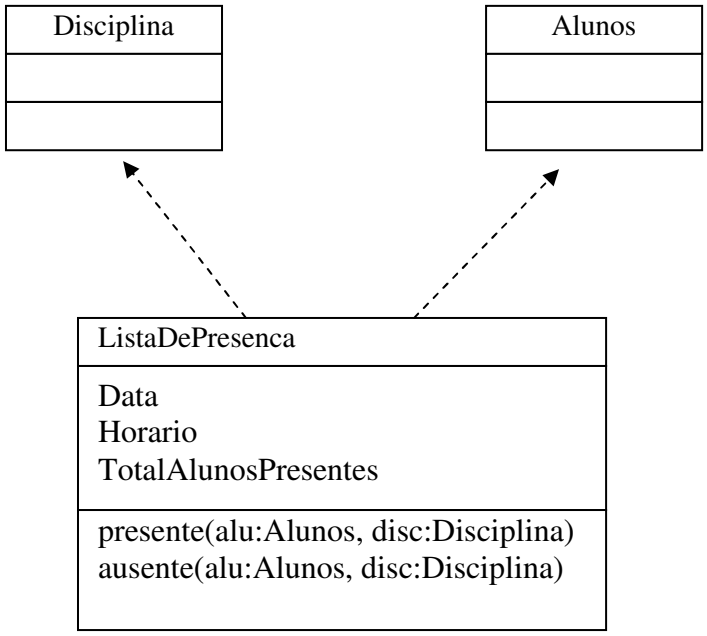


Figura 15. Exemplo de dependência

Uma instância da classe ListaDePresença (Figura 15) depende de Alunos e Disciplinas, pois seu método *presente* utilizará a informação de aluno e de disciplina, cujo objetivo é marcar como presente um aluno em uma determinada disciplina, em uma data e horário; caso exista uma marcação indevida, o método *ausente* será acionado para corrigir a situação.

- Relacionamento de associação

Um relacionamento de associação representa uma conexão semântica entre duas classes e é o relacionamento mais utilizado. É bidirecional e é representado por uma linha ligando as classes. Na Figura 13 tem-se o relacionamento de associação entre o 'contrato de aluguel' e a classe "cliente", além de 'contrato de aluguel' com a classe "veiculo alugado".

O relacionamento de associação representa uma dependência estrutural entre objetos; em geral, provenientes de classes diferentes, pode possuir um nome que deve estar próximo a linha que representa a associação. Uma associação pode apresentar o conceito de multiplicidade (conforme a Tabela 2) e navegabilidade conforme o exemplo da Figura 16.

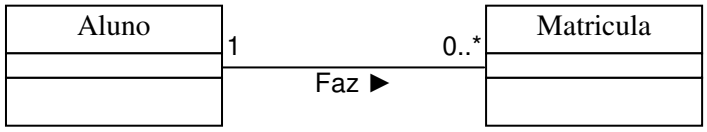


Figura 16. Exemplo de associação binária

A associação com navegabilidade (ponta de seta) indica que, ao estabelecer uma associação, a mesma só pode ocorrer na direção indicada pela ponta de seta. No exemplo da Figura 16, tem-se uma associação binária, mas ainda é possível encontrar-se outros tipos de associação: unária e ternária (Figura 17).

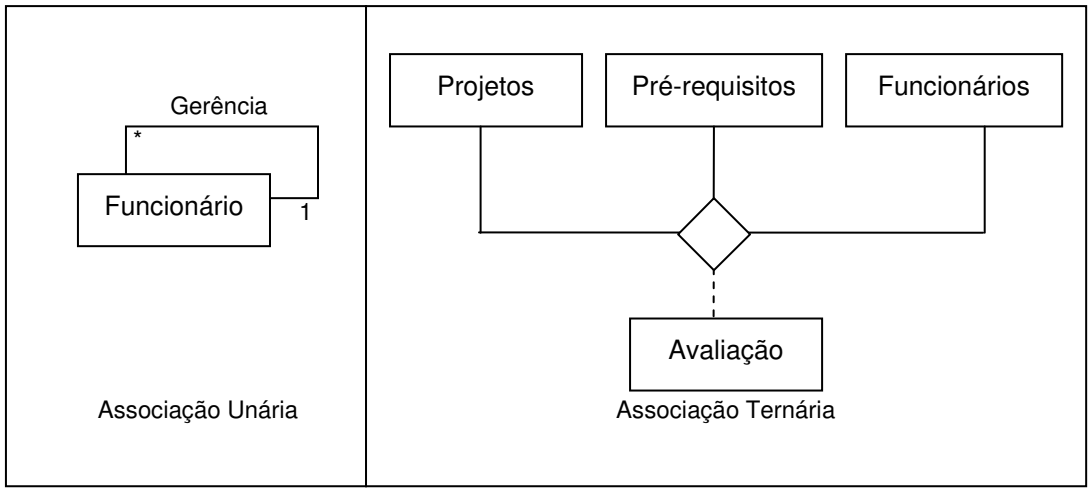


Figura 17. Associação unária e ternária

A associação unária também é conhecida como associação recursiva, pelo fato de ser um relacionamento entre objetos da mesma classe. As associações não estão limitadas ao conjunto de três classes participantes (ternária), como o exemplo da Figura 17 pode induzir. Pode-se representar associações *n-ária*, embora isso não seja comum.

- Relacionamento de agregação

Um relacionamento de *agregação* é uma forma especial de associação, que é usada para mostrar que um tipo de objeto é composto de outro objeto. Um relacionamento de agregação é também chamado de 'todo-parte'. Conforme mostra a Figura 18, "Pedido" é um agregado de "itens do Pedido". O relacionamento de agregação possui duas formas de representação, com significados diferentes. A *agregação por valor* (losango cheio) indica que o tempo de vida das partes são dependentes do tempo de vida do todo. Um item de pedido existe quando existe o pedido, e vice-versa. Na *agregação por referência* (losango sem preenchimento0, o tempo de vida das partes não são mutuamente dependentes do tempo de vida do todo.

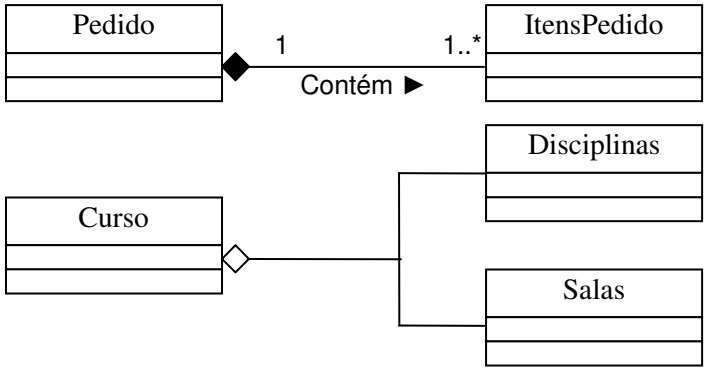


Figura 18. Exemplo de agregação

2.2.2 MULTIPLICIDADE

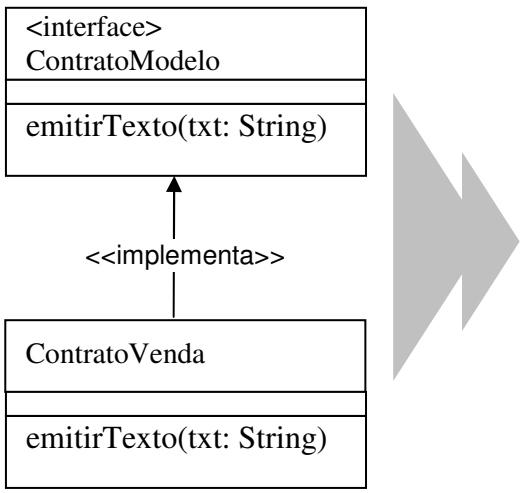
Nos relacionamentos de associação e de agregação, pode-se acrescentar a *multiplicidade* (similar à cardinalidade na modelagem estruturada), que especifica o número de instancias de uma classe em relação a outra em um relacionamento, por meio do número mínimo e máximo. A tabela 2 resume as possíveis variações da multiplicidade, onde <literal> é qualquer inteiro maior ou igual a 1.

Notação	Significado
0..1	Zero ou uma instância
1	Somente uma instância
0..*	Zero ou mais instâncias
*	Default, numero mínimo e máximo de instância são ilimitados
1..*	Uma ou mais instâncias
<literal>..*	Numero exato ou mais instâncias

Tabela 2. Notações da multiplicidade

2.2.3 INTERFACE

Há um tipo de classe a qual não pode ser instanciada, ou seja, não se conseguirá gerar objetos diretamente dela, o que a torna uma classe virtual, servindo apenas para especificar as operações externamente visíveis para uma classe. Uma interface descreve os padrões legais de interação entre dois objetos. A interface funciona como uma classe modelo, que outras classes poderão fazer uso, implementando as funcionalidades descritas.




```
public interface ContratoModelo
{
   public void emitirTexto(String txt);
}

public class ContratoVenda implements
Contrato Modelo
{
     public void emitirTexto(String txt)
     {
        //inserir codigo do metodo aqui
     }
}
```

Figura 19. Exemplo da classe Interface

2.3 DIAGRAMA DE INTERAÇÃO

Todos os aspectos vistos até este momento foram derivados de dois diagramas básicos: diagramas de casos de uso e diagrama de classes. Pode-se dizer que tais diagramas representam a parte estática de um sistema e, portanto, não estão qualificados para representar temporais ou de colaboração que possam existir entre os objetos das classes.

O diagrama de interação na verdade não existe; é um termo genérico aplicado à junção de dois outros diagramas: seqüência e colaboração. O diagrama de interação visa construir a modelagem comportamental ou dinâmica do sistema. Isso é possível através dos diagramas de seqüência e colaboração que juntos conseguem demonstrar o comportamento dos objetos, considerando-se a seqüência da troca de mensagens existentes entre eles, para que se cumpra um determinado papel ou se atenda a determinado contexto. Ao trocar mensagens para atingir determinado objetivo, se estabelece um contexto de colaboração entre os objetos.

Os diagramas de seqüência e colaboração favorecem a identificação de responsabilidades que as classes poderão ter, uma vez que as mensagens trocadas pelos objetos correspondem a métodos que devem estar previstos nas respectivas classes.

Para avaliar uma interação, é necessário eleger um caso de uso. Com foco em um caso de uso específico, se busca identificar quais objetos participam da interação. Na medida em que se vai identificando os objetos envolvidos, percebe-se a forma como eles estão relacionados, o que vem facilitar o entendimento de como deve-se estabelecer associações entre classes no diagrama de classes, bem como quais métodos devam existir para determinadas classes.

- Diagrama de seqüência

No diagrama de seqüência mostra-se a interação entre objetos com a preocupação de documentar os métodos (funções/procedimentos) executados ao longo do tempo, conforme mostra a Figura 20.

Diagramas de seqüência documentam interação organizada em uma seqüência de tempo entre os objetos participantes de uma operação e as trocas de mensagens entre eles. Um diagrama de seqüência possui duas dimensões: vertical, que representa o tempo; e horizontal, que representa o tempo; e horizontal, que representa diferentes objetos (se for necessário, as dimensões podem ser invertidas).

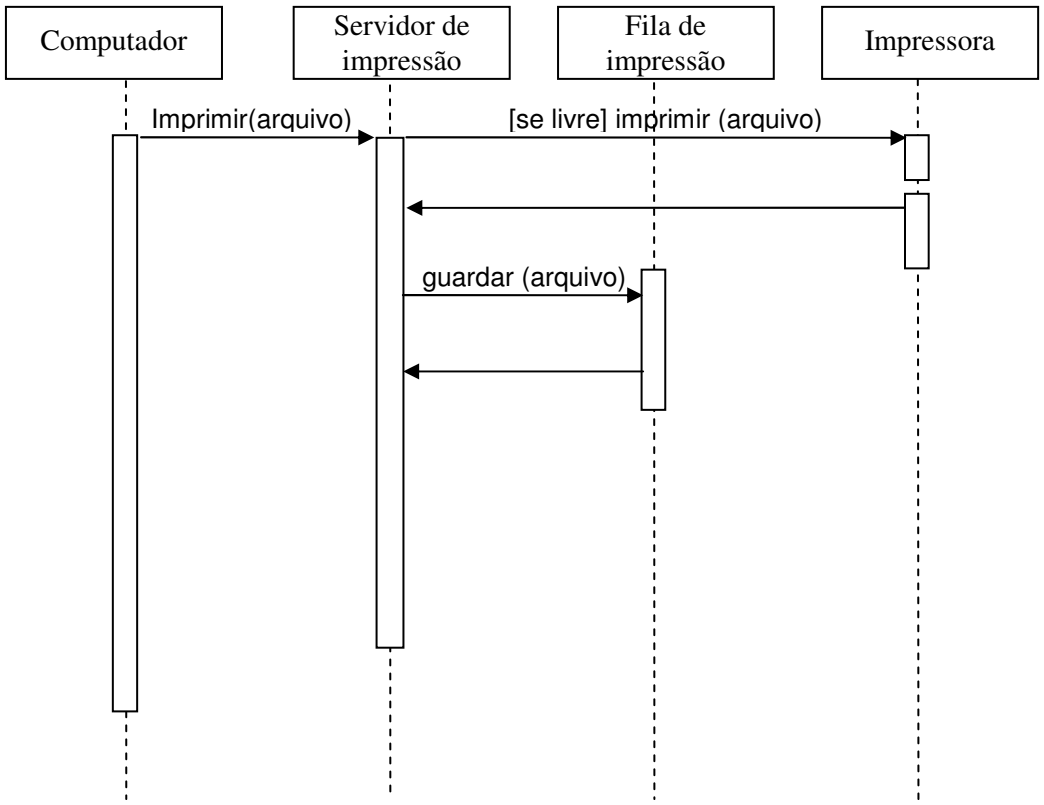


Figura 20. Exemplo de generalização diagrama de seqüência

- Diagrama de colaboração

É um modo alternativo para representar a troca de mensagens entre um conjunto de objetos, mostra a interação organizada em torno dos objetos e suas ligações uns com os outros, sem a preocupação de expressar a vida útil das mensagens no tempo. O diagrama de colaboração não mostra a dimensão do tempo, por isso as seqüências de mensagens e linhas concorrentes devem ser determinadas usando a seqüência de números.

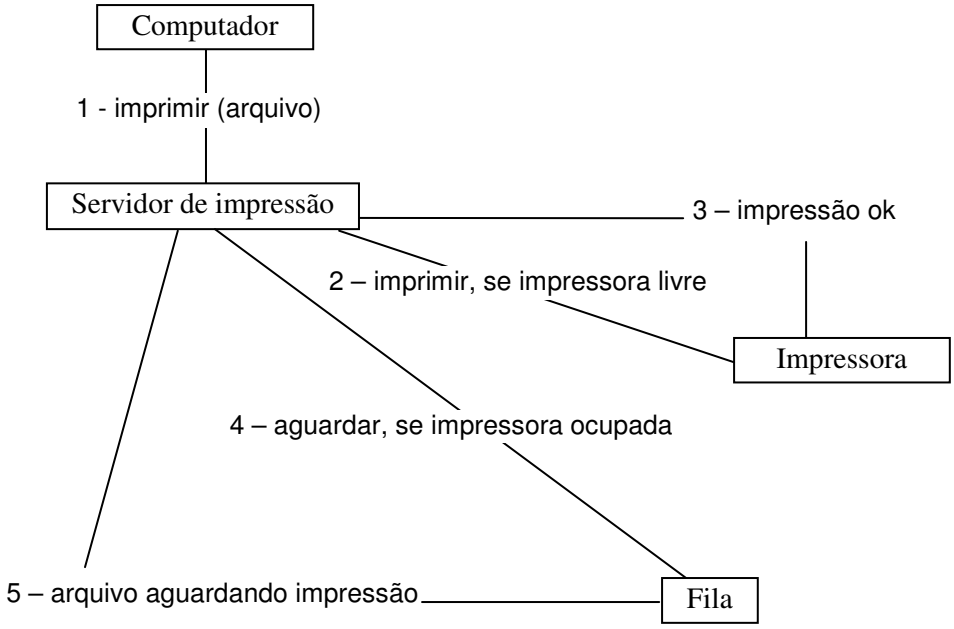


Figura 21 Exemplo de diagrama de colaboração

2.4 DIAGRAMA DE ESTADO

Normalmente, um sistema aberto reage a estímulos provenientes de fora dele ou ainda estímulos temporais por ele mesmo desencadeado. Essa reação pode originar respostas externas ao sistema. Essa dinâmica existente nos sistemas é fruto da colaboração entre objetos, os quais estarão em determinado estado em um certo momento no tempo.

O Diagrama de Estados é usado para mostrar os possíveis estados dos objetos de uma classe. A mudança de um estado para outro é chamado de *transição de estados*. Os eventos do diagrama de estados causam uma transição de um estado para outro e as ações resultam na mudança de estado. Cada diagrama de transição de estados está associado a uma classe ou a um diagrama de transição de estados de um nível mais alto.

O início de um diagrama de estados é indicado pelo chamado *estado inicial*, cuja representação gráfica é um circulo preenchido. Na verdade, ele não expressa um estado especifico do objeto da classe, indica apenas o inicio do diagrama. Na seqüência, se conecta o primeiro estado real com uma transição, rotulada ou não. Em cada diagrama de estado há somente um estado inicial. A notação gráfica de um estado inicial é mostrada no Figura 22.

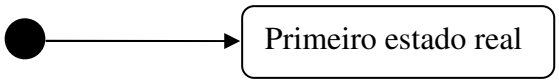


Figura 22. Exemplo de estado inicial

Um *estado* demonstra uma situação no tempo de algum aspecto do sistema sobre o qual existe interesse de controle. Durante a vida de um objeto, pode vir a existir controle sobre várias situações, cada qual podendo assumir diversos estados possíveis no tempo. Um objeto permanece em um estado por um tempo finito. A notação gráfica e um exemplo de estado são mostrados na Figura 23.

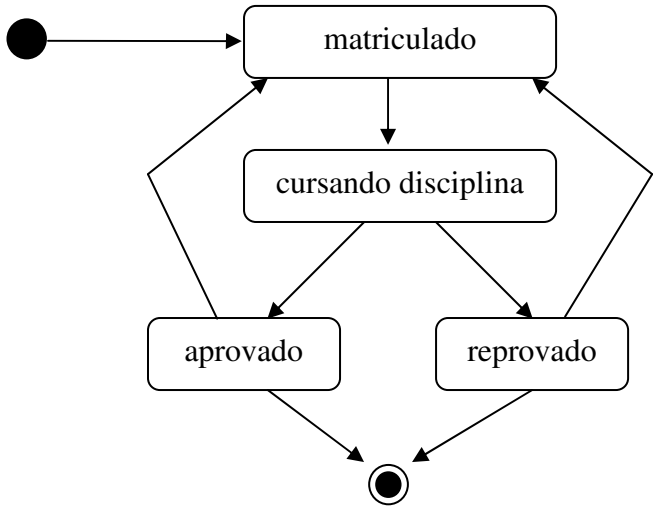


Figura 23. Representação de um estado do objeto

O *estado final* representa a termino é do ciclo previsto de controle para mudanças de estados de um dos aspectos do sistema. Na figura 24 verifica-se a representação gráfica do estado final, mediantes dois círculos, sendo o interno totalmente preenchido.

A mudança de um estado para outro é chamado de *transição de estados*. Trata-se de um relacionamento entre dois estados. Para o objeto passar de um estado para outro, é necessário dois mecanismos: condição e ação. A condição, se existir, deverá ser satisfeita para que a ação seja executada. Essa ação é responsável pela transição dos estados.

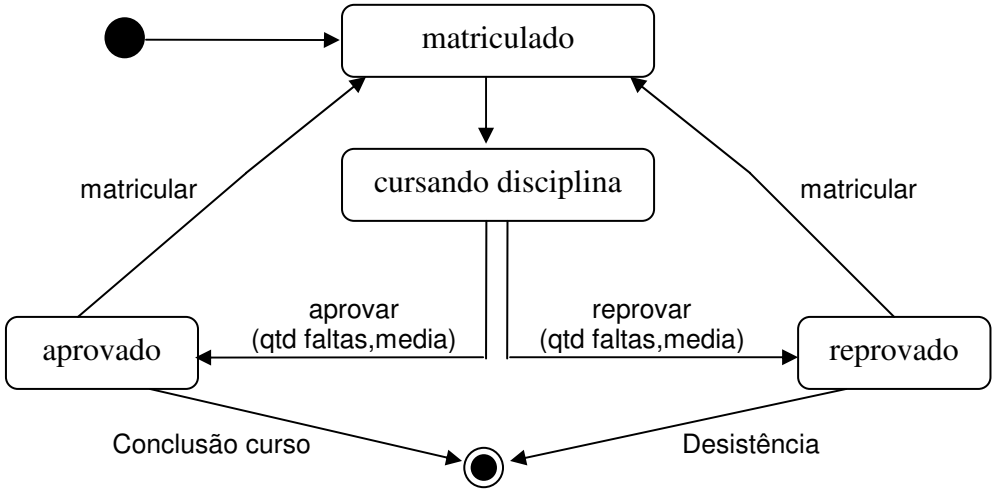


Figura 24. Exemplo de transição de estados

TONSIG, Sérgio Luiz, Engenharia de Software, Ed. Futura, 2003, São Paulo.