

Enyo José Tavares Gonçalves  
Mariela Inés Cortés

# Análise e Projeto de Sistemas

2012

Copyright © 2012. Todos os direitos reservados desta edição à SECRETARIA DE EDUCAÇÃO A DISTÂNCIA (SEAD/UECE). Nenhuma parte deste material poderá ser reproduzida, transmitida e gravada, por qualquer meio eletrônico, por fotocópia e outros, sem a prévia autorização, por escrito, dos autores.

---

**Presidente da República**

Dilma Vana Rousseff

**Ministro da Educação**

Aloizio Mercadante

**Secretário de Educação a Distância**

Carlos Eduardo Bielschowsky

**Diretor do Departamento de Políticas em  
Educação a Distância – DPEAD**

Hélio Chaves Filho

**Sistema Universidade Aberta do Brasil**

Celso José da Costa

**Governador do Estado do Ceará**

Cid Ferreira Gomes

**Reitor da Universidade Estadual do Ceará**

Francisco de Assis Moura Araripe

**Vice-Reitor**

Antônio de Oliveira Gomes Neto

**Pró-Reitora de Graduação**

Josefa Lineuda da Costa Murta

**Coordenador da Secretaria de Educação a Distância**

Antonio Germano Magalhães Junior

**Coordenador Geral UAB/UECE**

Francisco Fábio Castelo Branco

**Coordenadora Adjunta UAB/UECE**

Eloísa Maia Vidal

---

**Coordenador da Licenciatura em Informática**

Joaquim Celestino Junior

**Coordenador De Tutoria e Docência da Licenciatura em Informática**

Maria Wilda Fernandes

---

**Design instrucional**

Antonio Germano Magalhães Junior

Igor Lima Rodrigues

Pedro Luiz Furquim Jeangros

**Coordenador Editorial**

Rafael Straus Timbó Vasconcelos

**Projeto gráfico**

Rafael Straus Timbó Vasconcelos

Marcos Paulo Rodrigues Nobre

---

**Diagramação**

Marcus Lafaiete da Silva Melo

**Revisão**

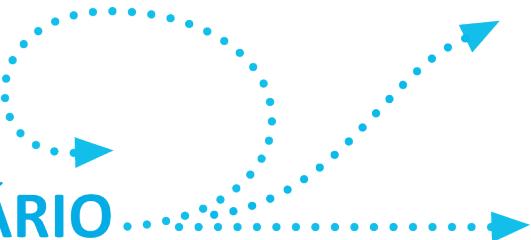
Professor a ser designado pelo curso

**Ilustração**

Marcos Paulo Rodrigues Nobre

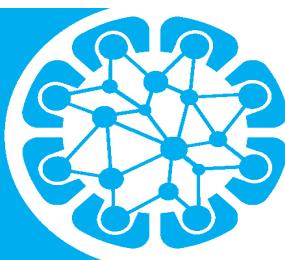
**Capa**

Emilson Pamplona Rodrigues de Castro



# SUMÁRIO

Apresentação .....	5
<b>Capítulo 1</b>	
Contextualização .....	7
1. Introdução .....	9
2. Introdução à UML .....	11
3. Ferramentas de modelagem .....	16
<b>Capítulo 2</b>	
Elicitação e especificação de requisitos, análise e projeto de sistemas orientados à objeto ...	21
1. Elicitação e especificação de requisitos .....	23
2. Análise .....	35
3. Projeto .....	38
<b>Capítulo 3</b>	
Diagramas da UML - Parte 1 .....	43
1. Diagrama de Casos de Uso .....	45
2. Diagrama de Classes.....	50
<b>Capítulo 4</b>	
Diagramas da UML - Parte 2 .....	59
1. Diagrama de Sequência.....	61
2. Diagrama de Atividades .....	68
<b>Capítulo 5</b>	
Outros diagramas da UML e padrões de projeto .....	77
1. Outros diagramas da UML .....	79
2. Padrões de projeto.....	88
<b>Dados dos Autores</b> .....	98





# APRESENTAÇÃO

É importante que os sistemas desenvolvidos sejam documentados e que estes artefatos permitam uma visão do sistema antecipadamente (antes que o mesmo seja implementado), de modo que clientes e profissionais de TI consigam expressar as características de acordo com suas necessidades através de uma linguagem padronizada.

Quanto antes as alterações forem realizadas em um projeto menos custo é envolvido para implementar estas alterações. Projetar um sistema antes de implementá-lo é um aspecto importante relacionado à possibilidade de alterações em fases iniciais.

Assim sendo a modelagem de sistemas colabora com a documentação do sistema, comunicação entre as partes envolvidas, possibilidade de identificação de correções, dentre outras vantagens.

O presente livro apresenta de forma clara e amigável os princípios de modelagem orientada à objeto, de acordo com a Linguagem de Modelagem Unificada, que vem sendo adotado com sucesso em organizações de desenvolvimento de Software. Organizações nos mais diversos segmentos de atuação estão cada vez mais preocupados na adoção de métodos e modelos de processo como um meio de alcançar maior competitividade no mercado através da satisfação dos seus clientes.

O livro está organizado em oito unidades. A primeira unidade fornece as bases necessárias para o entendimento dos conceitos básicos relacionados às atividades de modelagem, além de mostrar a evolução das linguagens de modelagem, UML e ferramentas envolvidas. A Unidade 2 apresenta conceitos relacionados à Engenharia de Requisitos e diferencia as fases de Análise da fase de Projeto, citando as principais características de cada uma delas. A Unidade 3 apresenta os diagramas de casos de uso e de classes da UML. A Unidades 4 apresenta os diagramas dinâmicos de sequencia e de atividades de UML. Finalmente a Unidade 5 trata dos demais diagramas da UML e de Padrões de Projeto.

O conteúdo apresentado no presente livro destina-se principalmente a professores e alunos de graduação em Ciências da Computação ou áreas afins, fornecendo um embasamento teórico e uma clara noção dos princípios e estratégias a serem seguidos no desenvolvimento de software de qualidade, com foco na plena satisfação do cliente.

## Os Autores





# Capítulo

# 1

## Contextualização

### Objetivos:

- O desenvolvimento de software há muito deixou de ser uma tarefa artesanal, consequentemente o uso de um processo de software é cada vez mais explorado. É necessário o entendimento de alguns conceitos ligados à processo como a própria definição e o encadeamento das fases, para que Requisitos, análise e projeto sejam pormenorizados.
- Nesta unidade também são introduzidos conceitos de modelagem de sistemas e da linguagem de modelagem UML, o uso de ferramenta case para capaz de modelar aplicações utilizando UML será apresentado, bem como os principais recursos oferecidos pela linguagem de modelagem.
- Estes conceitos são importantes no contexto atual do desenvolvimento de sistemas, uma vez que boa parte das aplicações são projetadas utilizando esta tecnologia. Os conhecimentos apresentados nesta unidade serão utilizados como base para as todas as demais unidades.





## 1. Introdução

O desenvolvimento de software há tempos deixou de ser visto como uma atividade artesanal e, cada vez mais, técnicas de engenharia vem sendo introduzidas para que o produto final atenda a custo, prazo e qualidade desejados. A utilização de um processo de desenvolvimento de software é uma das técnicas de engenharia de software que representa uma necessidade em fábricas de software (empresas cujo setor de atuação é o desenvolvimento de software), no setor de Tecnologia da Informação (TI) de empresas que desenvolvem atividades de outra natureza ou por profissionais de TI que atuam como prestadores de serviço em desenvolvimento de software.

Um **processo de desenvolvimento** de software é formado por um conjunto de fases que devem ser seguidas para que o produto (software) seja produzido. Cada uma das fases é composta por tarefas com entradas (O que é necessário para que a tarefa seja realizada) e saídas (O que é produzido pela tarefa) específicas e com o papel associado (Quem deverá realizar a tarefa). Estas atividades são compostas por passos.

Cada empresa cria seu processo de desenvolvimento de acordo com suas necessidades, mas de um modo geral poderíamos estabelecer algumas **fases** que boa parte das empresas utiliza em seus processos de acordo com a Figura 1.

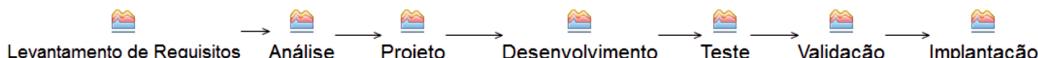


Figura 1 – Fases do processo de desenvolvimento de software

1. A atividade de Levantamento de Requisitos tem o objetivo de estabelecer quais funcionalidades o sistema a ser desenvolvido deverá ter, documentando isto através do documento de requisitos.
2. A fase de Análise tem como entrada o documento de requisitos e é responsável por gerar diagramas utilizando uma linguagem de modelagem específica, com o intuito de realizar uma primeira aproximação da descrição do documento de requisitos com a implementação;
3. A fase de Projeto tem o objetivo de detalhar os diagramas modelados na fase de análise, além de estabelecer a arquitetura que será utilizada para o desenvolvimento do sistema;
4. A fase de Desenvolvimento tem o objetivo de implementar o sistema em uma determinada linguagem de programação, tendo como base os modelos criados na fase de análise e evoluídos na fase de projeto;
5. A fase de Teste tem o objetivo de submeter o software desenvolvido a algumas situações e analisar os resultados produzidos por este com a finalidade de verificar se o mesmo está de acordo com requi-

! ATENÇÃO

Se a empresa baseia-se no desenvolvimento iterativo e incremental, o desenvolvimento de um único projeto poderá passar por todo este fluxo várias vezes.

sito que foi utilizado para implementá-lo. Além disto, o teste aborda algumas situações que o software é vulnerável a falhas para ver se estas acontecem. Outros testes ainda podem ser feitos relacionados às diversas finalidades como verificar o desempenho do sistema;

6. Na fase de Validação, o cliente analisa se o produto gerado atende suas necessidades;
7. A fase de Implantação tem o objetivo de disponibilizar o uso do sistema no ambiente de sua operação.

Como foi frisado anteriormente, pode haver variação entre os autores da área em relação à quais fases devem aparecer. Neste contexto, muitas vezes as fases de levantamento e análise são vistas como uma única fase chamada de Engenharia de Requisitos. Portanto, a compreensão de como ocorre o levantamento de requisitos é de fundamental importância ao entendimento das fases de análise e projeto.

Este livro concentra-se nas técnicas de engenharia utilizadas nas fases de Levantamento de Requisitos, Análise e Projeto.



## ATIVIDADES DE AVALIAÇÃO

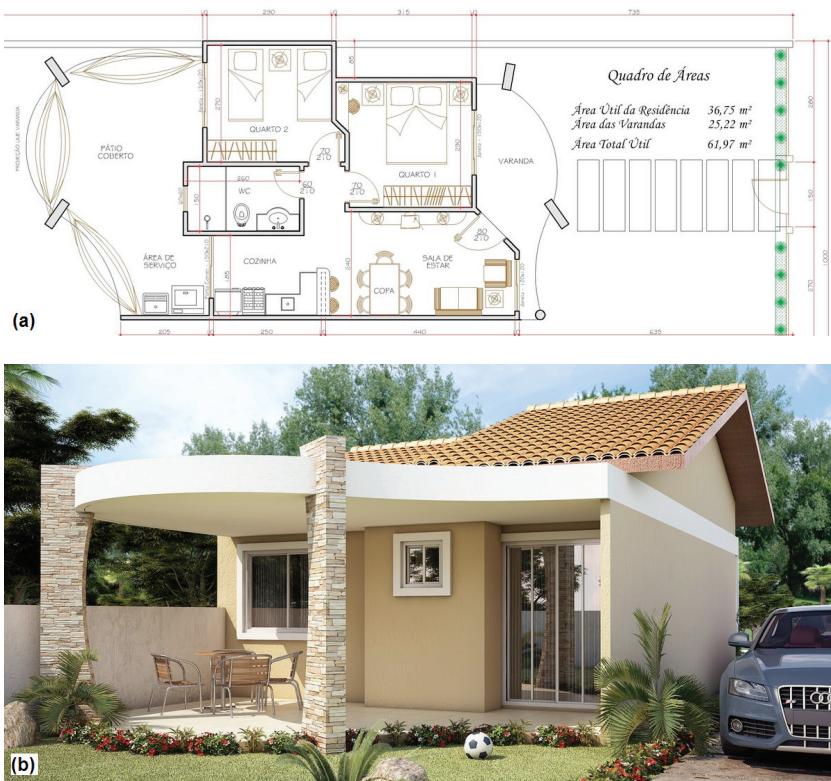
1. Qual a importância de um processo para o desenvolvimento de software?
2. Relacione as atividades do desenvolvimento de software que você conhece com as fases presentes na Figura 1.



## 2. Introdução à UML

# Analogia

Quando uma casa ou prédio serão construídos o engenheiro civil ou arquiteto realizam o projeto desta construção antes que a obra inicie. O projeto é constituído de desenhos que conseguem expressar as características do imóvel que será construído a partir de diferentes pontos de vista. A **planta baixa** (Figura 2.a) do imóvel consegue expressar a localização da parte hidráulica, o dimensionamento dos cômodos e a abertura das portas dos quartos, mas não consegue mostrar como será a estética do imóvel, a combinação das cores das paredes, o estilo das portas, a iluminação. Para expressar esses pormenores, geralmente é criado um projeto 3D (Figura 2.b) para o imóvel.



**Figura 2 – Planta baixa (a) e projeto 3D (b) de uma casa**

Ambos são importantes para a construção. Sem a planta baixa, erros podem ser cometidos na localização das portas ou na parte hidráulica, levando a gastos posteriores para consertá-los. Do mesmo modo que sem

Um paradigma de programação é um modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns. Os paradigmas mais populares são o paradigma estruturado e o paradigma orientado à objetos.

## SAIBA MAIS

Modelagem = construção de modelos (ou diagramas). Um modelo é uma simplificação da realidade que descreve um sistema através de representação gráfica.

o projeto 3D o contratante pode não ter uma boa visualização do que será construído e o produto final corre o risco de não agradá-lo visualmente.

Na construção de uma ponte ou de um prédio em terreno alagado ou um açude, vários riscos extras estão envolvidos. Neste caso surge a necessidade de cálculos adicionais e a necessidade de expressar graficamente outras nuances do projeto.

Observe que na construção civil o projeto é extremamente importante, ele pode fazer a diferença entre um projeto bem sucedido e o fracasso do mesmo. Os modelos funcionam de forma complementar (Planta baixa e projeto 3D, por exemplo) e em alguns casos é necessário representar as características adicionais.

Em computação não é diferente, há a necessidade de projetar o sistema antes que o mesmo seja construído. Para tanto uma notação gráfica também é utilizada composta por diferentes representações (Diagramas), onde cada representação expressa um conjunto de características que a aplicação a ser desenvolvida deve possuir.

Cada **paradigma de Programação** possui um conjunto de características que as diferenciam. As técnicas de **modelagem** de sistemas são propostas de acordo com o paradigma vigente. Para o paradigma estruturado as técnicas de modelagem comumente utilizadas são DFD (Diagrama de Fluxo de dados), MER (Modelo Entidade-Relacionamento) e DER (Diagrama Entidade-Relacionamento), já o paradigma orientado tem uma linguagem de modelagem padrão: *Unified Modeling Language* (UML).

## Guerra dos Métodos

O surgimento da Orientação a Objetos ocorreu nos anos 60, o uso dessa tecnologia disseminou-se pelas universidades e na indústria do software. A disseminação deve-se à criação da orientação a objeto, a aplicação destes conceitos no desenvolvimento de aplicações cada vez mais complexas crescia e a necessidade por uma linguagem de modelagem era clara.

Para suprir esta necessidade, alguns pesquisadores propuseram linguagens de modelagem entre 1988 e 1992. Podemos citar Grady Booch (OOAD); Peter Coad (OOA e OOD); Ivar Jacobson (OOSE); Jim Odell; Rumbaugh (OMT); dentre outros. Estes métodos eram muito parecidos, mas possuíam algumas diferenças entre si.

Durante esta época, qualquer tentativa de padronização era ignorada. Portanto nos anos 90, dada a existência destas inúmeras formas de modelagem orientada a objetos, o período ficou conhecido como a época da guerra dos métodos.

A unificação iniciou-se em 1995, quando Rumbaugh (OMT) e Booch passaram a trabalhar juntos na Rational Software e decidiram fundir seus métodos (e notações) resultando no Método Unificado. Jacobson juntou-se a eles mais tarde e seu método OOSE também foi incorporado.

Com isto a notação de projetos foi unificada e chamada de UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada). Além das notações de Booch, OMT e OOSE, a UML também agrega as ideias de inúmeros autores, tais como Harel e seu diagramas de estados, Shlaer-Mellor e o ciclo de vida dos objetos. Em suma, UML é uma tentativa de padronizar os artefatos de análise e projeto: modelos semânticos, sintaxe de notação e diagramas.

Em meados dos anos 90 surgiu a OMG (Object Management Group) (<http://www.omg.org/>) a qual teve papel fundamental em relação à UML, uma vez que Booch, Rumbaugh e Jacobson eram funcionários da Rational e a UML corria o risco de tornar-se uma linguagem de modelagem de propriedade da Rational. Este fato não chegou a concretizar-se e em 1997 a UML foi aprovada como um padrão pela OMG.

A Figura 3 mostra cronologicamente as evoluções ocorridas desde a guerra dos métodos até a versão final de UML. É importante observar que a ultima alteração que aparece no diagrama é a versão 2.0 de UML, porém a linguagem encontra-se em processo constante de melhoria e algumas subversões têm sido definidas, como a versão 2.3 em 2010. O diagrama expressa a evolução somente até a versão 2.0 por ter sido a ultima grande mudança da linguagem.

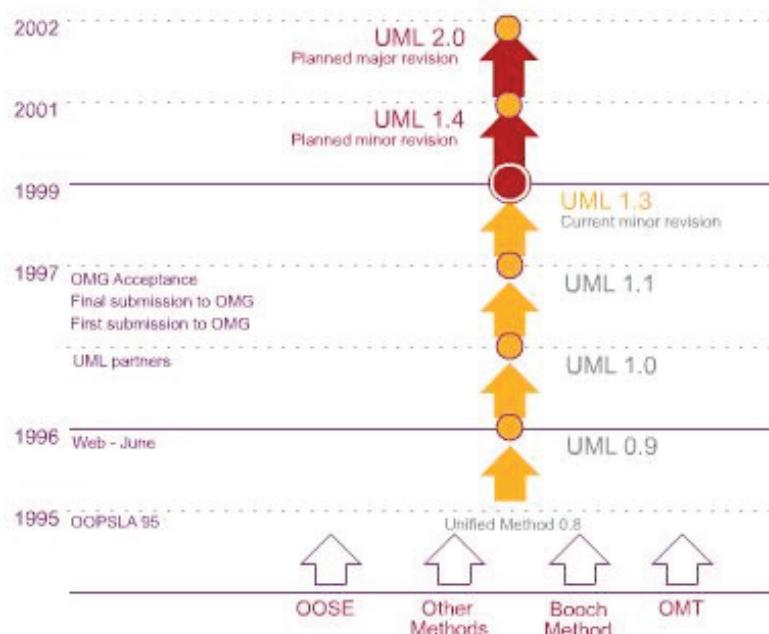


Figura 3 – Origem e evoluções de UML

## UML

Através de UML é possível especificar, visualizar e documentar os elementos de um sistema Orientado a Objetos. A UML é importante, pois:

1. Serve como linguagem para expressar decisões de projeto que não são óbvias ou que não podem ser deduzidas do código;
2. Provê uma forma concreta e suficiente para a compreensão das pessoas e para ser manipulada pelas máquinas.
3. É independente de linguagem de programação, ou seja, pode ser utilizada para desenvolvimento de um sistema através das principais linguagens de programação orientadas a objetos como Java, C++ ou PHP 5;
4. É independente dos métodos de desenvolvimento, ou seja, pode ser utilizada para modelar sistemas em organizações que utilizam métodos ágeis, RUP (*Rational Unified Process*) ou outro método como base.

Tenha muita atenção:

- **UML não é uma metodologia de desenvolvimento nem processo de desenvolvimento:** A UML define apenas a representação gráfica que deve ser utilizada para modelar sistemas orientados a objeto, ela não define fases ou como a linguagem deve ser utilizada ao longo das fases de um processo. Por estes motivos, não pode ser chamada de metodologia de desenvolvimento;
- **Não Dependente de ferramentas:** Para modelar sistemas orientados a objetos através de UML, o analista não necessariamente utilizará uma ferramenta para realizar a modelagem. A modelagem pode ser feita utilizando papel, algum editor gráfico ou uma ferramenta de modelagem. É importante frisar que apesar de não obrigatórias, as ferramentas de modelagem são extremamente importantes, pois aumentam a produtividade dos projetistas e evitam que os mesmos cometam vários erros de má formação dos diagramas criados.

Como o próprio nome sugere a UML é uma linguagem gráfica orientada a objetos. Cada elemento gráfico possui uma sintaxe e uma semântica, sendo que a sintaxe específica como o elemento deve ser desenhado (Uma classe deve ser representada através de um retângulo com três compartimentos, por exemplo). Já a semântica define o que significa o elemento e com que objetivo ele deve ser utilizado.

Estes elementos podem ser classificados em:

1. **Entidades:** Representam blocos de construção da orientação a objetos como Classes, interfaces, pacotes, anotações.
2. **Relacionamentos:** São utilizados para conectar as entidades que possuem alguma ligação entre si como herança, agregação e chamada de método;
3. **Diagramas:** Disponibilizam diferentes visões do sistema. Cada diagrama possui um conjunto de entidades e de relacionamentos que podem ser representados.

Segundo a OMG, os diagramas de UML são classificados em três categorias: Diagramas Estruturais (ou Estáticos), Diagramas Comportamentais (ou Dinâmicos) e Diagramas de Interação.

- **Diagramas Estruturais:** Tratam o aspecto estrutural tanto do ponto de vista do sistema quanto das classes, a partir da representação de seu esqueleto e estruturas “relativamente estáveis”. Os aspectos estáticos de um sistema de software abrangem a existência e a colocação de itens como classes, interfaces, colaborações, componentes. Os diagramas estruturais de UML são os diagramas de: Classes, Objetos, Componentes, Estrutura Composta, Pacotes e Implantação.
- **Diagramas Comportamentais:** Descrevem o sistema computacional modelado quando em execução, isto é, a modelagem dinâmica do sistema. São usados para representar as partes que “sofrem alterações”, como o fluxo de atividades, a interação entre usuários e o sistema e a transição dos estados. Os diagramas comportamentais de UML são os diagramas de: Casos de Uso, Atividade e Máquina de Estados.
- **Diagramas de Interação:** Também descrevem o sistema computacional modelado quando em execução, isto é, a modelagem dinâmica do sistema. São utilizados para representar a comunicação entre

entidades e a sequencia de operações a ser invocada. Os diagramas comportamentais de UML são os diagramas de: Sequência, Comunicação (ou Colaboração), Tempo e Interatividade (Visão geral de interações).



## ATIVIDADES DE AVALIAÇÃO

- 1.** Qual a importância da UML?
- 2.** Selecione duas abordagens antecessoras da UML e encontre semelhanças e diferenças entre as características destas abordagens e a UML.
- 3.** Entre no site da UML (<http://www.uml.org>) ou pesquise em outras fontes os novos recursos acrescentados à UML na versão 2.0.



### 3. Ferramentas de modelagem

Na atividade de desenvolvimento, as ferramentas auxiliam a criação de código com mais qualidade e menor esforço, aumentando a produtividade de programadores. Geralmente estas ferramentas são desenvolvidas para encontrar erros à medida que o desenvolvedor digita os comandos de seu programa e possuem depuradores que facilitam a descoberta do trecho que causou a falha. Eclipse e Dev são exemplos de ferramentas de desenvolvimento, conhecidas como IDE (*Integrated Development Environment* – Ambiente de Desenvolvimento Integrado).

Paralelamente, as ferramentas de modelagem têm um papel central nesta atividade. A sua importância advém do fato de que ferramentas projetadas tomando como base os princípios estabelecidos na linguagem propiciam a boa formação dos modelos gerados, reduzindo erros e aumentando a produtividade na execução das atividades de análise e projeto.

Várias ferramentas de modelagem têm sido propostas tomando por base a UML, podemos citar:

- **astah** - <http://astah.net/>
- **Rational Rose** - <http://www-01.ibm.com/software/awdtools/developer/rose>
- **Jude** - <http://jude.change-vision.com/jude-web/product/index.html> (Versão anterior do astah)
- **ArgoUML** - <http://argouml.tigris.org/>
- **DIA** - <http://projects.gnome.org/dia/>
- **Fujaba** - <http://www.fujaba.de/>
- **StarUML** - <http://staruml.sourceforge.net/en/>
- **Umbrello** - <http://uml.sourceforge.net/>
- **Visual Paradigm** - <http://www.visual-paradigm.com/download/>

Muitas vezes o software de modelagem disponibiliza duas versões: uma gratuita e outra paga. Este é o caso do astah, que disponibiliza uma versão gratuita: o astah community e outras versões astah Professional e astah UML que são do tipo Trial (gratuitas apenas para teste). A diferença entre elas encontra-se nas funcionalidades disponíveis, com o a versão Professional é possível gerar código a partir dos diagramas e realizar engenharia reversa (Gerar diagramas a partir do código Java, C++ ou C#), o que não é possível com a versão Community. A ferramenta é de fácil uso e o instalador está disponível para download em [http://members.change-vision.com/files/astah\\_community](http://members.change-vision.com/files/astah_community).

Para criação de diagramas basta dois passos: A criação de um novo projeto no menu File -> new. Depois disto, salve seu projeto (Ctrl + s) e pode começar a criar os diagramas. Para tanto, selecione o diagrama desejado no

menu Diagram, neste caso o novo diagrama aparecerá com seus respectivos elementos associados. A Figura 3 ilustra a visão geral do Astah Community em execução, juntamente com alguns dos seus componentes principais. As letras representam os seguintes recursos:

- a) **Structure** - Tem como funcionalidade central permitir a organização dos projetos e diagramas em uma estrutura de árvore a fim de que seja possível um melhor gerenciamento e manipulação destes;
- b) **Área de trabalho** - Os modelos que são criados precisam necessariamente ser visualizados com o objetivo de atender dois requisitos básicos quando se usa modelos: comprehensibilidade e a comunicação. Diante dessa necessidade, a área de trabalho permite visualizar e editar os modelos de forma interativa;
- c) **Paleta** – As entidades e relacionamentos disponíveis ao diagrama que está sendo criado aparecem na paleta do respectivo diagrama. Na Figura 3, aparecem os elementos do diagrama de classes, pois é o diagrama que estava sendo usado na ferramenta naquele momento.
- d) **Menus e Atalhos** – Nesta área aparecem os menus da ferramenta e os principais atalhos.

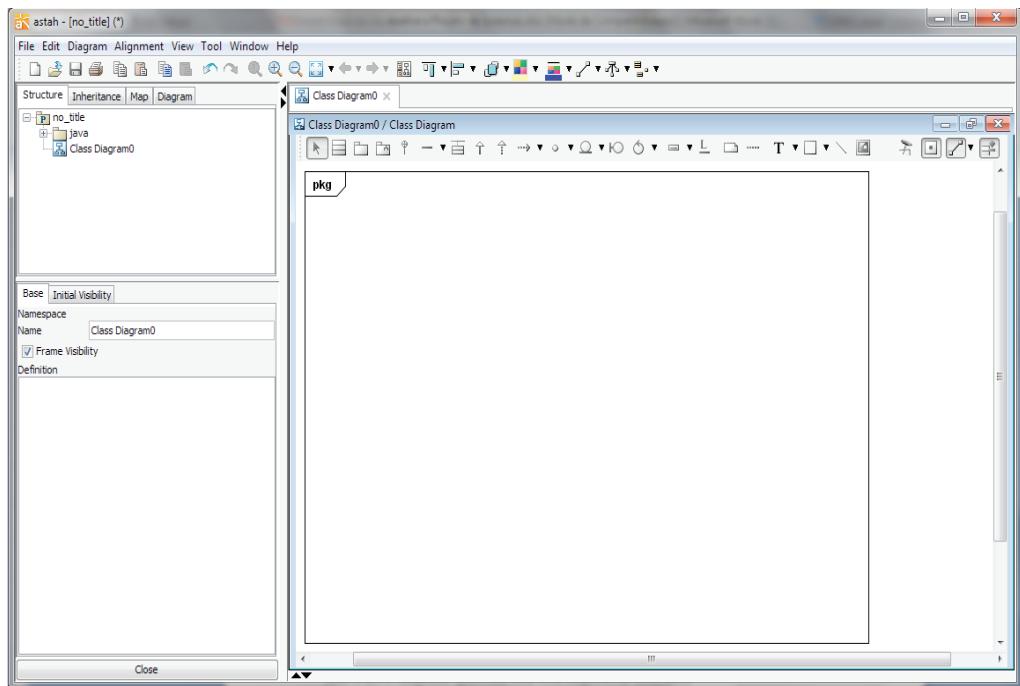
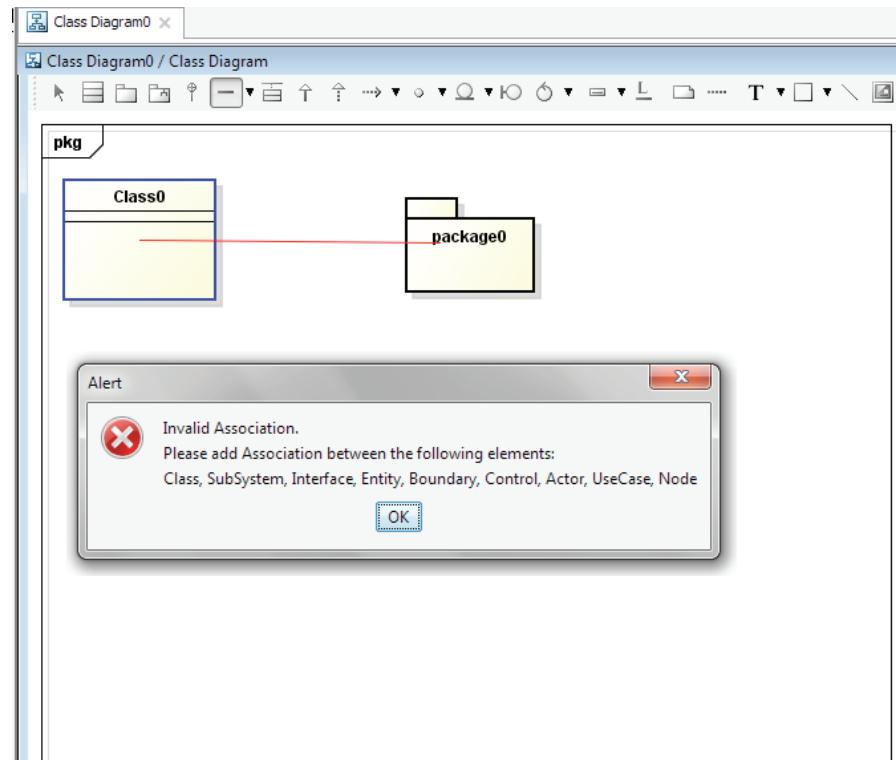


Figura 3 – Visão geral da ferramenta astah community e seus principais elementos.

Um recurso interessante é que a ferramenta garante a boa formação dos diagramas, ou seja, se o projetista tentar criar um relacionamento que não é permitido, a ferramenta emite uma mensagem. Além disto, o fato de cada diagrama disponibilizar na paleta apenas elementos permitidos pelo diagrama é outra maneira de garantir a boa formação.

Na Figura 4 podemos visualizar uma situação que a ferramenta exibe uma mensagem de erro e sua causa. Neste caso o relacionamento de associação estava sendo criado em um diagrama de classes entre uma classe e um pacote, o que pela regra de UML não é permitido.



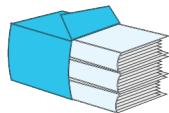
**Figura 4 – Validação de diagramas em astah community.**

Como prática, instale o astah community em seu computador e em seguida crie um diagrama dos seguintes tipos: Classes, Caso de uso, Máquina de estados, Atividades e Sequência. Ao criar os diagramas, adicione os elementos disponíveis na paleta e observe a representação gráfica de cada um deles. Note também a diferença entre os elementos da paleta de um diagrama para outro.



1. Em qual/quais dos diagramas da ferramenta os seguintes elementos aparecem:
  - a) Classe (Class);
  - b) Generalização (Generalization);
  - c) Linha da Vida (Life Line);
  - d) Anotação (Note);
  - e) Mensagem (Message);
  - f) Caso de Uso (Use Case);
  - g) Ponto de Inclusão (Include);
  - h) Pacote (Package)
  - i) Componente (Component)

2. Utilize outras ferramentas de modelagem e comente com seus colegas como ocorreu a experiência. Quais as vantagens da mesma em relação à astah community? Quais as desvantagens?
3. Por que uma ferramenta de modelagem é importante para as fases de análise e projeto?



## SÍNTESE DO CAPÍTULO

Neste capítulo foi apresentado um estudo introdutório sobre os fundamentos de modelagem de sistemas, UML e ferramentas de modelagem. Esta teoria é de fundamental importância, pois será utilizada como base para capítulos posteriores.



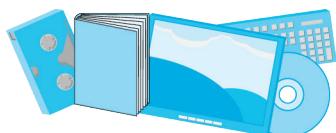
## REFERÊNCIAS

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. Elsevier.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: Guia do Usuário**. 2<sup>a</sup> Ed, Editora Campus.

FERRAMENTA ASTAH, material disponível em <http://astah.net/>

SOMMERVILLE, I. **Engenharia de Software**, 8<sup>o</sup> edição. São Paulo: Pearson addison Wesley.



## LEITURAS, FILMES E SITES

### Sites

**Object Management Group** - <http://www.omg.org>

**UML** - <http://www.uml.org/>



# Capítulo

# 2

## Elicitação e especificação de requisitos, análise e projeto de sistemas orientados à objeto

### Objetivos:

- As fases de Elicitação e Especificação de Requisitos, Análise e Projeto têm uma importância enorme no contexto de um processo de desenvolvimento de software devido ao fato de identificarem o que deve ser desenvolvido (requisitos e análise) e estabelecer como desenvolver (projeto). Portanto, os conceitos de Elicitação e Especificação de requisitos, Análise e Projeto de sistemas são apresentados nesta unidade, de modo a fornecer uma visão geral destas atividades no contexto do desenvolvimento de software.





## 1. Elicitação e especificação de requisitos

Para que um projeto de desenvolvimento de software seja considerado de sucesso, uma das premissas é que o produto gerado atenda o que o cliente deseja. Na grande maioria dos casos o cliente não sabe ao certo o que deseja e por este motivo a descrição das funcionalidades esperadas por parte do cliente pode mudar no decorrer do projeto. Portanto, há a necessidade de documentação das necessidades do cliente.

Quando estas descrições não são escritas ou são escritas e não são validadas com o cliente, é comum que no momento da entrega o cliente expresse que o produto não é o que ele havia solicitado anteriormente. Este é um dos grandes erros que os desenvolvedores individuais ou micro empresas de desenvolvimento de software que não se preocupam com requisitos estão sujeitas.

Veja a seguinte imagem (Figura 5) que explicita de forma lúdica o que a ausência do uso das técnicas de Engenharia de software causa nas fases iniciais do desenvolvimento de software.

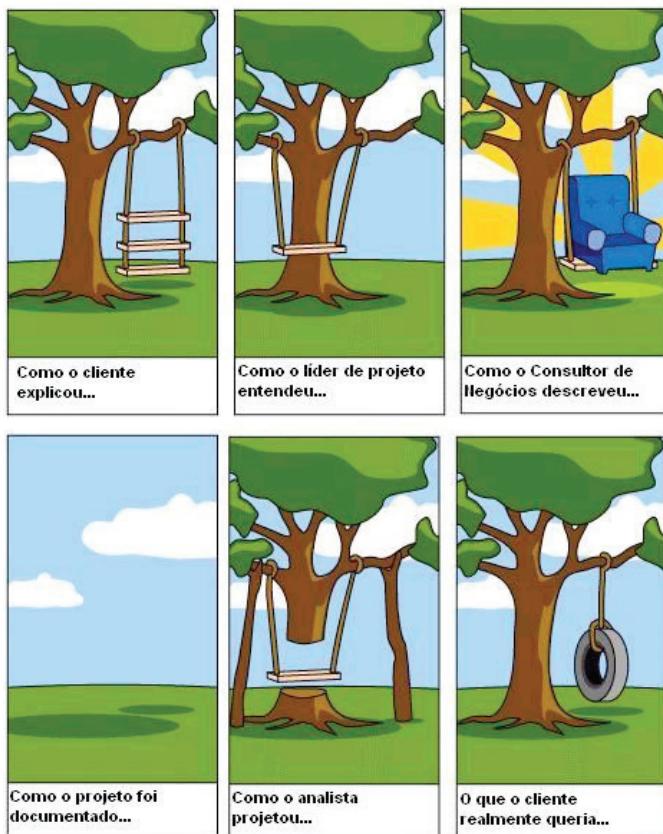


Figura 5 – Fases Iniciais do Desenvolvimento sem Técnicas de Engenharia de Software.

# ! ATENÇÃO

Rastreabilidade é o termo utilizado para expressar um mapeamento que é feito entre os artefatos que se relacionam. Geralmente este mapeamento é feito através de planilhas e aos pares (Requisitos Funcionais x Casos de Uso).



## SAIBA MAIS

Especificação de caso de uso é uma forma de descrever o sistema de modo detalhado. Histórias do usuário possuem formato semelhante aos casos de uso, porém estas estão relacionadas à metodologias ágeis. Ainda neste capítulo serão demostradas Especificações de Caso de uso

Como comprovar que o produto desenvolvido está em conformidade com o que foi solicitado?

A solução para esta situação é utilizar descrições do sistema, onde as necessidades são documentadas e validadas pelo cliente. Deste modo o escopo do software é delimitado, alinhado e acordado entre o cliente e fornecedor (equipe de desenvolvimento) do software.

Requisitos é o nome dado a este conjunto de documentos que descrevem os serviços a serem oferecidos pelo sistema e restrições operacionais de modo a satisfazer as necessidades do cliente.

Outro fator que demonstra a importância de requisitos é a necessidade de manter **rastreabilidade** entre os artefatos, tendo em vista que os demais artefatos são evoluções do documento de requisitos no decorrer do projeto. Em linhas gerais, o desenvolvimento dos artefatos ocorre da seguinte forma no decorrer do projeto:

1. O documento de requisitos é criado a partir do levantamento realizado;
2. Especificações de **casos de uso** são geradas a partir do documento de requisitos;
3. Diagramas da UML são criados para cada especificação de caso de uso;
4. O código é desenvolvido a partir dos diagramas gerados e da especificação de caso de uso (ou história do usuário);
5. As planilhas de teste são criadas tendo como base a especificação de caso de uso (ou história do usuário);
6. Quando o cliente vai realizar a validação do software desenvolvido, os requisitos que foram aceitos por ele são utilizados como parâmetro para validar o sistema;
7. Finalmente, quando o software será evoluído, o documento de requisitos é a raiz das mudanças a serem implementadas.

A seguir serão detalhadas as principais técnicas e tecnologias envolvidas na Elicitação (= Levantamento, compreensão) e na Especificação (= Documentação, escrita) de Requisitos.

## Elicitação de requisitos

Segundo Sommerville (2007), nesta atividade os profissionais de informática trabalham diretamente com os clientes e usuários finais do sistema para aprender sobre o domínio da aplicação, quais funcionalidades o sistema deve fornecer, o desempenho esperado e restrições de infraestrutura (hardware, rede...).

Esta atividade pode envolver profissionais com vários perfis dentro da organização para que se tenha uma visão abrangente das necessidades de cada perfil. Em um sistema a ser desenvolvido para a área de telemarketing, por exemplo, os gerentes, atendentes, diretores, setor de recursos humanos, porteiros e profissionais do setor de Tecnologia da Informação da empresa podem estar envolvidos. Estes perfis podem variar de acordo com a aplicação a ser desenvolvida, em um sistema de transporte um motorista pode ser um envolvido. Dada esta diversidade, os envolvidos são denominados através do termo técnico *Stakeholder*.

Além dos *Stakeholders*, outras fontes de requisitos são: Ambiente físico, Documentação (Formulários de cadastro de alunos, por exemplo), Dados existentes, Recursos existentes, Sistemas legados e Segurança.

Extrair os requisitos do domínio do problema nem sempre é uma tarefa trivial. Vários aspectos podem contribuir para isto, dentre eles podemos elencar a disponibilidade dos clientes e clareza do que é realmente necessário ao negócio.

Deste modo é necessário que seja utilizado um conjunto de técnicas de elicitação, a depender das dificuldades enfrentadas. A seguir, algumas técnicas de levantamento de requisitos e suas descrições:

- **Entrevista:** A equipe de análise de sistemas reúne-se com o(s) stakeholder(s) para uma conversa sobre as necessidades e expectativas em relação ao sistema a ser desenvolvido;



Entrevista entre um analista e um stakeholder.

- **Observação:** Como o próprio termo sugere, esta técnica consiste na observação do ambiente onde a aplicação irá funcionar a fim de captar informações acerca do funcionamento dos processos da empresa;
- **Demonstração de Tarefa:** Para algumas tarefas específicas a observação isolada pode não ser suficiente. Muitas vezes é necessário que uma determinada tarefa seja mostrada detalhadamente e repetidas vezes. Diante deste cenário, a técnica de demonstração de tarefa é utilizada, os stakeholders fazem a demonstração das tarefas aos analistas de sistemas;



Demonstração de Tarefa.

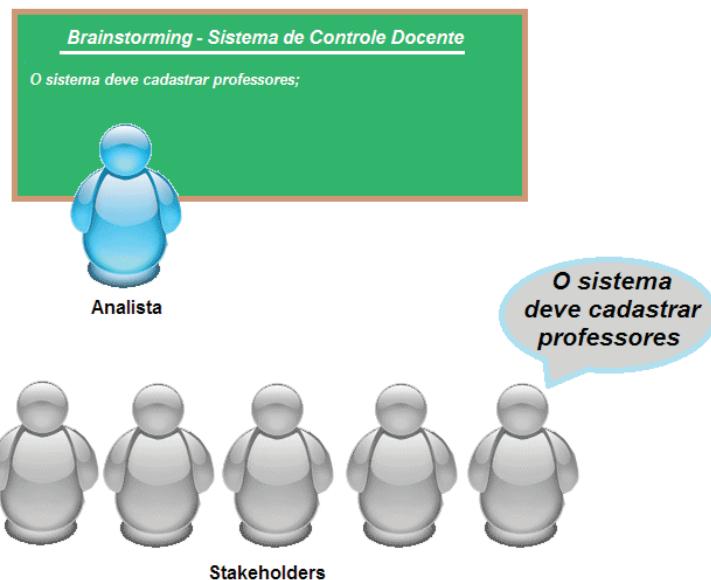
- **Estudo de Documentos:** Documentos em papel, como formulários de cadastro e relatórios já utilizados auxiliam bastante a atividade de elicitação de requisitos, uma vez que estes podem revelar um conjunto de dados esperado para uma funcionalidade;
- **Substituir o Usuário (Role playing):** Quando o domínio da aplicação é cheio de tarefas específicas e complexas, somente substituindo o funcionário que as realiza é que o analista de sistemas conse-

gue ter uma visão abrangente. Ao substituir o usuário, o analista repete os passos que o mesmo faria, sendo auxiliado pelo usuário em relação às dúvidas que possam surgir.



**Analista substituindo o Usuário.**

- **Questionário:** Questionário consiste em redigir um conjunto de perguntas que devem ser respondidas pelos usuários. Os questionários podem ser feitos para tirar dúvidas existentes em relação aos requisitos já elicitados ou na descoberta de requisitos novos;
- **Brainstorming (ou tempestade de idéias):** Para o desenvolvimento desta técnica é necessário que diferentes perfis de usuário participem. A execução desta técnica segue os seguintes passos: 1) um tempo máximo de reunião é estabelecido e os participantes têm um momento para refletir sobre o tema abordado; 2) Em seguida, os participantes vão ditando as idéias e todas elas são escritas de modo que todos possam ver; 3) Encerrada a sessão, as repetições de idéias são retiradas e idéias que geraram dúvidas são melhor explicadas; 4) As idéias são agrupadas e combinadas; 5) As melhores idéias escolhidas coletivamente são analisadas, melhoradas e aproveitadas. Esta técnica é importante para alinhar a visão de todos os participantes em relação ao sistema que será desenvolvido. Na Figura 6 é ilustrada a realização de um *brainstorming*, com a presença do analista e dos stakeholders.



**Figura 6 – Ilustração da realização de um brainstorming.**

- **Prototipação:** Consiste na criação de um esboço inacabado do sistema. Geralmente o esboço demonstra a interface de uma determinada funcionalidade do sistema para que o usuário possa ter uma melhor visualização do que será implementado. Este esboço representa a implementação inacabada do sistema, muitas vezes somente a parte de interface gráfica com os formulários com botões e campos de texto; no entanto, sem acesso a banco de dados, tratamento de campos ou processamento de cálculos;
- **Workshops ou Oficinas de Requisitos:** Reúnem todos os envolvidos durante um período curto, mas intensivo e focado. Neste período várias técnicas mencionadas podem ser aplicadas sequencialmente.

Uma vez que as técnicas foram empregadas e a elicitação dos requisitos foi dada como encerrada, surge uma nova necessidade: Escrever aquilo que foi identificado para fique documentado e para que possa ser validado pelo cliente. Assim sendo a ESPECIFICAÇÃO DE REQUISITOS é utilizada com tal finalidade.

## Especificação de requisitos

Os requisitos podem ser classificados em funcionais, não funcionais e requisitos de domínio.

Segundo Sommerville (2007), os requisitos Funcionais de um sistema descrevem o que o sistema deve fazer. Esses requisitos dependem do tipo do software que está sendo desenvolvido, dos usuários a que o software se destina e da abordagem geral considerada pela organização ao redigir os requisitos. Um exemplo de requisitos deste tipo seria: “*O sistema deve realizar o cadastro de clientes*”

Já os requisitos não funcionais, como o próprio nome sugere, são aqueles que não estão diretamente relacionados às funções específicas fornecidas pelo sistema. Eles podem estar relacionados às propriedades emergentes, como confiabilidade, tempo de resposta e espaço de armazenamento. Um problema comum aos requisitos não funcionais é que eles podem ser difíceis de verificar. Um exemplo de requisitos deste tipo seria: “*O sistema deve estar preparado para a realização de até 300.000 transações simultâneas*”

Os requisitos de domínio, por sua vez, refletem necessidades relacionadas ao domínio da aplicação e incluem uma terminologia específica de domínio ou fazem referência a conceitos do domínio. Os analistas muitas vezes encontram dificuldade em compreender e de relacioná-los com outros requisitos do sistema. Requisitos de domínio podem estar relacionados com velocidade de execução, confiabilidade, interoperabilidade com outros sistemas e leis que estejam relacionadas ao domínio da aplicação. Um exemplo de requisito deste tipo seria: “*O IMC deverá ser calculado através do valor do peso (em kg) dividido altura<sup>2</sup> (em metros)*”.

Os requisitos são necessários para a criação de diversos artefatos do sistema como diagramas, código, protótipo de interface e planilhas de teste. Dada esta importância, os requisitos devem ser escritos de forma precisa e isto muitas vezes é difícil devido às ambigüidades que frequentemente ocorrem ao se escrever na língua portuguesa. Veja a seguinte frase: “O professor falou com o aluno parado na sala”, a ambigüidade neste caso, se dá em relação a quem estava parado: o professor ou o aluno. Assim como na frase, comumente a escrita de requisitos é permeada de ambigüidades, que



Um exemplo de definição contraditória seria em determinado momento do documento de requisitos aparecer a informação que “Inicialmente nenhum usuário deve vir pré-cadastrado no sistema”. Enquanto que em outro trecho deste documento aparece a informação que “O primeiro acesso ao sistema deve ser feito com o perfil de administrador, este deve vir pré-cadastrado”.

devem ser eliminadas para evitar interpretação errada e criação de artefatos de forma errada.

Além da ausência de ambigüidade, duas propriedades desejáveis aos requisitos são a **completeza** e **consistência**. Completeza significa que tudo que o usuário necessita que seja implementado esteja definido e consistência significa que os requisitos não devem ter **definições contraditórias**.

O uso de jargões computacionais também figura no rol de *vícios* frequentemente cometidos que devem ser evitados em requisitos. Um exemplo deste erro seria: “A consulta aos clientes deve ser feita através de um *transfer object* que deve ser transitado entre as camadas”. Esta informação não é do escopo de requisitos, mas deve aparecer em outro documento (O documento de arquitetura – Este não está no escopo deste livro).

A seguir temos alguns requisitos do sistema e em seguida a classificação de cada requisito que aparece neste documento como Funcional, Não Funcional ou de Domínio.



## Documento de Requisitos de Sistema da Fábrica de Tecidos

- 1)** Quanto à interface de Usuário, o sistema será desenvolvido para ambiente Windows. Nesse ambiente, a interface do usuário com o sistema se dará através de aplicações Windows. (*Requisito Não Funcional*)
- 2)** Em caso de três tentativas consecutivas de acesso e falha no mesmo, a senha deve ser bloqueada: (*Requisito Funcional*)
- 3)** A distribuição do Sistema deve ser realizada a partir de uma instalação, em formato conhecido pelo Windows, com capacidade de desinstalação. (*Requisito Não Funcional*)
- 4)** Hardware

### Cliente

Compatível com PC Pentium III ou superior

256 MB de memória RAM

HD com capacidade mínima de 40 GB

CD-ROM

Porta de comunicação serial

Porta de comunicação paralela

Porta de comunicação Ethernet

Porta de comunicação USB

### Servidor

Compatível com PC Pentium IV ou superior

Um GB de memória RAM

HD com capacidade mínima de 80 GB

## CD-ROM

- Porta de comunicação serial
  - Porta de comunicação Ethernet
  - Porta de comunicação USB
  - Nobreak
- (Requisito Não Funcional)

- 6)** O idioma padrão utilizado no sistema será o português. Todas as telas, menus e relatórios serão apresentados nesse idioma. (Requisito Não Funcional)
- 7)** Possibilidade da conversão da unidade de medida: recurso utilizado para obtenção do peso líquido e peso líquido corrigido através de um fator multiplicativo (decimal) e escolha da unidade a ser originada: Exemplo: O peso líquido registrado na balança é 2.190 kg. Pretendo obter o peso líquido em toneladas. Como proceder?

Fator de conversão = 0,001;

Unidade após a conversão = t (campo para preenchimento livre até cinco caracteres alfanuméricos);

à Cálculo: Peso líquido convertido =  $2.192 * 0,001 = 2,190$  t

Agora, para obtenção do peso líquido corrigido, o cálculo será:

à Peso líquido corrigido = Peso líquido \* 0,001 \* Fator de correção final  
(Requisito de Domínio)

- 8)** Alguns tipos de veículos já deverão vir pré-cadastrados desde a instalação, os mesmos se encontram descritos abaixo:

Leve ou Toco



Comprimento (m)

Total      Eixo      Direc. e Último

10            6

Bitrem



Comprimento (m)

Total      Eixo      Direc. e Último

19,8        17,6

Rodotrem



### Comprimento (m)

Total	Eixo	Direc. e Último
19,8 / 24,7 / 26,5		17,6 / 22,5 / 24,1

(Este requisito apresenta informações funcionais por descrever que estes tipos de veículos devem vir pré-cadastrados e fornece os valores que devem ser cadastrados. Por outro lado as informações destes valores relacionam-se ao domínio do problema. Portanto, este requisito tanto pode ser classificado como Requisito de Domínio como requisito Funcional).

**9)** O sistema realizará o cadastro dos pontos de controle através de formulário especializado no sistema. Um usuário poderá incluir, excluir, alterar e consultar os pontos de controle no sistema.

- Os atributos que devem ser implementados são: Código, Descrição, Amarração com o IP / Nome da máquina e Operações amarradas a este ponto de controle.

*(Requisito Funcional)*

**10)** O sistema realizará o cadastro das tabelas de fatores de correção através de formulário especializado no sistema. Um usuário poderá incluir, excluir, alterar e consultar as tabelas de fatores de correção no sistema.

*(Requisito Funcional)*

**11)** O sistema realizará o cadastro dos motoristas através de formulário especializado no sistema. Um usuário poderá incluir, excluir, alterar e consultar os motoristas no sistema.

- Os atributos que devem ser implementados são: Código do motorista, Nome, RG, CPF, Envolvido em Problemas e Observação.

*(Requisito Funcional)*

Inicialmente os requisitos são escritos em um mesmo documento como o documento anterior e depois de classificados são divididos. Sendo que os requisitos funcionais evoluem para especificações de caso de uso e os requisitos não funcionais são agrupados em outro documento.

Documentos de caso de uso são maneiras de especificar requisitos de forma mais completa, detalhada e mais aproximada à implementação do sistema. Este documento é dividido basicamente em duas seções: *Happy Day* – Que descreve as situações onde não ocorrerão erros, e os fluxos alternativos, que representam as situações de exceção.

Os documentos de casos de uso também podem apresentar pontos de inclusão e pontos de extensão. Para diferenciar se devemos utilizar Ponto de Inclusão ou Ponto de Extensão, é simples: Ponto de Inclusão = passagem obrigatória e Ponto de Extensão = passagem facultativa.

O ponto de inclusão é utilizado quando dois casos de uso relacionam-se e um deles inclui o outro de forma obrigatória em sua execução. Um exemplo de ponto de inclusão pode ser dado com os casos de uso Solicitar Pedido de Peças e outro caso de uso chamado Validar Cliente, quando um cliente deseja fazer o pedido de alguma peça, ele **deve** ter seus dados validados (obrigatoriamente).

O ponto de extensão é utilizado entre dois casos de uso quando um caso deseja inserir um comportamento opcional ou excepcional disparado por alguma condição. Um exemplo de ponto de extensão seria um caso de uso chamado Operação Venda e outro caso de uso chamado Emissão de

Nota Fiscal, onde uma Operação de Venda pode disparar a Emissão de Nota Fiscal (de maneira não obrigatória). Neste caso é usado o relacionamento de extensão.

Pontos de Extensão e Pontos de Inclusão não são obrigatórios em documentos de casos de uso. Pode haver casos de uso que não incluem nem estendem outros casos de uso, neste caso a seção de pontos de extensão e de inclusão recebe o valor N/A (Não se aplica).

A seguir, segue um exemplo de caso de uso que descreve a funcionalidade de manter Usuários do Sistema.



## Especificação de casos de uso

Manter Usuários do Sistema

Responsável: Enyo José

### 1. HISTÓRICO

Data	Versão	Responsável	Alteração
25/02/2012	A.01	Enyo José	Criação do documento.

## MANTER USUÁRIOS DO SISTEMA

### 1. Descrição

Este caso de uso se responsabiliza pela inclusão, exclusão e pesquisa entre os registros de Usuário do Sistema.

### 2. Fluxo Básico

Passo 1. Este caso de uso inicia quando o usuário abre a tela do sistema;

Passo 2. O sistema exibe as opções de inclusão, exclusão e pesquisa;

Passo 3. O sistema aguarda que o usuário especifique a operação que deseja realizar.

Passo 4. Uma vez que o usuário solicite executar uma das funções desejadas (inserir, excluir, ou pesquisar), um dos seguintes sub-fluxos é executado:

Se o usuário solicitar “Inserir”, o sub-fluxo 2.1 Incluir Usuário do Sistema é executado;

Se o usuário solicitar “Excluir”, o sub-fluxo 2.2 Excluir Usuário do Sistema é executado;

Se o usuário solicitar “Pesquisar”, o sub-fluxo 2.3 Pesquisar Usuário do sistema é executado.

#### 2.1. Inserir Usuário do Sistema

Passo 1. Este sub-fluxo inicia quando o usuário solicita inserir um novo Usuário do Sistema;

Passo 2. O sistema solicita ao usuário o preenchimento dos campos a seguir:  
Nome\*: Representa o nome do Usuário do Sistema;• CPF\*: Representa o número do CPF do Usuário do Sistema;  
RG: Representa o número do RG do Usuário do Sistema;  
Endereço: Representa o endereço do Usuário do Sistema;  
Data de nascimento: Representa a data de nascimento do Usuário do Sistema;  
Informação Geral do Usuário: Representa comentários;  
Passo 3. O usuário preenche os campos e solicita salvar o cadastro do Usuário do Sistema;  
Passo 4. O sistema realiza a inclusão dos dados informados pelo usuário;  
Passo 5. O caso de uso retorna para o passo 2 do Fluxo Básico.

## 2.2. Excluir Usuário do Sistema

Passo 1. Este fluxo inicia quando o usuário solicita excluir um Usuário do Sistema;  
Passo 2. O sistema solicita o CPF do usuário;  
Passo 3. O usuário preenche o campo e solicita a exclusão;  
Passo 4. O Usuário do Sistema é removido do sistema;  
Passo 5. O caso de uso retorna para o passo 2 do Fluxo Básico.

## 2.3. Pesquisar Usuário do Sistema

Passo 1. Este fluxo inicia quando o usuário deseja pesquisar um usuário do sistema;  
Passo 2. O sistema solicita o preenchimento do campo CPF;  
Passo 3. O usuário preenche o campo e solicita a consulta;  
Passo 4. O sistema busca o Usuário do Sistema apresentando todos os campos citados no passo 2 do fluxo 2.1 Incluir Usuário do Sistema;  
Passo 5. Se o usuário solicitar “Editar”, o sub-fluxo 2.4 Editar Usuário do sistema é executado. Caso contrário, o caso de uso retorna para o passo 2 do Fluxo Básico.

## 2.4. Editar Usuário do Sistema

Passo 1. Este sub-fluxo inicia quando o usuário solicita editar um Usuário do Sistema;  
Passo 2. O sistema exibe os seguintes campos do Usuário do Sistema selecionado e permite ao usuário alterá-los:  
Nome\*: Representa o nome do Usuário do Sistema;• CPF\*: Representa o número do CPF do Usuário do Sistema;  
RG: Representa o número do RG do Usuário do Sistema;  
Endereço: Representa o endereço do Usuário do Sistema;  
Data de nascimento: Representa a data de nascimento do Usuário do Sistema;  
Informação Geral do Usuário: Representa comentários;  
Passo 3. O usuário Altera os campos e solicita salvar o cadastro do Usuário do Sistema;

Passo 4. O sistema realiza a alteração dos dados informados pelo usuário;

Passo 5. O caso de uso retorna para o passo 2 do Fluxo Básico.

Fluxos Alternativos (Exceptions)

### 3.1. Cancelar Inclusão do Usuário do Sistema

Se no passo 2 do sub-fluxo 2.1 Incluir Usuário do Sistema, o usuário solicitar cancelar a inserção do Usuário do Sistema, o caso de uso retorna para o passo 2 do Fluxo Básico.

### 3.2. Cancelar Exclusão do Usuário do Sistema

Se no passo 2 do sub-fluxo 2.2 Excluir Usuário do Sistema o usuário solicitar cancelar a exclusão do Usuário do Sistema, o caso de uso retorna para o passo 2 do Fluxo Básico.

### 3.3. Cancelar Pesquisa do Usuário do Sistema

Se no passo 2 do sub-fluxo 2.3 Pesquisar Usuário do Sistema, o usuário solicitar cancelar a Pesquisar Usuário do Sistema, o caso de uso retorna para o passo 2 do Fluxo Básico.

### 3.4. Cancelar Edição do Usuário do Sistema

Se no passo 2 do sub-fluxo 2.4 Editar Usuário do Sistema, o usuário solicitar cancelar a Edição de Usuário do Sistema, o caso de uso retorna para o passo 2 do Fluxo Básico.

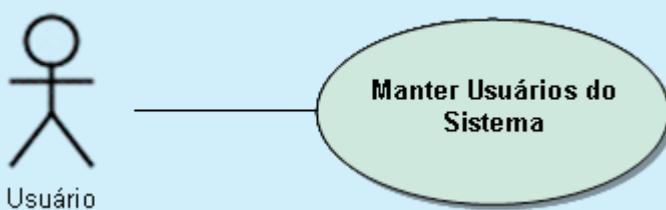
### 3.5. Pontos de Inclusão

-N/A

### 3.6. Pontos de Extensão

-N/A

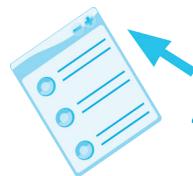
### 3.7. Diagrama do Caso de Uso



Observe que o documento de caso de uso descreve a interação entre usuário do sistema e o sistema em funcionamento, informando os campos que devem aparecer na tela e as possibilidades de interação. Para entender melhor o documento acima, desenhe os caminhos possíveis do caso de uso em um papel como uma espécie de árvore tendo o fluxo básico como nó raiz.

A figura que consta no documento de caso de uso apresentado é um diagrama de Casos de Uso de UML. Este diagrama será abordado neste livro e os conceitos de casos de uso serão abordados novamente, do ponto de vista de modelagem.

Na fase de elicição e especificação de requisitos, é comum que diagramas de casos de uso sejam criados para auxiliar no entendimento do sistema. Outros diagramas que podem ser utilizados com esta finalidade são os diagramas de atividades e diagramas de sequência.



## ATIVIDADES DE AVALIAÇÃO

- 1.** Qual a importância de utilizar requisitos em um projeto de desenvolvimento de Software?
- 2.** Diferencie Elicitação de Especificação de requisitos.
- 3.** Quais as principais técnicas de Elicitação de Requisitos? Descreva-as.
- 4.** Cite 5 Requisitos Funcionais para:
  - a) Sistema da padaria de pequeno porte;
  - b) Sistema de alocação docente.
- 5.** Cite 5 Requisitos Não Funcionais para:
  - a) Sistema da padaria de pequeno porte;
  - b) Sistema de alocação docente.
- 6.** Escreva um caso de uso chamado *Manter Professor* para o sistema de alocação docente.



## 2. Análise

O resultado do levantamento de requisitos são documentos que contém as descrições do que deve ser implementado descritas em alto nível de modo a permitir o entendimento e aceitação entre os futuros usuários e os analistas. Na fase de análise, as especificações de requisitos são estudadas para que seja feito o detalhamento e refinamento destas através de modelos, realizando assim uma aproximação em direção à solução final. Observe que o desenvolvimento de um processo dar-se através destas aproximações rumo à solução definitiva que possa ser utilizada pelo cliente.

O grande objetivo desta fase é a obtenção de uma melhor compreensão dos requisitos, mantendo uma descrição dos requisitos que seja compreensível e auxilie no posterior *design* (projeto) do sistema.

Um dos desafios de projetar em orientação a objetos é representar o sistema a partir de uma perspectiva global (Objetivo da Análise). Podemos dizer que o que se faz na fase de análise é uma espécie de tradução das especificações dos requisitos, que se encontram na linguagem do cliente, para uma representação que use uma linguagem do desenvolvedor. Assim, passamos de um modelo de casos de uso para um modelo de análise. O modelo de casos de uso corresponde a uma visão do sistema sob a óptica do cliente, ou seja, para o que o cliente deseja que seja implementado. O modelo de análise corresponde a uma visão do mesmo sistema já sob um ângulo diferente, ou seja, a descoberta e detalhamento do que o usuário necessita e sua descrição, preparando o terreno para a fase de *design* que virá no futuro.

No entanto, os modelos criados neste momento não são detalhados o suficiente ao ponto de serem codificados diretamente. Estes necessitam de um maior detalhamento para que a implementação aconteça da melhor maneira, no que diz respeito aos algoritmos utilizados, relacionamentos entre os elementos identificados, dentre outros aspectos.

As fases de análise e projeto diferenciam-se no seguinte sentido:

- A fase de **análise** busca responder as seguintes perguntas do ponto de vista da equipe de desenvolvimento (Não do ponto de vista do cliente, como os requisitos): o quê? O que o sistema deve fazer... o que o sistema deve ser... o que o sistema realiza.
- A fase de **projeto**, ao contrário, buscará responder a seguinte pergunta: como? Como o sistema fará... como o sistema será... como o sistema realiza.

Durante a fase de análise, os principais artefatos de software desenvolvidos são o diagrama de Casos de uso e da Arquitetura de Análise, decomposta em diversos diagramas de classes (O diagrama de Classes será visto no Capítulo 2 da Unidade 3) que devem conter as **classes e pacotes** de análise.



**SAIBA  
MAIS**

Classes e Pacotes tem o mesmo significado que foi visto na disciplina Linguagem de Programação II, com java. Porém neste momento, não é representada através de código, mas de diagramas.

Segundo Bezerra (2007), o modelo de casos de uso é uma representação das funcionalidades externamente observáveis do sistema e como os elementos externos ao sistema interagem com ele. Este tem sua importância, pois direciona outras tarefas posteriores do processo de desenvolvimento.

A arquitetura do sistema especifica a organização e disposição em camadas das classes, componentes, e demais elementos do sistema. Muitas vezes a arquitetura dos sistemas baseiam-se no padrão arquitetural Model-View-Controller (MVC), que prega a separação do código em três camadas:

- **View** (Visão): Nesta camada é mantido o código relacionado à visualização gráfica dos elementos;
- **Controller** (Controlador): Esta camada mapeia as ações requisitadas pela visão (view) ao Modelo (Model) ou vice-verso;
- **Model** (Modelo): Esta camada é responsável por acessar o banco de dados (ou outras formas de persistência) e pela lógica de negócio.

Classes de análise representam abstrações de uma ou mais classes ou subsistemas que irão aparecer no projeto do sistema. As classes de análise representam a solução em mais alto nível por meio da qual estaremos a descrever a funcionalidade do sistema que estamos desenvolvendo. Dentre outras características, as classes de análise focalizam nos requisitos funcionais do sistema, postergando os requisitos não-funcionais para a fase de *design*. Assim, deve-se preocupar com a descoberta de atributos de alto nível e relacionamentos conceituais.

Várias propostas têm sido feitas sobre como identificar as diferentes entidades do sistema. Estas podem ser utilizadas tanto na fase de análise quanto na fase de projeto. A seguir algumas delas são listadas.

- **Análise gramatical** do documento de requisitos é realizada de modo que as classes e atributos são representados por substantivos nos documentos de requisitos e os verbos representam as operações (Métodos). Exemplo: “*O usuário pode salvar e editar os dados do Funcionário*” neste exemplo o substantivo Funcionário representa uma classe do sistema enquanto salvar e editar representam operações a serem implementadas;
- **Entidades do domínio** – Consiste em levar em consideração entidades do domínio que tem significado para o sistema como tipos de caminhão em um sistema de transporte rodoviário.
- **Análise baseada em cenários** – Neste caso, cenários de uso do sistema são identificados e analisados um de cada vez. Um método chamado de cartões CRC (Classes-Responsabilidades-Colaboradores) é eficaz no apoio a essa abordagem baseada em cenários. Exemplo de um cartão CRC é dado a seguir no contexto de um sistema de alocação docente, para a entidade Disciplina.

Nome: <i>Disciplina</i>	
Responsabilidades	Colaboradores
1. Conhecer seus pré-requisitos 2. Conhecer seu código 3. Conhecer seu nome 4. Conhecer a quantidade de créditos	<i>Disciplina</i>

Além do diagrama de casos de uso e do diagrama de classes de análise, outros diagramas como o diagrama de interação, diagrama de estados

e diagrama de atividades podem ser utilizados na análise. O foco do uso destes diagramas é o melhor entendimento (Visão macro) do sistema que está sendo desenvolvido.



## ATIVIDADES DE AVALIAÇÃO

- 1.** Faça uma pesquisa mais aprofundada sobre o padrão arquitetural MVC, observe seus detalhes e formas de adaptação.
- 2.** Cite três diferenças entre a fase de análise e projeto.
- 3.** Qual a importância da fase de análise para um projeto de desenvolvimento de software?
- 4.** Encontre modelos criados para os seguintes diagramas: casos de uso, classes, interação, estados e atividades. Observe os elementos de cada um, as diferenças e semelhanças entre eles.



### 3. Projeto

A passagem da fase de análise para a fase de projeto é quase imperceptível no desenvolvimento de sistemas orientados a objeto, tendo em vista que a mesma notação é utilizada para expressar os artefatos das duas fases. Tanto, que em projetos de sistemas muito pequenos, as duas fases geralmente se unem em uma só.

A fase de projeto realiza um refinamento dos artefatos criados durante a análise, especificando de maneira pormenorizada os modelos gerados e concluindo o projeto de arquitetura iniciado na fase anterior.

Adicionalmente outros diagramas são criados levando em consideração os documentos de requisitos e os diagramas já criados na análise, persistência dos dados é definida e o projeto de interface gráfica é criado.

A seguir, algumas definições das atividades características da fase de projeto segundo Bezerra (2007):

#### Refinamento dos Aspectos Estáticos e Estruturais

As classes de análise não apresentam detalhes de implementação e representam o domínio do problema em alto nível. O detalhamento do modelo de classes de análise é evoluído na fase de projeto, sendo que pode ser constatado que uma classe de análise deve ser desmembrada em várias classes de projeto, por exemplo. Além disto, vários detalhes são definidos para que a implementação possa dar-se a partir destes modelos.

O detalhamento acontece em relação à definição de atributos e métodos, além dos relacionamentos entre as classes como herança por exemplo. Polimorfismo, Classes abstratas e interfaces também são identificados e detalhados no diagrama de classes de projeto.

#### Projeto dos Algoritmos

O projeto de algoritmos é outra atividade da fase de projeto. Deste modo o projetista pode especificar qual algoritmo deve ser utilizado em determinado elemento do diagrama. Os algoritmos a serem utilizados podem ser representados através de diagramas de atividades da UML, por exemplo. A escolha de um bom algoritmo geralmente tem impacto no tempo de resposta da aplicação e na qualidade e precisão desta resposta.

## Detalhamento dos Aspectos Dinâmicos

Os aspectos dinâmicos estão relacionados à forma como a aplicação deve se comportar, ou seja, preocupa-se em expressar o fluxo de execução de uma determinada funcionalidade, de um caso de uso ou do sistema inteiro. Embora o estudo de aspectos dinâmicos inicie na fase de análise, é na fase de projeto que o detalhamento ocorre através dos diagramas de interações, sequência, de estados ou de atividades de UML.

Dúvidas em relação aos elementos de modelagem estabelecidos na Análise costumam aparecer em projeto ou novas entidades podem ser extraídas quando se tem uma melhor compreensão do sistema. Deste modo as técnicas de identificação das diferentes entidades do sistema (Análise gramatical, Entidades do domínio, Análise baseada em cenários, etc...) que já foram utilizadas na análise podem ser novamente utilizadas neste momento.

## Inclusão de Padrões de Projeto

Um padrão de projeto descreve uma situação problema recorrente no desenvolvimento, juntamente com a solução genérica adequada, exemplos de uso e outros padrões relacionados. O primeiro catálogo de padrões de projeto orientados a objeto foi proposto por Gamma et al. no livro *Design Patterns – Elements of Reusable Object-Oriented software*. Como o livro possui quatro autores, tanto o livro quanto os padrões ficaram conhecidos de *Gang of Four* – GOF (Gangue dos Quatro). Existem padrões relacionados às diversas tecnologias/ambientes. Padrões J2EE são padrões voltados para desenvolvimento web, existem padrões arquiteturais...

Na fase de projeto, os padrões costumam ser utilizados para modelar as soluções da melhor maneira possível. Mais detalhes de padrões de projeto e exemplos de uso são fornecidos no Capítulo 2 da Unidade 5 deste livro.

## Inclusão de Frameworks

Segundo Sommerville os frameworks são constituídos de um conjunto de classes concretas, classes abstratas e interfaces de modo a facilitar o desenvolvimento de um determinado foco específico dentro do projeto. Inúmeros frameworks têm sido propostos, frameworks capazes de controlar a parte de visão (interação dos campos visíveis pelo usuário com o restante da aplicação) são bastante utilizados.

Frameworks de persistência de dados, que utilizam o artifício de mapeamento objeto-relacional também são frequentemente utilizados. Eles são importantes, pois atualmente as linguagens mais utilizadas para o desenvolvimento de sistemas são as linguagens orientadas a objeto, onde os dados são salvos em objetos e, por outro lado, os dados destes objetos são persistidos em bancos de dados que, em quase na totalidade dos projetos de software, são relacionais. Estes frameworks fazem este mapeamento entre os atributos dos objetos e os campos do banco.

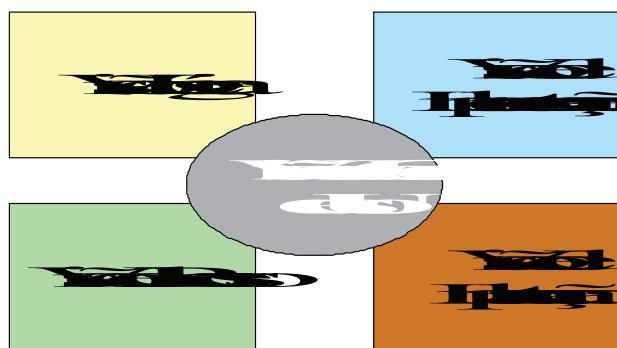
Durante a fase de projeto, é necessário estabelecer os frameworks que serão utilizados. Esta escolha tem um impacto na definição da arquitetura, e vice-verso, além de impactar no refinamento dos aspectos estáticos.

## Projeto da Arquitetura

Na fase de projeto a arquitetura é realmente especificada. O projeto arquitetural é a atividade do processo de desenvolvimento no qual os subsistemas, camadas do software e dependências entre subsistemas são definidos. A escolha de componentes reutilizáveis disponíveis também faz parte do projeto arquitetural.

A arquitetura do sistema, conforme definida por Rumbaugh diz respeito à estrutura organizacional de um sistema, incluindo a sua decomposição em partes, a inter-relação entre essas partes, além dos mecanismos e princípios que devem guiar o projeto de um sistema.

Desta forma, podemos definir a arquitetura como a captura dos aspectos estratégicos da estrutura em alto nível de um sistema. A arquitetura, conforme o padrão de modelagem descrito pela UML, pode ser analisada através de 5 divisões (4 + 1) de escopo chamadas de visões, conforme mostrado na Figura 1.



A motivação principal para se usar diferentes visões para a arquitetura é a de reduzir a complexidade como um todo. Cada uma destas visões representa um aspecto específico da arquitetura e será tratada individualmente nas seções seguintes deste documento. Conforme pode ser observado, a UML centraliza seu enfoque no tratamento de casos de uso, visão que permeia as outras visões. Seguindo o mesmo princípio, a arquitetura proposta neste documento terá como base casos de uso das interações entre os componentes do sistema e depois abordará as outras visões mostrando as intersecções.

O resultado é o documento de arquitetura que define cada decisão do projeto arquitetural e a estrutura básica do projeto, criada de acordo com a especificação da arquitetura.

## Persistência dos Dados

Geralmente os sistemas desenvolvidos persistem os dados recebidos de alguma forma. A atividade de persistência dos dados tem por finalidade transformar o Modelo de Domínio nas estruturas de dados necessárias para implementar o software (Modelo de Dados), garantindo que os dados persistentes sejam armazenados com consistência e eficiência.

Aspectos relacionados à persistência devem ser tratados na fase de projeto como a forma em que os dados serão persistidos (XML, banco de dados, arquivos, planilhas...) e a forma em que as transações são controladas. No caso de persistência através de banco, é importante o estabelecimento das tabelas e suas respectivas informações, relacionamentos entre tabelas,

identificação de chaves primárias, chaves estrangeiras e qualquer comportamento definido no banco de dados, como procedimentos armazenados, triggers, restrições, etc.

## Projeto de Interface Gráfica com o Usuário

Quando um sistema envolve o uso de uma interface gráfica com o usuário é necessário que a interface seja projetada para que haja uma identidade visual aos usuários. Questões como padronização de cores, padronização de mensagens de erro, logotipo que deve aparecer, posicionamento dos campos, etc.

São princípios de projeto de interface com o usuário:

Princípio	Descrição
Familiaridade com o usuário	A interface deve utilizar termos e conceitos que tenham como base a experiência das pessoas que mais vão utilizar o sistema.
Consistência	A interface deve ser consistente no sentido de que operações semelhantes devem ser ativadas da mesma maneira, sempre que possível.
Mínimo de Surpresa	Ações semelhantes devem ter efeitos equivalentes.
Facilidade de Recuperação	A interface deve incluir mecanismos para permitir aos usuários recuperação a partir de erros.
Orientação para o usuário	A interface deve fornecer respostas significativas, quando ocorrem erros, e oferecer recursos sensíveis ao contexto de ajuda ao usuário.
Diversidade de Usuário	A interface deve fornecer recursos de interação apropriados a diferentes tipos de usuários de sistema.

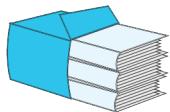
Existe todo um estudo relacionado ao projeto de interface com o usuário. Este tema de tão abrangente muitas vezes é foco de uma disciplina inteira nas universidades e de livros. Aqui o tema foi descrito de maneira bastante sucinta, de modo a fornecer entendimento geral desta atividade de projeto.

Como vimos, os diagramas de classes e caso de uso são utilizados nesta fase, no entanto, qualquer um dos demais diagramas da UML pode ser utilizado quando necessário.



## ATIVIDADES DE AVALIAÇÃO

1. Qual a diferença entre a modelagem das classes de análise e de projeto?
2. Qual o objetivo da fase de projeto?
3. Elabore um cartão CRC para a entidade Aluno do sistema de alocação docente.



## SÍNTSE DO CAPÍTULO

Neste capítulo, as fases de Elicitação e Especificação de Requisitos, Análise e Projeto são conceituadas de modo a propiciar ao leitor uma diferenciação das atividades desempenhadas e objetivo de cada uma destas fases. Os diagramas de UML que geralmente são utilizados são elencados no decorrer das fases.



## REFERÊNCIAS

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. Elsevier.

PRESSMAN, R. S. **Engenharia de Software**. 6 Ed. São Paulo: McGraw-Hill Brasil Técnicos.

SOMMERVILLE, I. **Engenharia de Software**, 8º edição. São Paulo: Pearson Addison Wesley.

# Capítulo

# 3

## Diagramas da UML - Parte 1

### Objetivos:

- Neste capítulo serão apresentados dois diagramas da UML: O diagrama de caso de uso e o diagrama de classes. O primeiro é importante para dar uma visão mais genérica do sistema e das permissões de acesso em relação aos perfis de usuário necessários. O segundo é um dos principais diagramas disponibilizados pela UML, este é capaz de explicitar detalhes de implementação que devem ser seguidos pelo implementador que codificará o sistema.





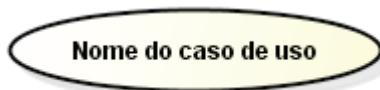
## 1. Diagrama de Casos de Uso

Este é um diagrama comportamental utilizado nas fases iniciais do projeto, portanto seu principal objetivo é explicitar de forma macro (sem detalhes) os requisitos funcionais de um sistema, partindo do ponto de vista do usuário.

Tem uma importância no que diz respeito a delimitar o escopo do sistema e definir as funcionalidades oferecidas ao usuário. Assim sendo, o diagrama concentra-se em compreender o problema a ser resolvido e não a detalhes de implementação neste momento.

Os elementos disponibilizados pelo diagrama de casos de uso são:

- **Casos de uso:** Expressa uma funcionalidade que o sistema deve conter. É representado através de uma elipse que contém o nome do caso de uso.

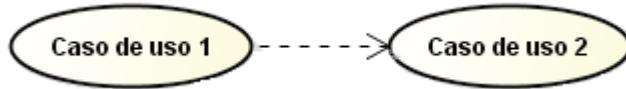


- **Atores:** Representam papéis (perfis) desempenhados por usuários ou qualquer entidade externa ao sistema. Administrador do Sistema, Professor, Aluno, tutor presencial e tutor à distância são exemplos de atores para um sistema de educação à distância para a Universidade Aberta do Brasil. O *moodle*, por ser um sistema já existente, poderia ter relação com alguma funcionalidade do novo sistema e, neste caso, ser considerado como um ator.
- A notação da UML para representar um ator é uma figura de palito com o nome do ator em baixo da figura.

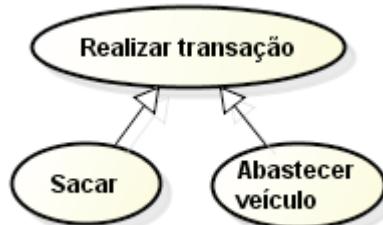


- **Relacionamentos:** As mesmas regras apresentadas no capítulo próximo (diagrama de classes) para os relacionamentos (Dependência, generalização e associação) devem ser consideradas para o diagrama de casos de uso:

- No diagrama de casos de uso, a **dependência** pode ser estabelecida entre casos de uso, atores ou entre um ator e um caso de uso. A seguir, segue a ilustração da representação da dependência entre casos de uso.



- **Generalização/Especialização:** As entidades devem ser do mesmo tipo, ou seja, dois casos de uso ou dois atores. No contexto de um sistema bancário, o diagrama a seguir é capaz de ilustrar uma situação de relacionamento de generalização entre o caso de uso realizar transação (genérico) e os casos de uso sacar e abastecer veículo (específicos).



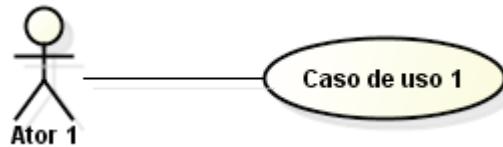
- **Associação:** É o relacionamento mais comum em diagramas de caso de uso. No diagrama de caso de uso, a associação pode ser utilizada para conectar dois atores, dois casos de uso ou um ator e um caso de uso. Neste diagrama, associações entre casos de uso e os usuários que tem permissão de acessá-los é o tipo de relacionamento mais utilizado. A seguir ilustrações do uso da associação entre as entidades do diagrama de caso de uso.



Associação entre atores



Associação entre casos de uso.

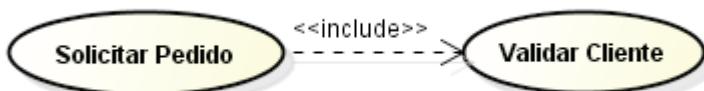


Associação entre um ator e um caso de uso

Adicionalmente, os relacionamentos de agregação e composição também podem ser utilizados no diagrama de casos de uso.

- **Ponto de Inclusão:** É utilizado quando um caso de uso base incorpora um ou mais casos de uso e o utiliza de maneira obrigatória. Este relacionamento é ilustrado no diagrama de casos de uso através de uma seta pontilhada de cabeça aberta, que parte do caso de

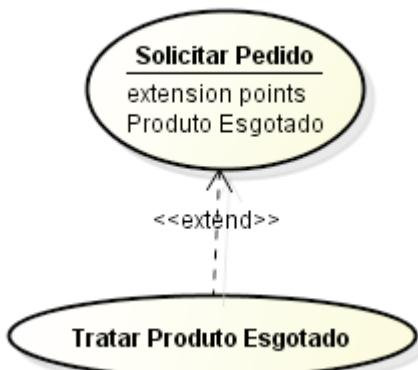
uso que inclui (Solicitar Pedido de Peças, por exemplo) para o caso de uso que é incluído (Validar Cliente, por exemplo), juntamente com o estereótipo <<include>>.



- **Ponto de Extensão:** É utilizado quando um caso de uso base incorpora um ou mais casos de uso e o utiliza de maneira opcional. Este relacionamento é ilustrado no diagrama de casos de uso através de uma seta pontilhada de cabeça aberta, que parte do caso de uso que estende (Tratar Produto Esgotado, este é um fluxo que só é executado quando um pedido é solicitado e não há produtos disponíveis) para o caso de uso que é estendido (Solicitar Pedido, por exemplo), juntamente com o estereótipo <<extend>>.



O ponto de extensão pode ser informado no caso de uso que é estendido.



**Pontos de inclusão e extensão são mecanismos de reuso e modularização**

## Exemplo de uso

A Universidade Aberta do Brasil (UAB) necessita de um sistema para automatizar seus processos acadêmicos (ex., inscrição em disciplinas, lançamento de notas, alocação de recursos para turmas, emissão de históricos escolares, etc.). Para isso, a UAB resolveu contratar uma empresa de desenvolvimento de software.

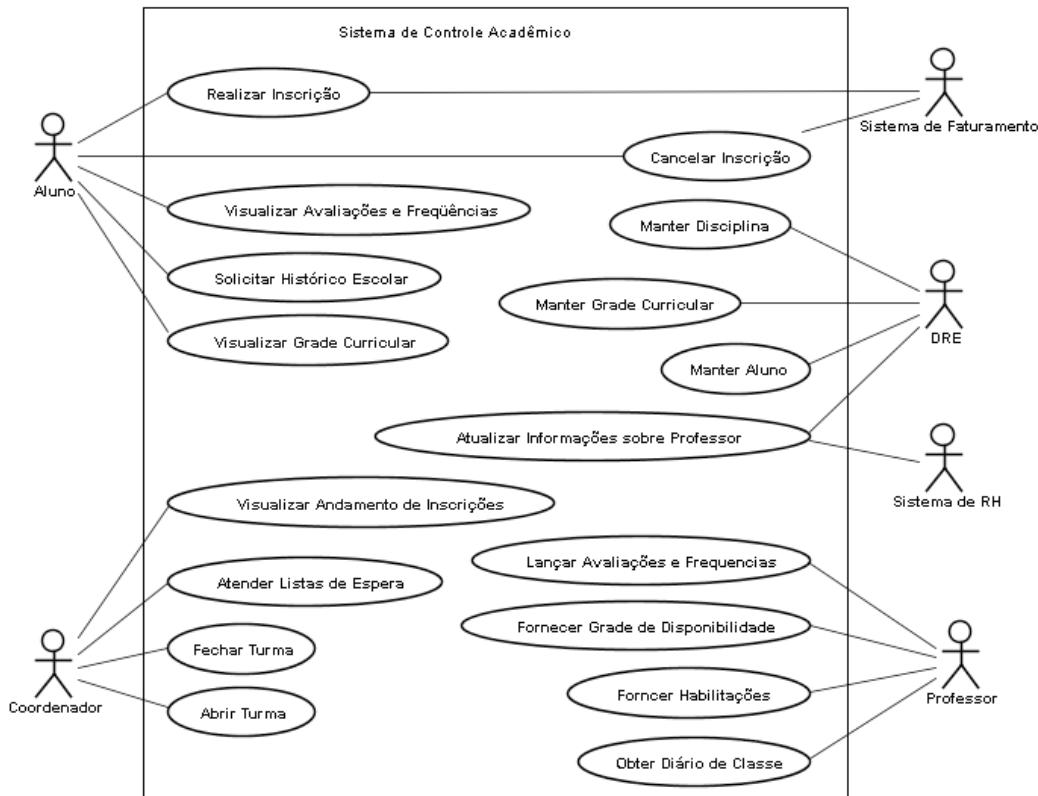
Os seguintes requisitos funcionais foram identificados:

<b>Identificador</b>	<b>Descrição</b>
R1	O sistema deve permitir que os alunos visualizem as notas por semestre letivo.
R2	O sistema deve permitir o lançamento das notas das disciplinas lecionadas em um semestre letivo e controlar os prazos e atrasos neste lançamento.
R3	O sistema deve manter informações cadastrais sobre disciplinas no currículo escolar.
R4	O sistema deve permitir a abertura de turmas para uma disciplina, assim como a definição de salas e laboratórios a serem utilizados e os horários e dias da semana em que haverá aulas em tal turma.
R5	O sistema deve permitir que os alunos realizem a inscrição em disciplinas do semestre letivo.
R6	O sistema deve permitir o controle do andamento das inscrições em disciplinas feitas por alunos.
R7	O sistema deve se comunicar com o Sistema de Recursos Humanos para obter dados cadastrais sobre os professores.
R8	O sistema deve se comunicar com o Sistema de Faturamento para informar as inscrições realizadas pelos alunos.
R9	O sistema deve manter informações cadastrais sobre os alunos e seus históricos escolares.

E os seguintes atores foram identificados para o sistema:

<b>Nome</b>	<b>Descrição</b>
Aluno	Indivíduo que está matriculado na faculdade, que tem interesse em se inscrever em disciplinas.
Professor	Indivíduo que leciona disciplinas na faculdade
Coordenador	Pessoa interessada em agendar as alocações de turmas e professores, e visualizar o andamento
Departamento de Registro Escolar (GRE)	Departamento da faculdade interessado em manter informações sobre os alunos matriculados e sobre seu histórico escolar.
Sistema de Recursos Humanos	Este sistema legado é responsável por fornecer informações cadastrais sobre os professores.
Sistema de Faturamento	Este sistema legado tem interesse em obter informações sobre inscrições dos alunos para realizar o controle de pagamento de mensalidades.

Para este sistema, o seguinte diagrama de caso de uso poderia ser criado.



**Observação:** Este sistema será abordado nos capítulos das unidades 3 e 4 deste livro.

## ATIVIDADES DE AVALIAÇÃO

- Um aluno de uma universidade particular deve escolher disciplinas do semestre. Em seguida ele é alocado às turmas para então receber uma fatura emitida pelo sistema de faturamento com o valor a ser pago em função do número de turmas em que conseguiu vaga. Quais são os atores e casos de uso?
- A secretaria de uma universidade deve cadastrar turmas, apagá-las e modificá-las e enviá-las aos departamentos acadêmico. Quais são os atores e casos de uso?
- Construir um diagrama de casos de uso para o seguinte cenário:

Um cliente deseja um sistema que permite jogar jogo da velha e forca. O sistema é destinado a um usuário e deve armazenar as estatísticas de uma sessão (do lançamento ao término do sistema). Em uma sessão o usuário pode jogar diversas vezes cada um dos jogos. Ao término de cada jogo, atualizam-se as estatísticas da sessão: número de vezes que jogou velha, número de vitórias absoluto e percentual e o mesmo para forca. O usuário deseja que o painel de estatísticas esteja sempre visível.



## 2. Diagrama de Classes

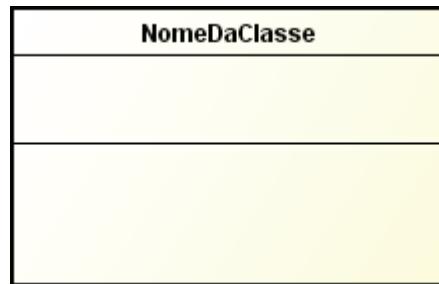
O diagrama de classes é um diagrama estrutural capaz de mostrar a composição interna das entidades envolvidas. Este é possivelmente o diagrama mais utilizado de UML.

### Entidades

As entidades do diagrama de classes representam as necessidades de representação de entidades genéricas, entidades específicas que representam dados e operações e entidades de agrupamento do sistema a ser modelado. As entidades do diagrama de classes são Classes, Interface e Pacote.

### Classe

A principal entidade do diagrama de classes é a **Classe**. Esta entidade é composta de atributos e operações, onde os atributos representam os dados que a entidade é capaz de armazenar e as operações que é capaz de executar. Cada classe é identificada por um nome. Graficamente, a classe é representada por um retângulo dividido em três compartimentos, no compartimento superior deve constar o nome, no compartimento intermediário são adicionados os atributos e no compartimento inferior constam as operações.



Visibilidade diz respeito ao local onde o recurso pode ser acessado. A visibilidade pública permite acesso ao membro por qualquer outra classe, a visibilidade privada permite acesso somente dentro da classe que o define. A visibilidade protegida permite acesso para qualquer classe filha e a visibilidade pacote permite acesso por qualquer classe do mesmo pacote.

O nome da classe deve ser único no diagrama modelado, portanto não podemos ter duas classes com o nome Funcionário (por exemplo) no mesmo modelo. Atributos possuem basicamente as informações de **visibilidade**, nome, tipo e valor padrão da seguinte forma: *visibilidade nome : tipo = valor\_padrão*

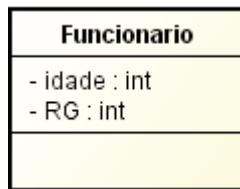
Os valores possíveis para a visibilidade são:

Símbolo	Tipo de Visibilidade
+	Pública
-	Privada
#	Protegida
~	Pacote

O tipo de um atributo pode ser um tipo primitivo como inteiro (int), números de ponto flutuante (double ou float), lógico (boolean); ou pode ser utilizado como tipo de um atributo, uma classe já definida no diagrama.

O valor padrão é um valor que é atribuído inicialmente à variável.

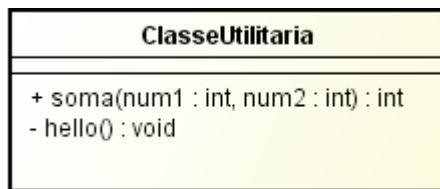
Deste modo exemplos de definição dos atributos idade e RG em uma classe Funcionário são exibidos a seguir:



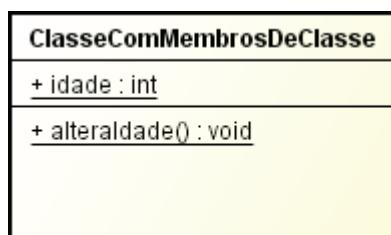
A sintaxe da UML para as operações é:

*visibilidade nome (lista-de-parâmetros) : tipo-de-retorno*

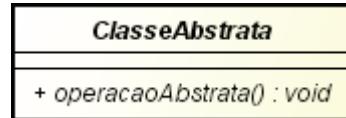
A visibilidade pode assumir um dos tipos de visibilidade já apresentados para atributos. O nome deve ser único (não são permitidas duas operações com o mesmo nome). Os parâmetros são atributos que devem ter valores atribuídos no momento que a operação for chamada. O tipo de retorno pode ser *void* quando não há um valor a ser retornado, pode ser um dos tipos primitivos ou pode ser utilizada uma classe já definida no diagrama.



Para estabelecermos atributos ou operações como membros de classe basta formata-lo como sublinhado. Um membro de classe é aquele comum a todas as instâncias da classe definida. A classe a seguir possui o atributo de classe chamado idade e a operação de classe chamada Alteralidade



Uma classe dita **abstrata** não pode ser instanciada e geralmente pelo menos uma de suas operações é abstrata, ou seja, durante a implementação apenas o cabeçalho é oferecido e a primeira classe filha concreta terá que implementar os métodos abstratos herdados. Para que uma classe seja definida como classe abstrata em UML, o nome da classe deve aparecer em *íntico* (um pouco deitado à direita). As operações abstratas também se diferenciam das operações concretas através da formatação em *íntico*. Segue um exemplo de classe abstrata chamada *ClasseAbstrata* e uma operação chamada *operacaoAbstrata*.



## Interface

Uma interface é uma entidade que possui somente operações públicas e abstratas. A interface possui duas representações básicas em UML: i) Um círculo com o nome da interface, neste caso os detalhes de implementação como atributos e operações não são revelados; e ii) Uma representação semelhante a uma classe, mas com o estereótipo <>interface<> associado.



Vale lembrar que todas as operações de uma interface devem ser públicas e abstratas.

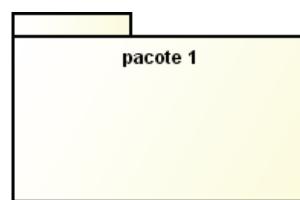
## Objeto

É a instância de uma classe. No diagrama de classes é representado através de um retângulo contendo o nome da instância seguido de dois pontos e o nome da classe base. O texto que representa o nomeInstância:nomeClasseBase aparece em sublinhado. A seguir, a ilustração de um objeto no diagrama de classes.

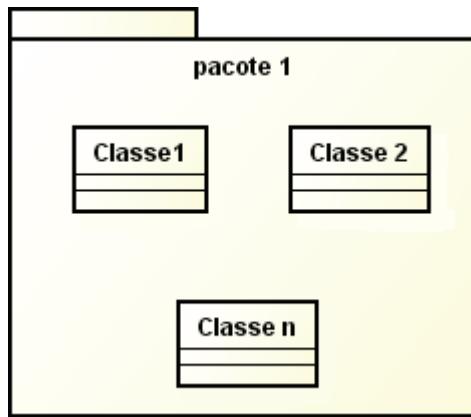


## Pacote

Pacote é uma entidade que pode ser utilizada nos diagramas de Classes e Casos de Uso. Esta entidade é utilizada para agrupar um conjunto de entidades de um diagrama. No diagrama de classes, um pacote é capaz de agrupar classes, interfaces e os relacionamentos que os envolvem. A representação gráfica de um pacote é dada por um retângulo maior com outro retângulo menor no canto superior esquerdo, como segue abaixo.

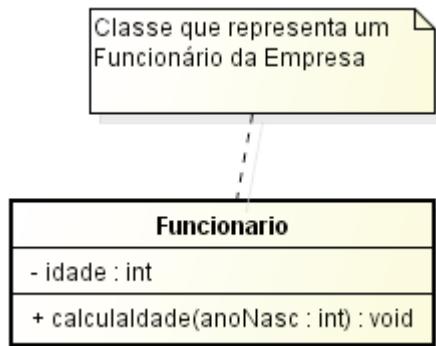


No compartimento do pacote, são inseridas as entidades que estão sendo agrupadas.



## Notas explicativas

Notas explicativas são utilizadas para oferecer alguma informação importante como observações, descrições ou detalhamento de alguma operação aos diagramas de UML. A nota é uma entidade que pode ser utilizada em todos os diagramas de UML, sendo aplicável a todos os seus elementos (tanto entidades quanto relacionamentos). A representação gráfica é dada por um retângulo com o vértice superior direito dobrado. A nota deve ser ligada ao elemento ao qual pertence, para tanto existe um elemento de ligação que une a nota ao elemento ao qual se aplica. A seguir ilustrações do uso de comentário em um diagrama de classes.

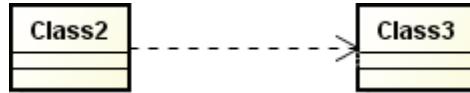


## Relacionamentos

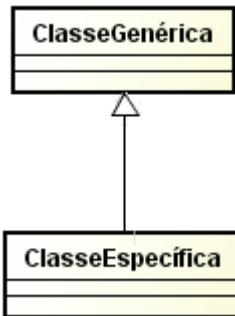
Os relacionamentos do diagrama de classes conectam suas entidades de acordo com as necessidades do sistema a ser modelado. Os relacionamentos do diagrama de classes são Dependência, Generalização/Especialização, Associação, Agregação, Composição e Realização.

O relacionamento de **Dependência** indica que um item usa as informações e serviços de outro item, mas não necessariamente o inverso. A representação gráfica é feita através de uma linha tracejada partindo do item que é dependente. No diagrama de classes a dependência é utilizada principalmente entre classes, entre interfaces, entre uma classe e uma interface, entre uma classe e um pacote ou entre uma interface e um pacote.

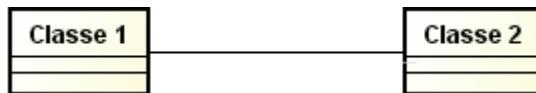
No exemplo a seguir a classe 2 depende da classe 3, possivelmente utiliza seus atributos e/ou métodos.



**Generalização/Especialização** é um relacionamento que conecta entidades mais abrangentes e entidades mais específicas. As entidades devem ser do mesmo tipo, ou seja, duas classes ou duas interfaces, por exemplo. Para verificar se o relacionamento de generalização deve ser utilizado entre duas entidades, basta aplicar o termo *é um*. Em UML, a notação utilizada para representar a generalização é uma linha contínua com um triângulo na extremidade da entidade mais genérica. A entidade mais genérica é conhecida como pai e a entidade mais específica é conhecida como filha.



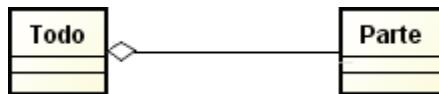
**Associação:** É o relacionamento mais comum em diagramas de caso de uso. Este relacionamento estabelece comunicação entre as entidades envolvidas. A representação gráfica é feita através de uma linha sólida conectando entidades. No diagrama de caso de uso, a associação pode ser utilizada para conectar dois atores ou um ator e um caso de uso.



A associação pode ter algumas informações adicionais explicitadas, tais como: Nome, Papel e Multiplicidade.

Além disto, Agregação e Composição são dois tipos de associação, estes tipos também serão abordados na próxima seção.

- **Agregação:** Em uma associação, não há entidade mais importante que a outra. Porém, em alguns casos uma das entidades é parte da outra, neste caso o relacionamento de agregação deve ser utilizado. O relacionamento de agregação é um tipo de associação, mas conecta as entidades de maneira “mais forte”. A composição é representada em UML através de uma linha sólida com um losango não preenchido na extremidade próxima à entidade que representa o todo.

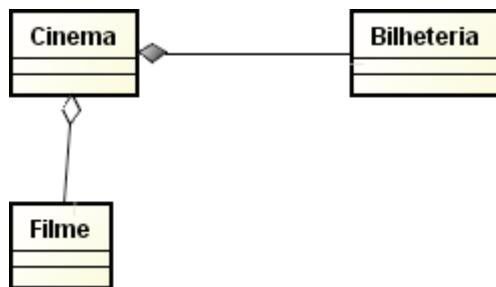


- **Composição:** É uma agregação de fato, o todo é composto pelas partes. Existe uma relação forte entre o todo e as partes, pois quando o todo é destruído as partes também o serão, ou seja, a eliminação do todo se propaga para as partes. De outra forma, o todo e as

partes têm tempos de vida semelhantes. A agregação é representada em UML através de uma linha sólida com um losango preenchido na extremidade próxima à entidade que representa o todo.

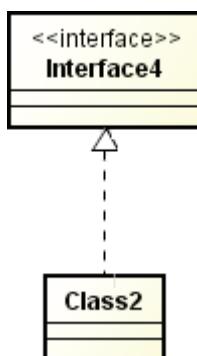


Um exemplo simples capaz de mostrar a diferença entre composição e agregação é dado pelo seguinte contexto: Modelar um sistema de cinema onde aparecem as entidades Cinema, Bilheteria e Filme. Se dúvida o relacionamento Cinema-Bilheteria é mais forte que o relacionamento Cinema-Filme. Um cinema pode existir sem exibir filmes (pode exibir documentários, por exemplo), mas um cinema não pode existir sem uma bilheteria (para um estabelecimento comercial).



Outro exemplo simples capaz de diferenciar associação, agregação e composição seria pensar nas seguintes entidades: Ser humano vivo, coração, braço e blusa. Ao pensarmos em um ser humano vivo, este **tem** coração, **tem** braços e **tem** blusa, porém o coração é vital ao ser humano (Composição), já os braços são extremamente importantes, mas não são vitais, teoricamente podem ser retirados sem prejuízo à vida (agregação). A blusa é utilizada pelo ser humano, mas se retirarmos a blusa deste não haverá dano físico ao mesmo.

**Realização:** Relacionamento que indica um contrato especificado por uma entidade, que a outra entidade deve executar. No diagrama de classes este relacionamento envolve interface e classe. A representação é feita através de uma linha tracejada com seta de cabeça fechada não preenchida. A seguir, segue a ilustração do relacionamento.

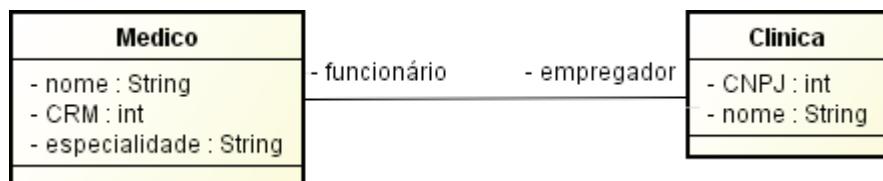


Informações mais detalhadas em relação ao relacionamento de associação podem ser fornecidas. As informações possíveis são nome e sua direção, papel e multiplicidade.

O **nome** pode ser utilizado para descrever o relacionamento de associação entre duas entidades. Um exemplo pode ser dado em relação a duas classes, uma chamada clínica e outra chamada médico. Neste caso, uma clínica tem médicos de maneira não essencial, portanto o relacionamento de associação é o mais indicado. Por sua vez podemos informar que os médicos consultam em uma clínica e a **direção** deste nome, parte do médico para a clínica (O contrário não faz sentido). O nome deve aparecer próximo à linha do relacionamento e a direção é representada por um triângulo preenchido que indica a direção. A seguir a representação deste cenário e destes elementos da UML.



Quando uma entidade participa de uma associação, ela pode desempenhar um **papel**. É possível explicitar o papel que cada entidade desempenha em uma associação. O papel é representado através de texto próximo à entidade que desempenha o papel na associação. Graficamente é mostrado o papel para o cenário do Médico-Clínica.



Outra informação importante que pode ser explicitada é a **multiplicidade** dos membros da associação. A multiplicidade explica a quantidade de objetos que podem ser conectados em relação à associação. A quantidade é definida através de valores exatos (1) ou de valores mínimo e máximo da quantidade de objetos (1..\* - um ou muitos).

As multiplicidades possíveis são:

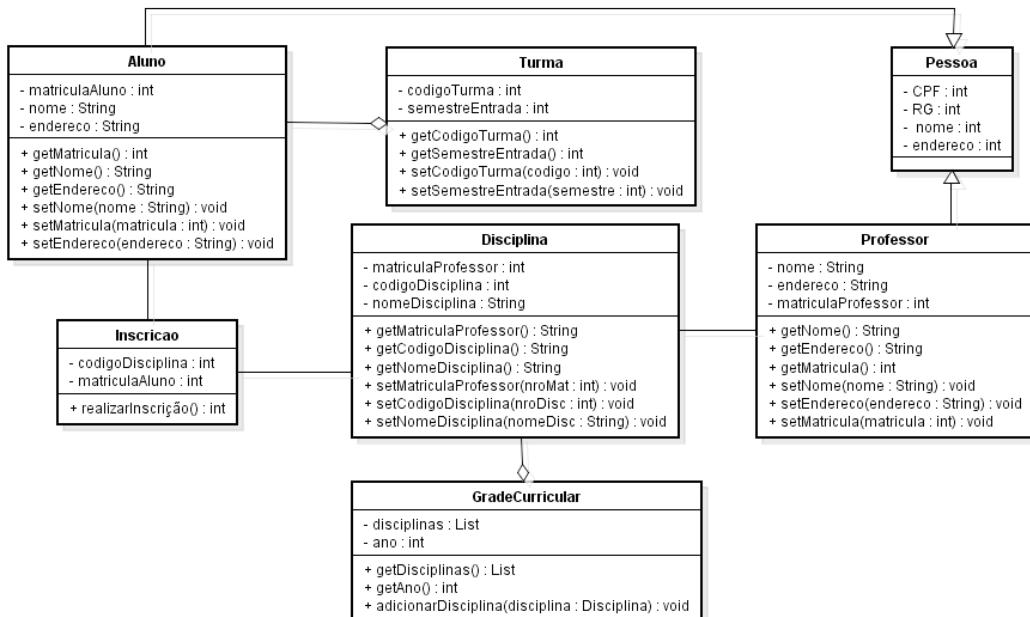
Representação	Significado
1	Exatamente um
0..*	Zero ou mais
*	Zero ou mais
1..*	Um ou mais
0..1	Zero ou um
2..8	Intervalo específico (2,3,4,5,6,7,8), os valores podem ser outros.
4..7,9	Combinação (4,5,6,7 ou 9)

Um médico pode consultar em várias clínicas e uma clínica possui um ou mais médicos que consultam, por exemplo. Observe neste exemplo que para ler a multiplicidade deve-se partir da entidade e ler a multiplicidade associada à entidade de destino.



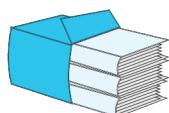
## Exemplo de uso

Para este Sistema, o seguinte diagrama de classes poderia ser criado.



## ATIVIDADES DE AVALIAÇÃO

- Utilize o astah para se familiarizar com os elementos do diagrama de classes. Crie um diagrama de classes e adicione as entidades apresentadas neste capítulo.
- Modele um diagrama de classes para um sistema odontológico. Pense funcionalidades, classes necessárias e relacionamento entre elas.
- Modele um sistema de controle de uma mercearia. Pense funcionalidades, classes necessárias e relacionamentos entre elas.



## SÍNTESE DO CAPÍTULO

Nesta unidade são apresentados os diagramas de classes e diagrama de casos de uso. Estes diagramas são essenciais ao desenvolvimento de um software e são associados às fases de elicitação e especificação de requisitos, análise e projeto apresentadas no Capítulo 2.



## REFERÊNCIAS

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML.** Elsevier.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: Guia do Usuário.** 2<sup>a</sup> Ed, Editora Campus.

FOWLER, M. **UML Essencial.** 3<sup>a</sup> Ed, Editora Bookman.

**UML** - <http://www.uml.org/>

# Capítulo

# 4

## Diagramas da UML - Parte 2

### Objetivos:

- Os diagramas abordados no capítulo anterior são capazes de explicitar aspectos estáticos do sistema que demonstram a estrutura de implementação do sistema, além de suas permissões e funcionalidades. Além dos diagramas de casos de uso, os aspectos dinâmicos do sistema são modelados principalmente através do diagrama de sequência e do diagrama de atividades do sistema, estes são importantes para mostrar a sequência de mensagens trocadas entre os objetos e mostrar detalhes de implementação de algum algoritmo a ser utilizado.



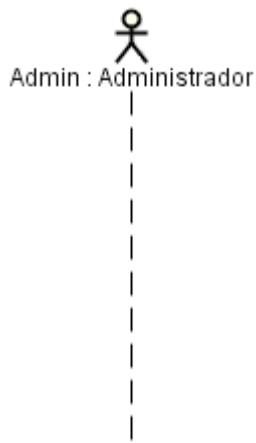


## 1. Diagrama de Sequência

Como já citamos, as fases de um processo de desenvolvimento realizam aproximações rumo ao código. De um modo geral, o diagrama de casos de uso é criado de modo a dar uma visão das funcionalidades que devem ser desenvolvidas e o diagrama de classes é utilizado em seguida para modelar as entidades de software responsáveis por representar, a nível de código, as funcionalidades que aparecem no diagrama de casos de uso. O diagrama de classes consegue modelar os atributos (dados) e operações, mas não possui artifícios adequados para explicitar a sequência de mensagens a ser trocada entre os objetos criados a partir das classes. Adicionalmente os atores podem aparecer para mostrar a interação entre os usuários do sistema e os objetos do sistema.

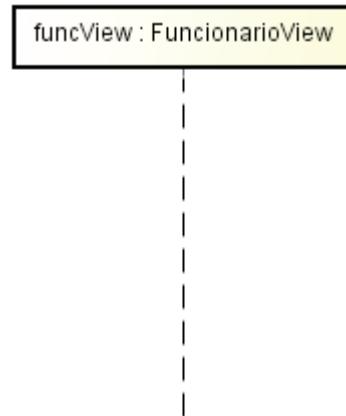
As entidades envolvidas um diagrama de sequência são: **Atores**, **Objetos e linha da vida** e **operadores de controle**. Os objetos e atores relacionam-se no decorrer da linha do tempo através de **mensagens** trocadas.

A representação de Atores é a mesma apresentada no Capítulo 2 da Unidade 3. Porém, estes são representados no diagrama de sequência com uma linha da vida associada, sendo que a linha da vida é representada por uma linha pontilhada que inicia no elemento e desce após este. No diagrama de sequência, os atores podem comunicar-se com objetos.

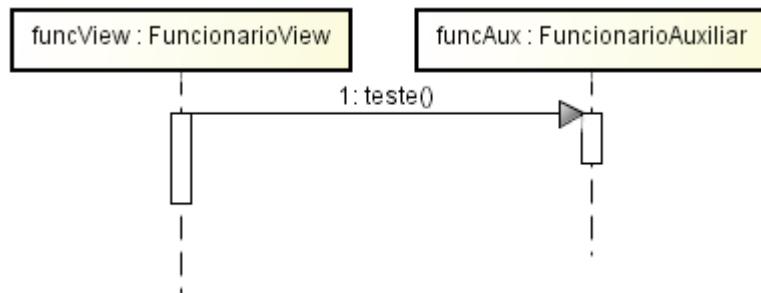


Lembre-se que o diagrama de sequência explicita a execução do sistema. Quando o sistema executa, as classes definidas são instanciadas e tornam-se objetos. Neste caso, o diagrama de sequência é capaz de representar objetos (instâncias das classes definidas no diagrama de classes, por exemplo). A representação de um objeto é feita através de um retângulo que contém o nome da instância, adicionalmente a classe base utilizada pode ser explicitada, e sua respectiva linha da vida também deve constar

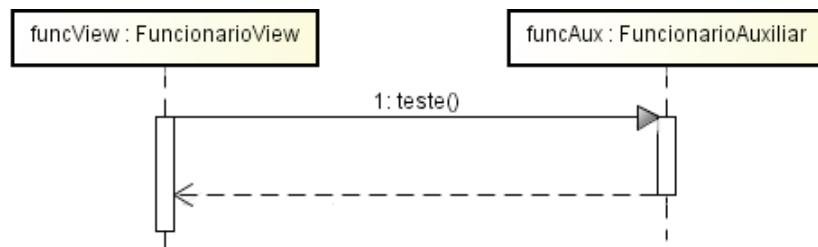
na representação. A seguir um exemplo de um objeto chamado funcView, criado a partir da classe FuncionarioView (classe definida em um diagrama de classes), observe que o identificador do objeto e o tipo são separados pelo operador dois pontos.



Mensagens estão relacionadas às chamadas de operações por parte de uma entidade, elas especificam a comunicação entre objetos e o resultado esperado desta comunicação. A representação gráfica de uma mensagem é feita através de uma seta contínua com cabeça preenchida, partindo do objeto que faz a chamada e chegando ao objeto que tem o método chamado, o nome do método pode ser informado próximo à seta da mensagem. Ao enviar uma mensagem, outro elemento aparece junto à representação: O **foco de controle**. Um foco de controle é um retângulo fino vertical sobreposto à linha de vida que mostra o período durante o qual um objeto está realizando uma ação. Uma ilustração de envio de mensagem é dada a seguir, neste caso, um método chamado teste do objeto funcAux é chamado pela instância funcView. Observe o fluxo de controle que se forma após o envio da mensagem.

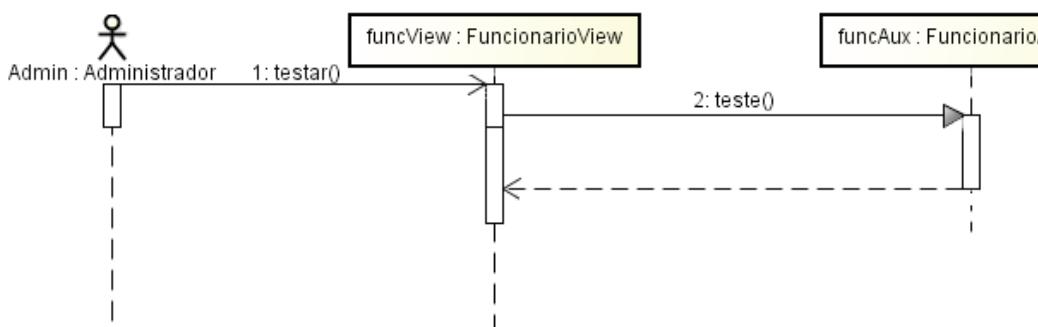


Opcionalmente, pode ser indicado o retorno de uma mensagem síncrona através de uma linha com traço interrompido. Observe que o retorno está associado à utilização de uma mensagem.

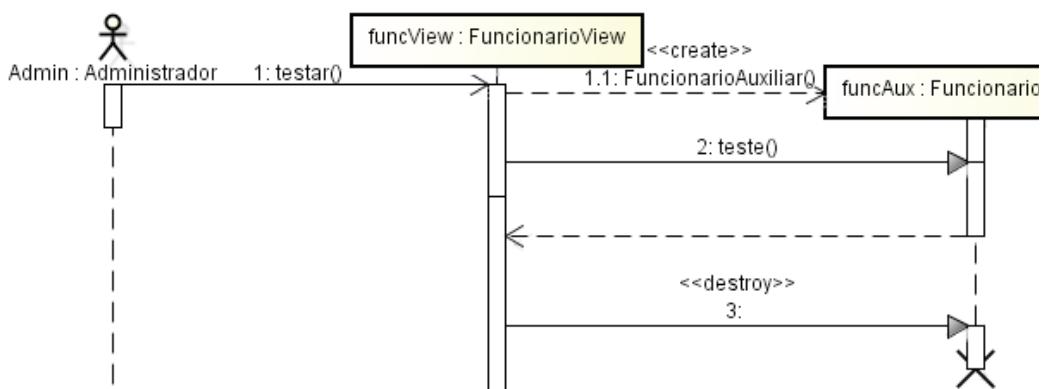


A mensagem apresentada é do tipo mensagem síncrona. Mensagem síncrona é aquela que necessita que o processamento de sua chamada seja executado no objeto de destino para que sua execução continue. Outro tipo de mensagem é a mensagem assíncrona, esta caracteriza-se por realizar o envio de um sinal, neste caso o objeto emissor pode continuar sua execução sem necessitar esperar pela resposta do receptor. Os mecanismos de envio de mensagens suportados pelos sistemas operacionais são o exemplo mais comum deste tipo de mensagem. Além disso, de uma forma geral, as comunicações entre atores e objetos representam trocas de mensagem assíncronas.

Uma mensagem assíncrona é representada através de uma reta contínua com cabeça aberta, partindo do objeto emissor para o receptor. Uma mensagem assíncrona é ilustrada a seguir entre o ator Admin e o Objeto funcView.



Outras mensagens possíveis são as mensagens de criação (*create*) e destruição (*destroy*) de objetos. Na verdade estas são estereótipos associados aos tipos de mensagem citados neste capítulo. Observe na imagem a seguir que a introdução das mensagens de criação e de destruição altera a estrutura do diagrama de sequência modelado. Instância funcAux está um nível abaixo, isto ocorre, pois esta instância passa a existir somente após sua criação através da mensagem *create*. A mensagem de criação está invocando um construtor da classe FuncionárioAuxiliar para criar o objeto funcAux. Observe também que a mensagem de destruição é acompanhada do indicativo de parada do objeto funcAux.



Como o diagrama de sequência mostra o comportamento do sistema ao ser executado, muitas vezes é necessário explicitar uma repetição ou um comando condicional. Para explicitar tais situações existem operadores de controle de vários tipos. Os operadores de controle são representados por um retângulo com uma informação no canto superior esquerdo que indica

**SAIBA  
MAIS**

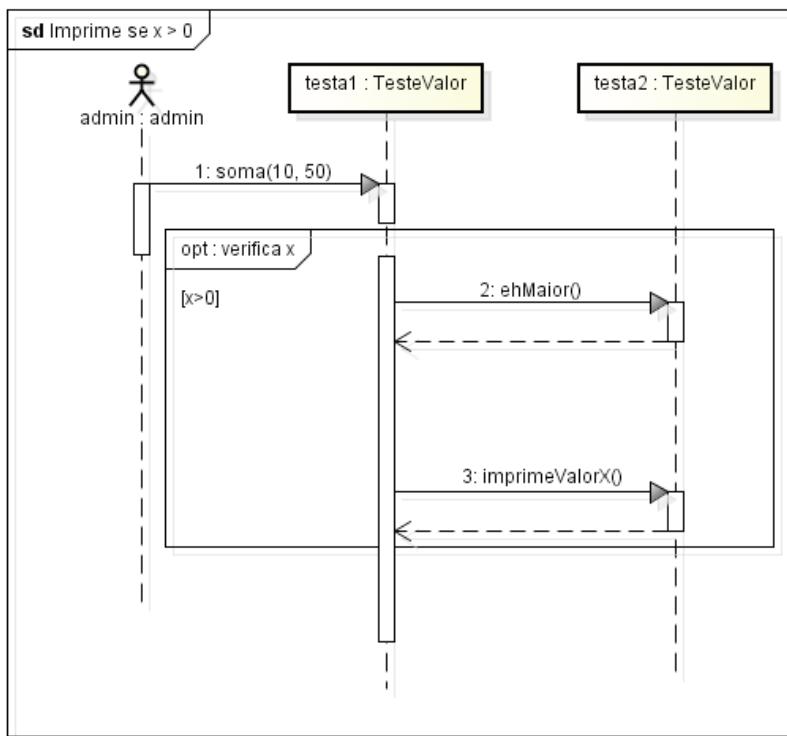
Alguns autores dividem a classificação de operadores de controle apresentada aqui através de dois elementos: i) Uso de interação – que representa a tag `ref`; e ii) Fragmento combinado que representa as demais tags.

seu tipo dentre os seguintes tipos de tag possíveis. A seguir as principais opções de tag para operadores de controle:

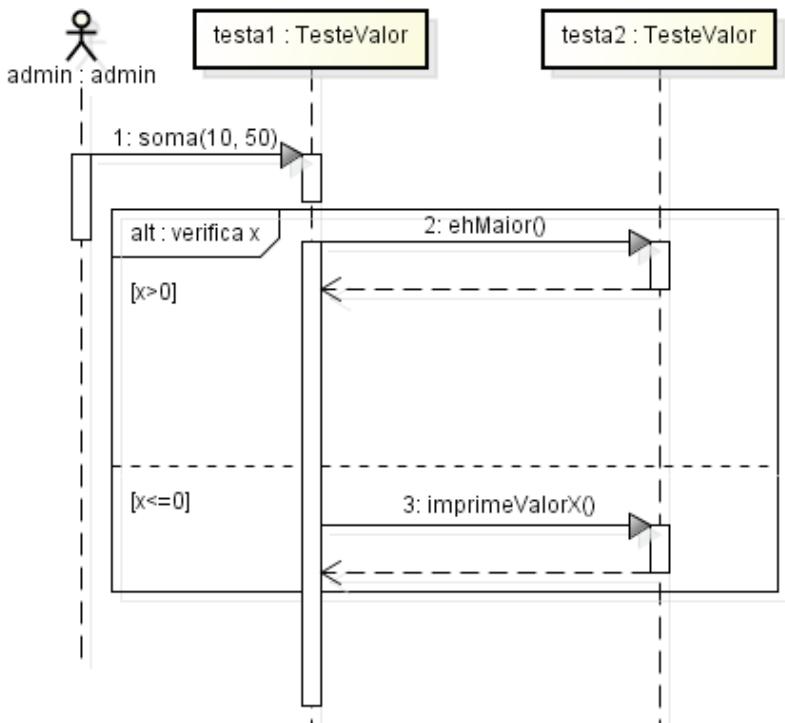
- 1. opt (opcional)** – equivalente ao if sem else;
- 2. alt (condicional)** – equivale ao if com else;
- 3. loop (repetição)** – permite representar um laço de repetição;
- 4. par (execução em paralelo)** – Indica que as mensagens que estão neste elemento podem ser executadas em paralelo (simultaneamente);
- 5. ref (referencia)** – Permite que a solução seja modularizada através de módulos e estes são acessados através de referência.

Para que uma mensagem seja associada a um operador de controle, basta que esta esteja dentro do operador. A seguir serão apresentados exemplos do uso de cada um dos operadores de controle citados.

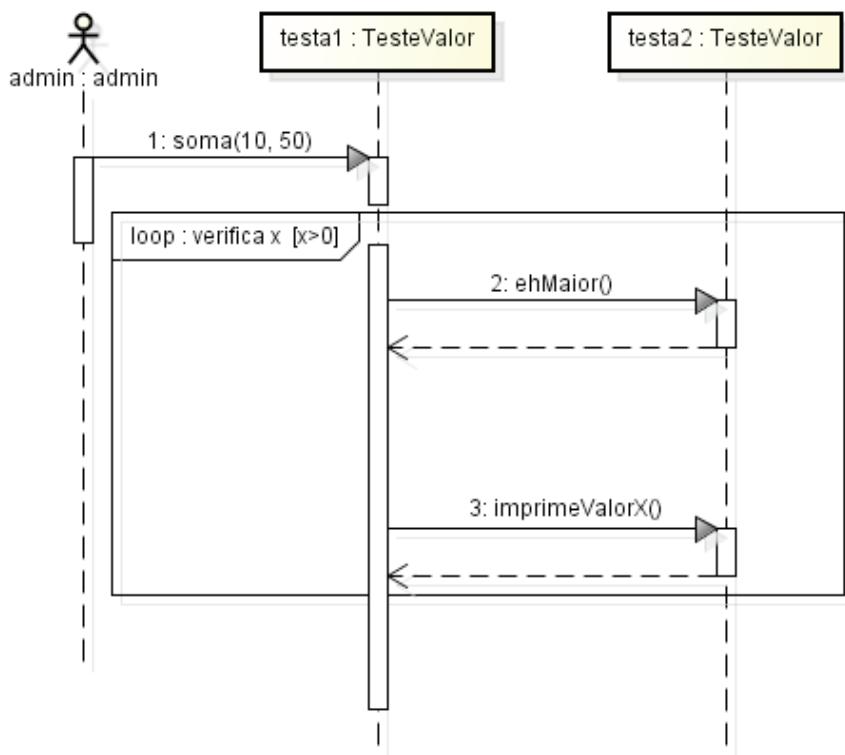
Inicialmente vamos analisar o operador de controle opt. Este operador possui uma condição de guarda associada (representada neste exemplo por  $[x>0]$ ). Além disto, observe que este operador pode ser nomeado com o objetivo que possui como verifica x deste exemplo. As mensagens ehMaior e imprimeValor X, que estão dentro do opt, pertencem ao operador e tem sua execução condicionada à satisfação da condição  $x>0$ . O nome do diagrama de sequência pode aparecer, neste caso o diagrama chama-se imprime se  $x>0$ . Nos próximos exemplos, o nome do diagrama será omitido para dar uma melhor legibilidade às figuras.



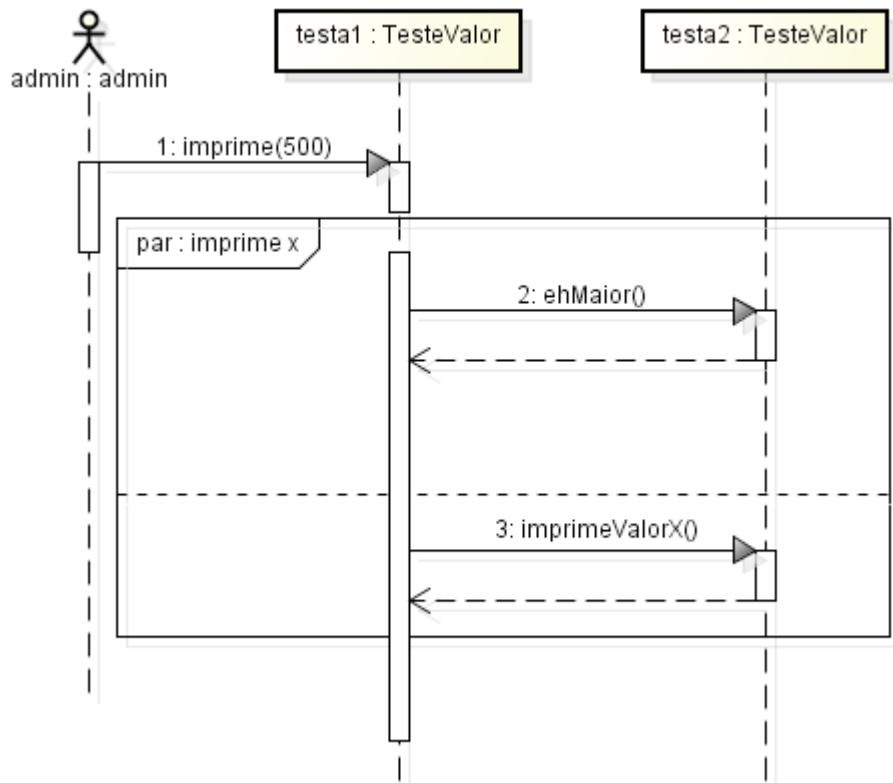
O operador de controle alt possui duas condições de guarda associada (representada neste exemplo por  $[x>0]$  e  $[x\leq 0]$ ). Este operador também pode ser nomeado com o objetivo que possui como verifica x deste exemplo. As mensagens ehMaior e imprimeValor X, que estão dentro do alt, pertencem ao operador e tem sua execução condicionada à satisfação da condição  $x>0$ , para ehMaior, ou  $x\leq 0$ , para imprimeValorX.



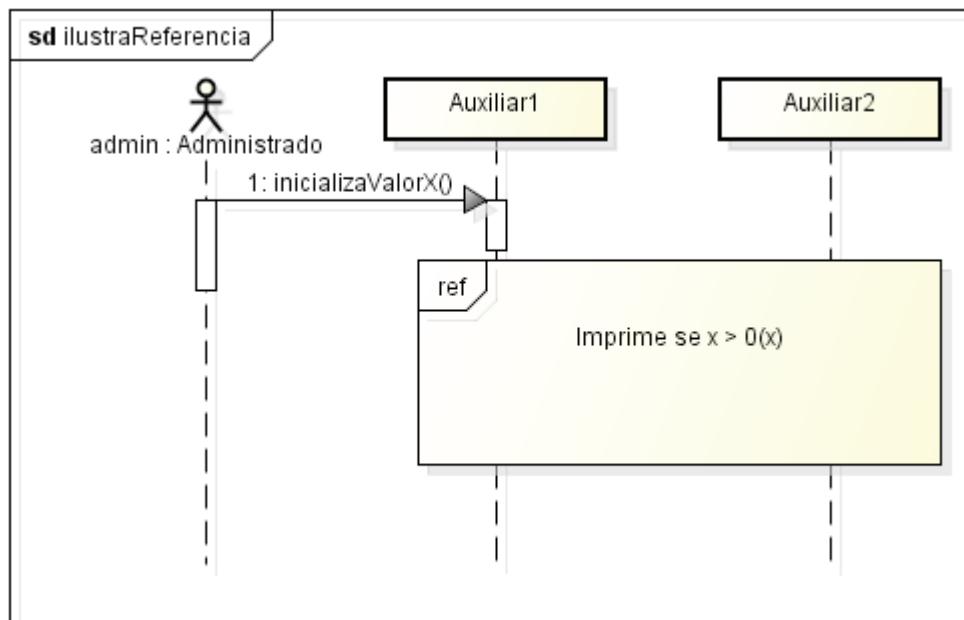
Em relação ao operador de controle loop, este é indicado para repetições. A condição de continuação pode ser explicitada através de uma condição de guarda. Uma ilustração de uso da tag loop é dada a seguir:



O valor par indica que as mensagens existentes dentro deste comando serão executadas em paralelo. A seguir uma ilustração de uso deste recurso, onde **ehMaior** e **imprimeValorX** são executados em paralelo após a chamada de **imprime**.

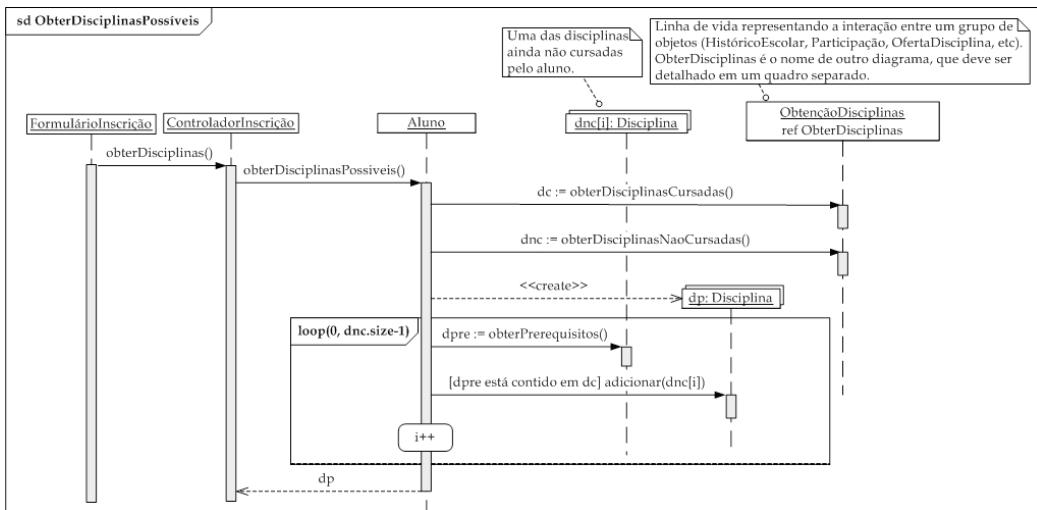


O valor ref indica que outro diagrama será referenciado. Tomemos o diagrama identificado como imprime se  $x > 0$ . Este diagrama pode ser referenciado em outro diagrama através de ref. Isto implica que após a chamada à operação inicializaValorX0 pelo usuário, o sistema executará os passos contidos no diagrama Imprime se  $x > 0$  e neste caso passará o valor de x (observe que no operador, o valor de x está entre parentesis).



## Exemplo de uso

No contexto de nosso sistema de controle acadêmico, o diagrama de sequência a seguir poderia ser proposto:



O diagrama modela a interação entre alunos e disciplinas durante o procedimento de inscrição.



## ATIVIDADES DE AVALIAÇÃO

1. Utilize o astah para se familiarizar com os elementos do diagrama de sequência. Crie um diagrama de sequência e adicione os elementos apresentados neste capítulo.
2. Modele um diagrama de sequência para um sistema de controle docente.
3. Crie o diagrama de sequência para o sistema odontológico. Utilize como base as classes definidas na questão 2 do exercício da página 49 (Capítulo 3).
4. Modele o diagrama de sequência para um sistema de mercearia.



## 2. Diagrama de Atividades

O diagrama de atividades é um diagrama comportamental que pode ser utilizado para explicitar características desde a fase de elicitação e especificação de requisitos (relacionado ao esclarecimento passos de um caso de uso) até a fase de projeto (estabelecendo detalhes de um algoritmo a ser utilizado por uma função ou a sequência de operações).

O Diagrama de Atividades não consegue detalhar como os objetos interagem, portanto deve ser utilizado de maneira complementar aos diagramas de caso de uso e sequência, não em substituição.

Segundo Booch, Rumbaugh e Jacobson o diagrama de atividades é essencialmente um gráfico de fluxo que mostra o fluxo de controle entre atividades, podendo explicitar situações de concorrência e ramificações de controle.

Este diagrama costuma conter Ações, Ações de Comportamento, Nós de objeto, Fluxos de Controle, Nó inicial, Nó final de atividade, Final de Fluxo, Decisão, Bifurcação, União, Partições, Sinal de Envio de Ação, Sinal de Recebimento de Ação, Conectores e Relacionamento de Dependência.

### Ações

Uma ação é representada por um retângulo de bordas arredondadas com o nome da ação associada. Selecionar site, ordenar nomes ou cadastrar Funcionários são exemplos de texto que pode ser associado a uma ação em um diagrama de atividades. A seguir é ilustrada a representação de uma ação.

ação1

### Ações de comportamento

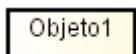
Uma ação de comportamento é utilizada para representar um conjunto de ações. Ela realiza uma ligação entre dois diagramas de atividades diferentes, onde um dos diagramas utiliza o outro como uma de suas ações. A representação é semelhante à representação da ação, com a adição de um pequeno ícone em formato de tridente no canto inferior direito.

Ação de  
comportamento



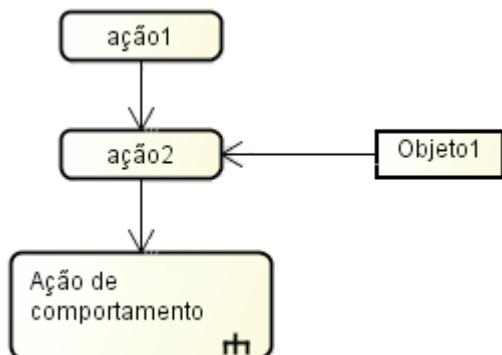
## Nós de objetos

Um diagrama de atividades pode ser utilizado para representar um fluxo de dados. Neste contexto, objetos podem ser explicitados através de nós de objetos para explicitar os objetos necessários para que uma ação possa ser realizada ou os objetos gerados após a finalização de uma ação. A representação de objetos no diagrama de atividades é feita através de um retângulo com o nome do objeto dentro.



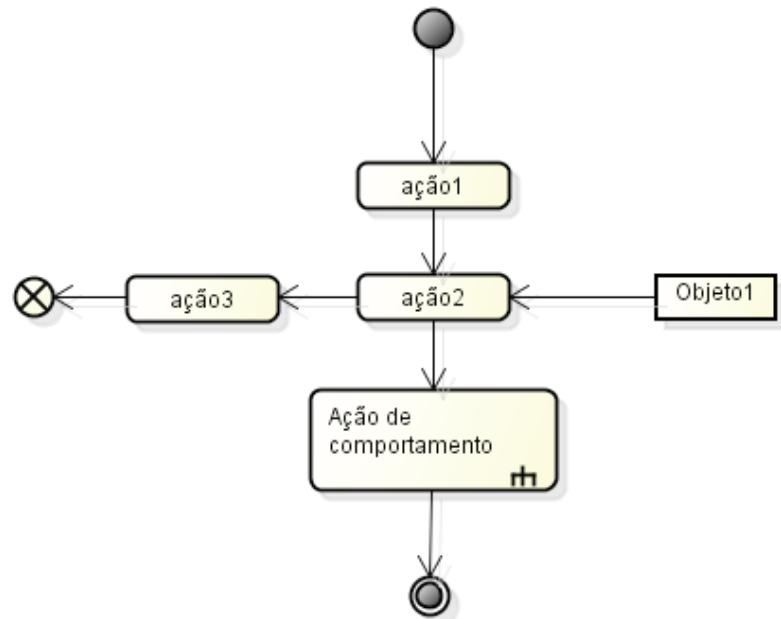
## Fluxos de controle

O fluxo de controle realiza o encadeamento entre as ações de um diagrama de atividades, mostrando a sequência correta a ser seguida para o cumprimento da atividade em questão. O fluxo de controle é representado através de uma seta contínua de cabeça aberta que interliga ações, ações de comportamento e objetos entre si. A seguir a ilustração de fluxos de controle entre as entidades.



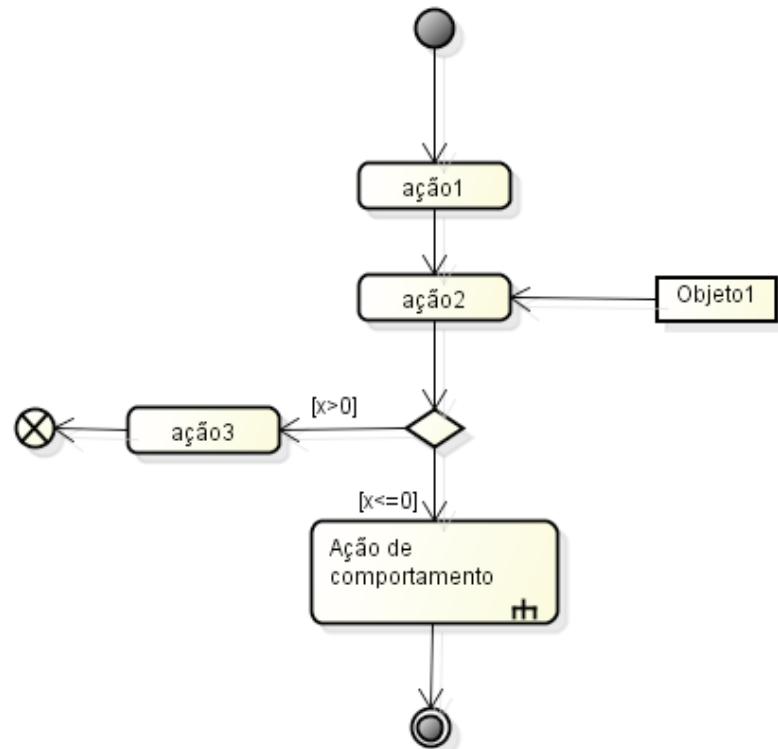
## Nó inicial, nó final de atividade e final de fluxo

Como o próprio nome sugere, o nó inicial é utilizado como ponto de partida em um diagrama de atividades, enquanto que o nó final indica o final da atividade. O nó de final de fluxo encerra apenas o fluxo que chega nele, a atividade pode continuar ocorrendo se outros fluxos existirem, até que um nó de final de atividade apareça e a atividade seja encerrada. O nó inicial é representado por um círculo preenchido, o nó final de atividade é representado por um círculo preenchido circundado por outro círculo e o nó final de fluxo é representado por um círculo com um x na parte interna.



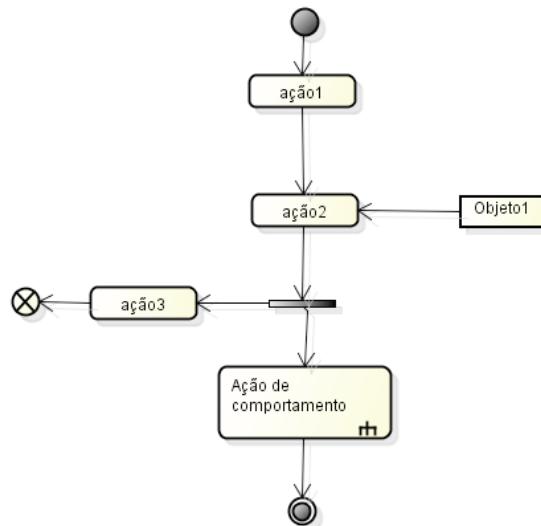
## Decisão, bifurcação e união

Uma decisão é utilizada para representar estrutura condicional no diagrama de atividades. Este elemento é representado graficamente por um losango que tem fluxos de controle que partem deste, geralmente associados às condições de guarda que podem ser definidas nos próprios fluxos de controle. A seguir uma ilustração do fluxo de controle com condições de guarda.

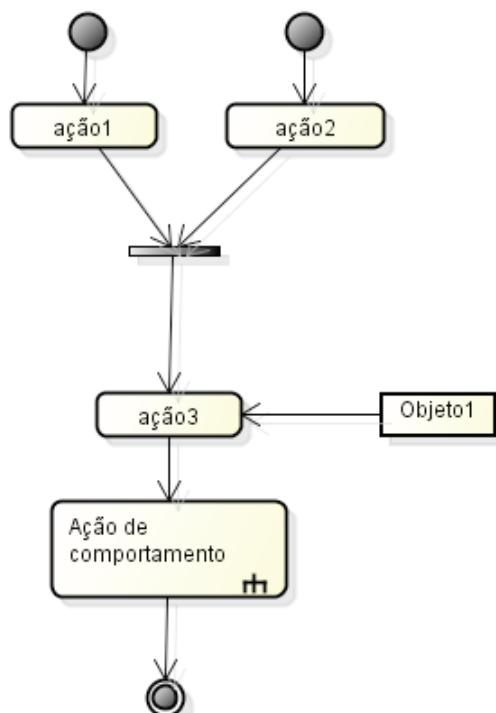


A bifurcação (*fork*) é utilizada para representar paralelismo no fluxo do diagrama de atividades. A união (*join*) representa o inverso, dois fluxos paralelos que são unidos por este elemento. A representação gráfica de ambos é semelhante, uma barra preenchida, a diferença é que na bifurcação há somente uma entrada e várias saídas e na união há várias entradas e somente uma saída.

A seguir, segue um exemplo de bifurcação, observe que somente o fluxo que parte da ação2 e tem saída para ação3 e para a ação de comportamento.



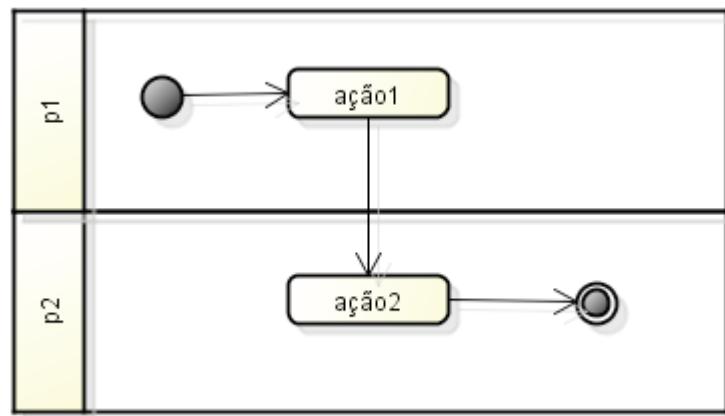
A seguir, segue um exemplo de junção, observe que somente o fluxo que parte da ação1 e da ação2 e tem saída somente para a ação3.



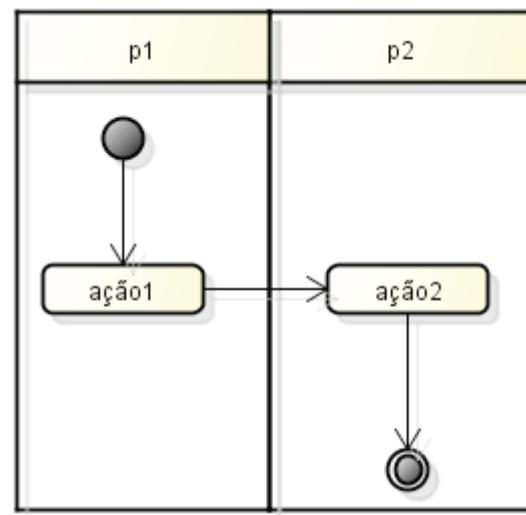
## Partições

Também conhecidas como Raias de Natação ou *Swimlanes*, as partições são capazes de mostrar a interação das ações entre diferentes participantes (como usuário e sistema, módulo de vendas e módulo do fornecedor, por exemplo).

As partições são representadas por retângulos com dois compartimentos, no compartimento menor localiza-se o nome do participante e no compartimento maior localizam-se as ações associadas. A seguir segue uma ilustração duas partições, p1 e p2. Observe que a ação1 é pertencente à partição p1 e a ação 2 é pertencente à partição p2.



Na ilustração anterior as partições aparecem na horizontal, porém estas podem ser representadas no sentido vertical como segue.



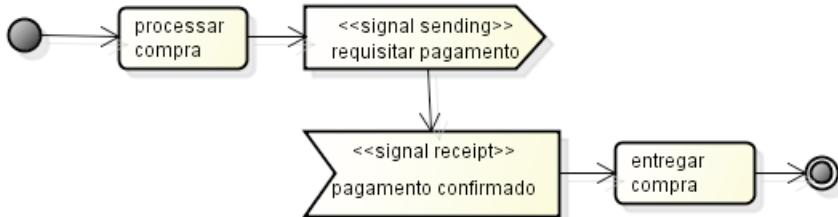
Os demais elementos do diagrama de atividades também podem ser utilizados no contexto de partições.

## Sinal de envio e sinal de recebimento de ação

Sinal de envio e sinal de recebimento de ação são **eventos** que podem ser explicitados no diagrama de atividades. Estes eventos geralmente representam a chamada de uma operação através da interação de ações, uma

enviando sinal e outra recebendo. A ação de envio é representada por um pentágono com um vértice para fora e o sinal de envio é representado por um pentágono com um vértice para dentro. Adicionalmente, os estereótipos *signal sending* e *signal receipt* podem ser utilizados para envio e recebimento, respectivamente.

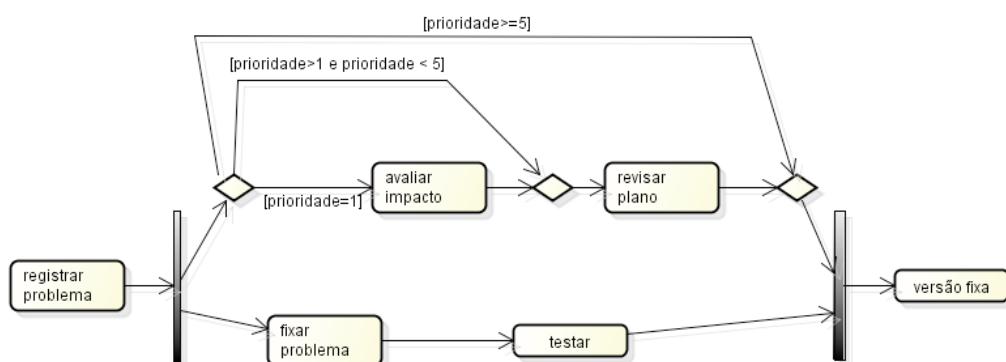
A seguir, uma ilustração de sinais de envio e recebimento é exibida.



## Conectores

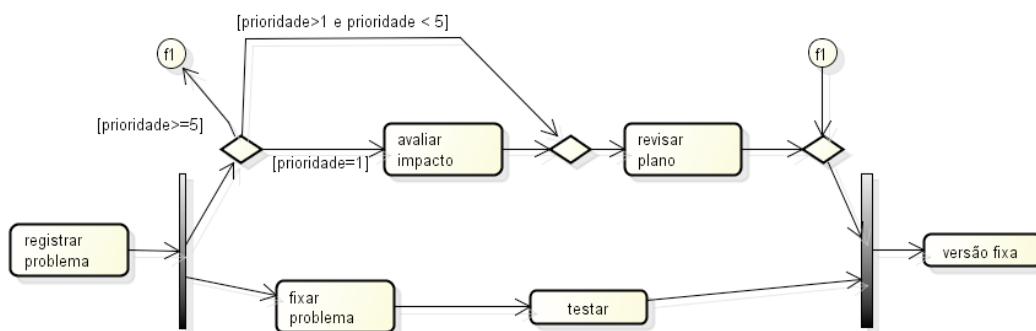
Quando os diagramas de atividades são criados para funcionalidades complexas ou para muitas funcionalidades, uma grande quantidade de entidades e consequentemente de fluxos de controle é criada. Isto pode limitar a legibilidade do diagrama, prejudicando seu entendimento aos demais membros da equipe de projeto ou pelos *stakeholders*.

No exemplo a seguir a legibilidade foi comprometida, devido uma condicional relacionada ao valor de prioridade.



Este exemplo foi retirado do tutorial Diagramas de Atividade e Diagramas de Estado, disponível em <http://www.dca.fee.unicamp.br/~gudwin/ftp/ea976/AtEst.pdf>

Um conector pode ser utilizado com o intuito de melhorar a legibilidade deste. Conectores são elementos que fazem a ligação entre fluxos de um diagrama e são representados através de um círculo com o nome do conector internamente. A seguir o exemplo anterior é explicitado através de conectores.



## Relacionamento de dependência

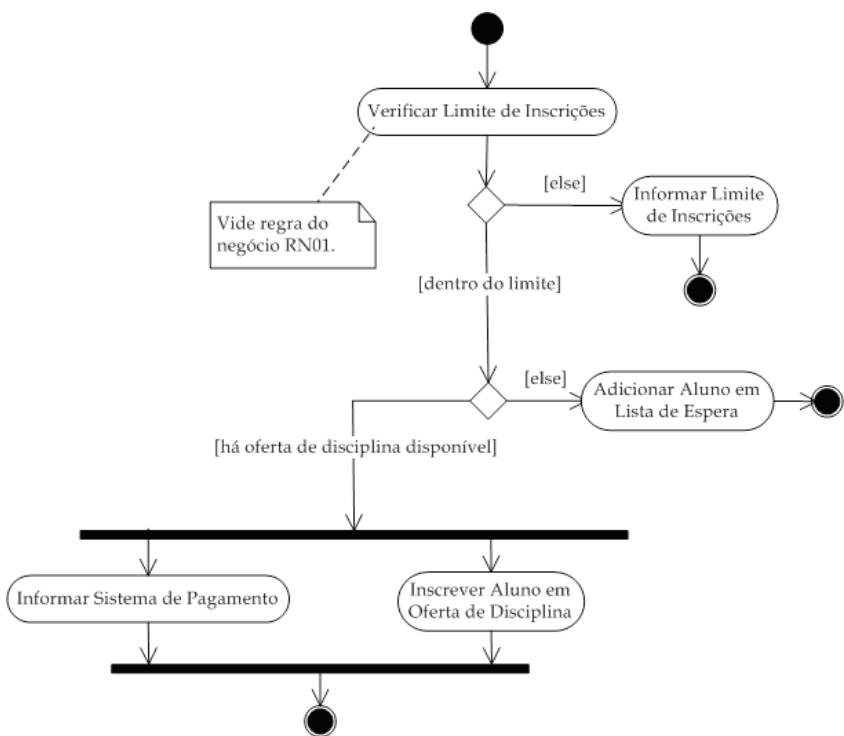
O relacionamento de dependência já foi abordado no diagrama de classes. Este também pode ser utilizado no diagrama de atividades com o mesmo intuito de indicar que um item usa as informações e serviços de outro item, mas não necessariamente o inverso. A representação gráfica é a mesma, feita através de uma linha tracejada partindo do item que é dependente.

Este relacionamento pode ser utilizado entre todos os elementos do diagrama de atividades apresentados neste capítulo.

OBS.: Comentários também podem ser utilizados no diagrama de atividades.

### Exemplo de uso

No contexto de nosso sistema de controle acadêmico, o diagrama de atividades a seguir poderia ser proposto:



O diagrama modela o fluxo de atividades durante o procedimento de inscrição.

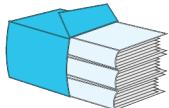


## ATIVIDADES DE AVALIAÇÃO

1. Utilize o astah para se familiarizar com os elementos do diagrama de atividades. Crie um diagrama de atividades e adicione os elementos apresentados (Diagrama de Atividades até Exemplo de uso).
2. Construir um diagrama de atividades para o seguinte cenário:

Um cliente deseja um sistema que permite jogar jogo da velha e forca. O sistema é destinado a um usuário e deve armazenar as estatísticas de uma sessão (do lançamento ao término do sistema). Em uma sessão o usuário pode jogar diversas vezes cada um dos jogos. Ao término de cada jogo, atualizam-se as estatísticas da sessão: número de vezes que jogou velha, número de vitórias absoluto e percentual e o mesmo para forca. O usuário deseja que o painel de estatísticas esteja sempre visível.

3. Modele um diagrama de atividades para um sistema de controle docente.
4. Crie o diagrama de atividades para o sistema odontológico. Utilize como base as classes definidas na questão 2 do exercício da página 49 (Capítulo 3) e o diagrama de sequência do tópico anterior.



## SÍNTESE DO CAPÍTULO

Neste capítulo são apresentados os diagramas dinâmicos de sequência e atividades. Estes diagramas são essenciais ao desenvolvimento de um software e são associados pois modelam o comportamento dinâmico das entidades do sistema, suas interações.



## REFERÊNCIAS

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. Elsevier.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: Guia do Usuário**. 2<sup>a</sup> Ed, Editora Campus.

FOWLER, M. **UML Essencial**. 3<sup>a</sup> Ed, Editora Bookman.

**UML** - <http://www.uml.org/>



# Capítulo

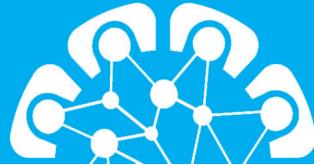
# 5

## Outros diagramas da UML e padrões de projeto

### Objetivos:

- Outros diagramas de UML podem ser utilizados para suprir necessidades específicas de modelagem do sistema. Diferentes domínios podem ter seus sistemas projetados e cada domínio possui suas especificidades que devem ser explicitadas da melhor maneira possível. Muitas vezes, um diagrama não consegue explicitar todas as necessidades relacionadas ao domínio do problema, portanto outros diagramas devem ser criados de forma complementar. Além disto, durante a fase de projeto, padrões de projeto podem ser utilizados de modo a facilitar o desenvolvimento e para que a implementação seja feita da maneira mais adequada possível. Este Capítulo apresenta em linhas gerais os demais diagramas de UML e introduz o conceito de padrões de projeto.





## 1. Outros diagramas da UML

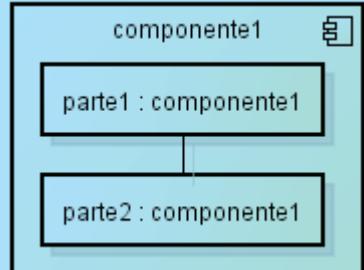
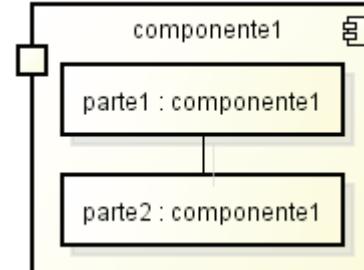
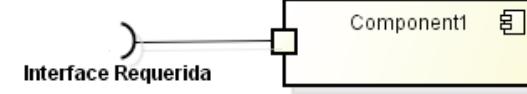
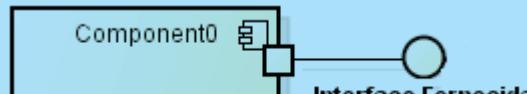
Até o momento foram apresentados o diagrama estático de classes, os diagramas dinâmicos de casos de uso, atividades e o diagrama de sequência, representando os diagramas de interação. Neste capítulo, os diagramas de Componentes (Estático), Instalação (Estático), Estrutura (Estático), Máquina de Estados (Comportamental) e Comunicação (Interação) serão apresentados em linhas gerais.

### Diagrama de componentes

Um componente representa o código de uma funcionalidade ou conjunto de funcionalidades que pode ser reutilizado em diferentes projetos. Os componentes disponibilizam uma forma de acesso, para que possam ser utilizados. Eles podem ser combinados para dar origem a um componente composto por vários outros.

O diagrama de componentes é um diagrama estrutural que vislumbra explicitar por meio de interfaces de comunicação, componentes e relacionamentos. Este é importante quando o sistema estiver dividido em componentes ou quando componentes reutilizáveis forem utilizados.

ELEMENTO	Descrição	Representação
<b>Componente</b>	Representa uma parte do sistema que pode ser utilizada de maneira independente.	
<b>Parte</b>	Um componente pode ser composto de várias partes de implementação. As partes são representadas no contexto dos componentes que as possuem.	<p>Ex1:</p> <p>Ex2:</p>

ELEMENTO	DESCRÍÇÃO	REPRESENTAÇÃO
<b>Conector</b>	Relacionamento de comunicação entre duas partes ou duas portas.	
<b>Porta</b>	É um mecanismo de comunicação do componente. Esta recebe mensagens de acordo com interfaces especificadas.	
<b>Interface</b>	Entidade já apresentada no capítulo do diagrama de classes (Unidade 3 – Capítulo 2).	
<b>Interface Requerida</b>	É referente à comunicação entre componentes. Mostra qual o tipo de informação esperada (requerida) pelo componente.	
<b>Interface Fornecida</b>	É referente à comunicação entre componentes. Mostra qual o tipo de informação fornecida pelo componente. OBS.: geralmente há a combinação dos conceitos de interface requerida por um componente e interface fornecida, por outro componente.	

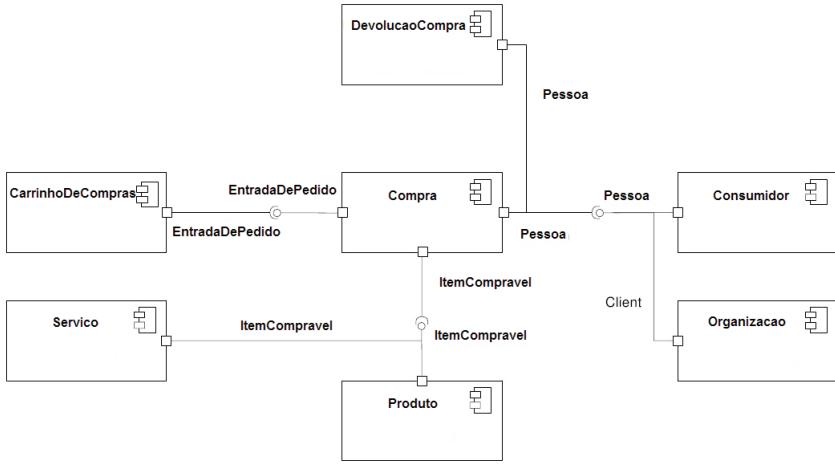
ELEMENTO	Descrição	Representação
<b>Classificador</b>	<p>É um nome genérico que se dá a elementos de modelo que descrevem características comportamentais e estruturais.</p> <p>São considerados classificadores: classe, ator, componente, tipo de dado, interface, nodo, sinal, subsistema e caso de uso.</p> <p>A representação desta entidade é semelhante à representação de artefato, a diferença é que o nome do classificador aparece em negrito.</p>	<b>Classificador1</b>
<b>Artefato</b>	Os artefatos são utilizados para modelar elementos como documentos, tabelas e arquivos.	<b>Artefato1</b>
<b>Relacionamento de Dependência</b>	Relacionamento já apresentado no diagrama de classes. Pode ser utilizado aqui para informar que uma interface requerida depende de uma interface fornecida, por exemplo.	- - - - - →
<b>Realização</b>	Relacionamento já apresentado no capítulo do diagrama de classes (Unidade 3 – Capítulo 2).	- - - - →

## Ilustração do diagrama de componentes

Uma ilustração do diagrama de componentes é dada a seguir no contexto de um sistema de Vendas.

# ! ATENÇÃO

Este diagrama foi criado com base no material do professor Ricardo R. Gudwin, disponível em <http://www.dca.fee.unicamp.br/~gudwin/>



## Diagrama de implantação

No Tópico 1 do Capítulo 1 foram apresentadas as atividades presentes em um processo de desenvolvimento. Os diagramas até então preocupam-se em oferecer subsídios à fase de desenvolvimento. O diagrama de instalação é um diagrama estrutural que busca detalhar como deve ser feita a implantação do sistema (última fase do processo apresentado). Este diagrama mostra como a estrutura física relaciona-se com um conjunto de artefatos de software que será disponibilizado.

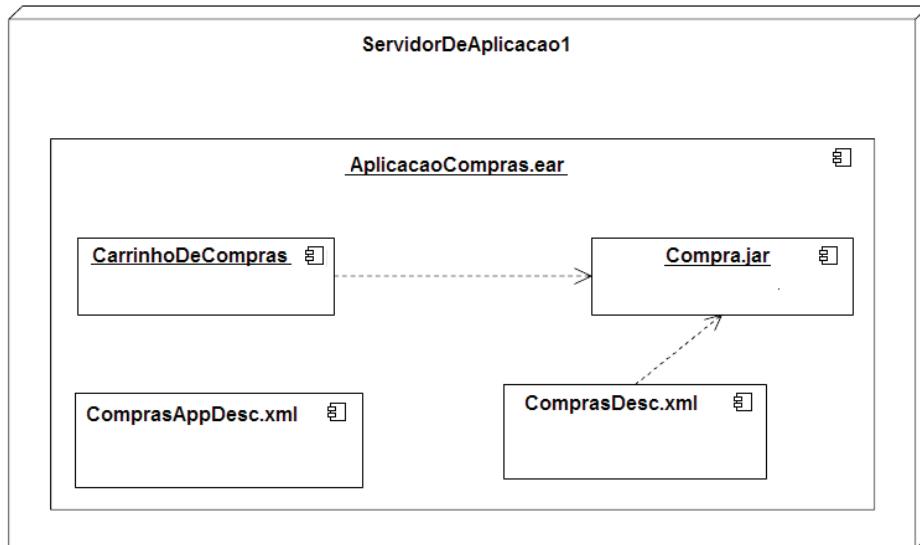
ELEMENTO	Descrição	Representação
<b>Artefato</b>	São representações do software como ele é implantado. Geralmente representam arquivos de instalação, arquivos de configuração ou arquivos de dados, por exemplo.	«artifact» <b>Order.jar</b>
<b>Nó</b>	É um elemento físico que é utilizado para instalar e executar os artefatos de um sistema. Nós podem ser conectados.	<b>Nó</b>
<b>Instância do Nó</b>	Uma instância do nó definido.	<b>instanciaNó : Nó</b>
<b>Componente</b>	Apresentado na tabela do Diagrama de Componentes.	<b>Componente1</b>
<b>Instância do Componente</b>	Instância de um componente definido.	<b>instanciaComponente : Componente1</b>
<b>Objeto</b>	Entidade já apresentada no capítulo do diagrama de Classes (Unidade 3 – Capítulo 2).	<b>objeto : Class</b>
<b>Interface</b>	Entidade já apresentada no capítulo do diagrama de classes (Unidade 3 – Capítulo 2).	<b>Interface1</b> ou «interface» <b>Interface1</b>

Os relacionamentos de Realização, Associação, Agregação, Composição, Realização e Dependência podem ser utilizados neste diagrama. Estes relacionamentos já foram apresentados no diagrama de Classes (Capítulo 3 – Tópico 2).



## Ilustração do diagrama de implantação

Uma ilustração do diagrama de implantação é dada a seguir no contexto de um sistema de Vendas.



Este diagrama foi criado com base no material do professor Ricardo R. Gudwin, disponível em <http://www.dca.fee.unicamp.br/~gudwin/>

## Diagrama de estrutura composta

Este é um diagrama estrutural muito semelhante ao diagrama de componentes. Seu intuito é mostrar a estrutura interna de uma classe e colaborações, permitindo que uma classe complexa seja mostrada como um conjunto de partes.

ELEMENTO	Descrição	Representação
<b>Classe</b>	Entidade já apresentada no capítulo do diagrama de Classes (Unidade 3 – Capítulo 2).	
<b>Classe Estruturada</b>	Uma classe composta de outras classes. Pode ser utilizada para mostrar as estruturas de dados utilizadas somente naquela classe.	
<b>Parte</b>	É uma unidade de implementação que pode ser combinada com outras para representar a estrutura interna de uma classe estruturada.	<p>Ex1:</p> <div style="border: 1px solid black; padding: 5px; background-color: #e0f2e0;">parte1 : Classe</div> <p>Ex2:</p> <div style="border: 1px solid black; padding: 5px; background-color: #e0f2e0;"> <b>ClasseEstruturada</b> <div style="border: 1px solid black; padding: 5px; margin-top: 5px;">             parte1 : Classe         </div> <div style="border: 1px solid black; padding: 5px; margin-top: 5px;">             parte2 : Classe         </div> </div>

Este diagrama foi criado com base no livro UML Essencial presente na bibliografia deste Capítulo.



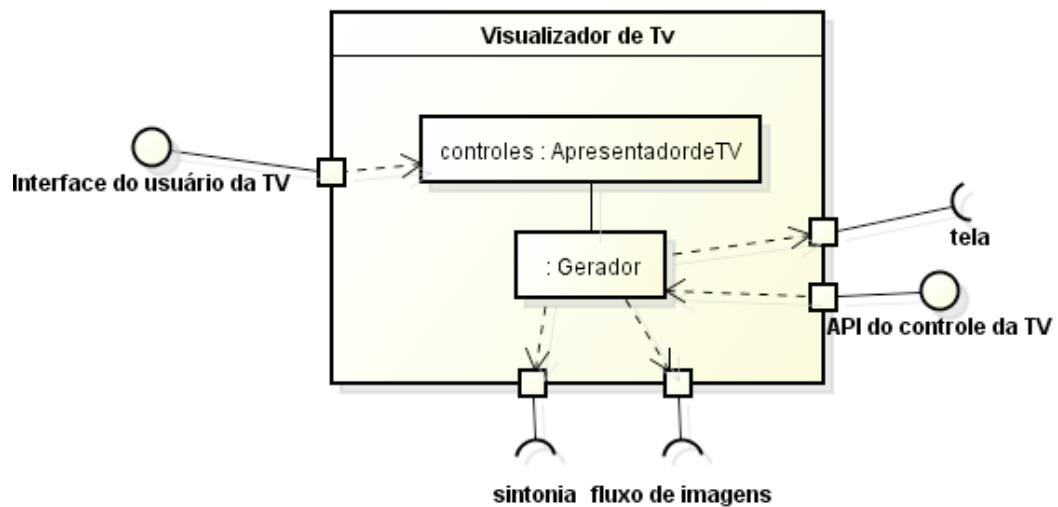
**SAIBA MAIS**

ELEMENTO	DESCRÍÇÃO	REPRESENTAÇÃO
<b>Conector</b>	É uma conexão entre duas partes de modo a associá-las.	
<b>Interface</b>	Entidade já apresentada no capítulo do diagrama de classes (Unidade 3 – Capítulo 2).	
<b>Interface Requerida</b>	Entidade já apresentada em diagrama de componentes deste capítulo.	
<b>Interface Fornecida</b>	Entidade já apresentada em diagrama de componentes deste capítulo.	

Todos os relacionamentos do diagrama de classes podem ser utilizados no diagrama de estrutura: Generalização/especialização, Dependência, Associação, Agregação e Composição.

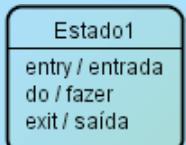
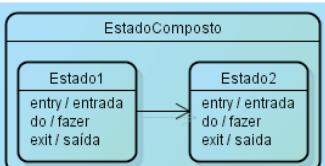
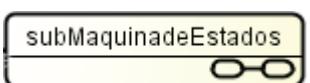
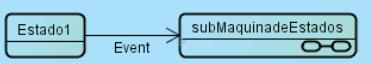
## Ilustração do diagrama de estrutura composta

O diagrama de Estrutura composta a seguir mostra um visualizador de TV, suas partes e interações.



## Diagrama de máquina de estados

O diagrama de máquina de estados é um diagrama comportamental semelhante ao diagrama de atividades. Porém uma máquina de estados representa o comportamento de um objeto através de uma sequência de estados pelos quais o objeto passa durante o tempo de vida, em resposta a eventos.

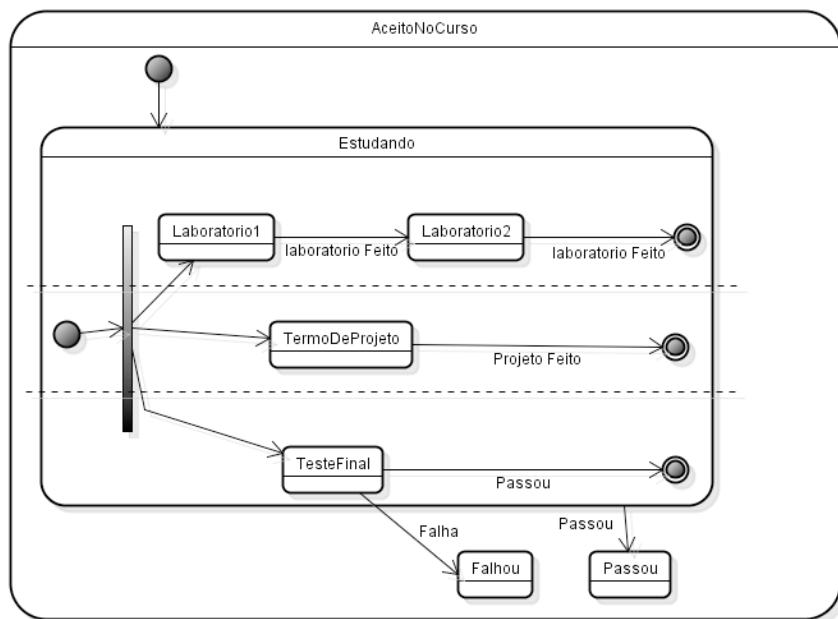
ELEMENTO	Descrição	REPRESENTAÇÃO
<b>Estado</b>	É uma situação de um objeto em determinado momento do seu ciclo de vida. Este pode representar as ações de entrada (entry), saída (exit) e o que é feito (do), relacionado a este estado. Estados podem ser compostos de outros estados.	 
<b>Sub máquina de estados</b>	Mostra outro diagrama de máquina de estados existente e relaciona-o com o diagrama atual.	
<b>Estado inicial</b>	Entidade já apresentada no capítulo do diagrama de atividades (Unidade 4 – Capítulo 2).	
<b>Estado final</b>	Entidade já apresentada no capítulo do diagrama de atividades (Unidade 4 – Capítulo 2).	
<b>Junção de estados</b>	Podem ter várias transições de entrada, transições de saída e testes relacionados. Semelhante à entidade de Escolha de pseudoestados. A representação da junção de pseudoestados é semelhante à representação de estado inicial, mas o círculo da junção tem um raio menor.	
<b>Escolha de estados</b>	Entidade já apresentada no capítulo do diagrama de atividades (Unidade 4 – Capítulo 2).	
<b>Bifurcação (Fork)</b>	Entidade já apresentada no capítulo do diagrama de atividades (Unidade 4 – Capítulo 2).	
<b>Junção (Join)</b>	Entidade já apresentada no capítulo do diagrama de atividades (Unidade 4 – Capítulo 2).	
<b>Transição</b>	Relacionamento entre dois estados que indica a mudança de um estado para o outro quando um evento específico ocorrer.	 

# ! ATENÇÃO

Este diagrama foi criado com base no material do professor Ricardo R. Gudwin, disponível em <http://www.dca.fee.unicamp.br/~gudwin/>

## Ilustração do diagrama de máquina de estados

O exemplo a seguir ilustra o diagrama de máquina de estados no contexto de um sistema de curso, onde o aluno deve passar por laboratórios, termo de projeto para passar ou teste final, onde pode passar ou sair reprovado.



## Diagrama de comunicação ou colaboração

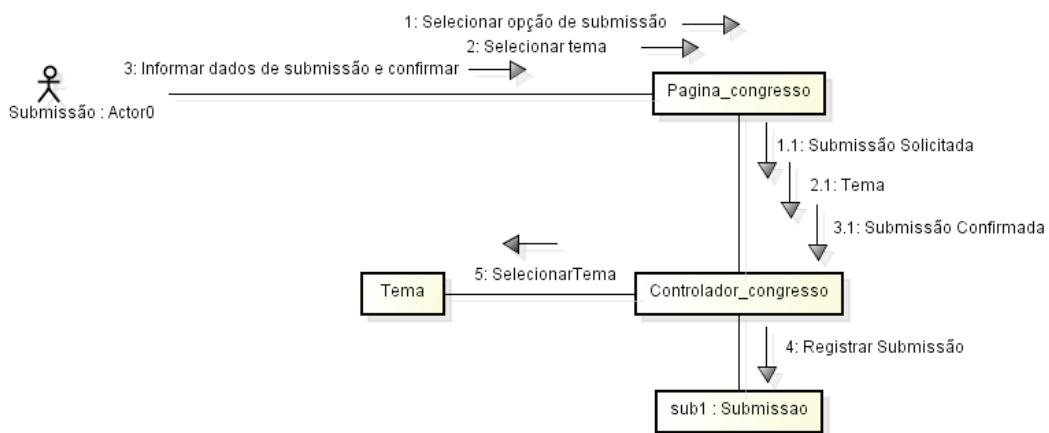
Este diagrama forma juntamente com o diagrama de sequência, os diagramas de interação. Assim sendo, muitas semelhanças aparecem entre eles, assim como as semelhanças entre diagrama de atividades e o diagrama de máquinas de estados.

O diagrama de comunicação (ou colaboração) é um diagrama de interação que mostra o relacionamento entre as instâncias e é melhor para o entendimento de todos os efeitos sobre uma determinada instância, bem como para um design procedural.

ELEMENTO	DESCRIÇÃO	REPRESENTAÇÃO
<b>Autor</b>	Entidade já apresentada no capítulo do diagrama de casos de uso (Capítulo 3 – Tópico 1).	ator1 : Actor1
<b>Classe</b>	Entidade já apresentada no capítulo do diagrama de Classes (Capítulo 3 – Tópico 2).	f1 : Funcionario
<b>Objeto</b>	Entidade já apresentada no capítulo do diagrama de Classes (Capítulo 3 – Tópico 2).	f1 : Funcionario
<b>Interface</b>	Entidade já apresentada no capítulo do diagrama de classes (Capítulo 3 – Tópico 2).	Interface1 ou <<interface>> Interface1
<b>Link</b>	É uma conexão entre dois objetos de modo a associá-los.	ator1 : Actor1 —>  Linha da vida1 Linha da vida2

## Ilustração do diagrama de comunicação

O exemplo a seguir ilustra o diagrama de comunicação no contexto de um sistema de submissão de artigos. As mensagens trocadas entre o Autor da Submissão, a página do congresso, o controlador do congresso, o tema e a Submissão são detalhados a seguir.



Os diagramas estruturais Diagramas Estruturais de Objetos e Pacotes e os Diagramas de Interação Tempo e Interatividade não serão detalhados neste livro. Estes diagramas serão abordados na primeira questão das Questões de Avaliação deste capítulo. As informações sobre estes diagramas podem ser encontradas nas referências da bibliografia presente no final deste capítulo.

## ATIVIDADES DE AVALIAÇÃO

1. Consulte sobre os demais diagramas de UML não abordados neste livro.
2. Quais diagramas de UML você utilizaria para projetar um sistema de controle de aviação?
3. Elabore uma rastreabilidade entre os elementos (Entidades e Relacionamentos) e diagramas apresentados neste livro. Quais elementos se repetem em diferentes diagramas?
4. Escolha um dos diagramas deste capítulo e modele o sistema odontológico que vem sendo trabalhado ao longo dos capítulos deste livro. Utilize como base as classes definidas na questão 2 do Tópico 1 (Capítulo 3), o diagrama de sequência do Tópico 1 (Capítulo 4) e o diagrama de atividades da questão 3 do Tópico 2 (Capítulo 4).



## ! ATENÇÃO

Granularidade diz respeito ao nível de decomposição que um sistema deve ter. Se dividirmos muito as classes, a granularidade do sistema é alta, se dividirmos pouco as classes a granularidade é baixa. Uma analogia pode ser criada em relação à divisão de um terreno que mede 5m x 5m. Se o terreno for dividido tendo por base cada centímetro quadrado com unidade, a granularidade desta divisão é alta. Mas se o mesmo terreno for dividido levando em consideração cada metro quadrado, a granularidade será baixa.

## 2. Padrões de projeto

Projetar um sistema orientado a objeto não é simples, mas realizar este projeto levando em consideração o reuso é ainda mais complexo. Você deve estabelecer as classes pertinentes, com a **granularidade** necessária e relacioná-las da melhor maneira possível.

Uma das maneiras de facilitar esta tarefa é realizar o reuso de boas soluções propostas para problemas recorrentes do desenvolvimento de software. Este cenário de reuso é descrito através de padrões de projeto. Vale frisar que padrões promovem o reuso das ideias de como solucionar o problema, não de código.

Padrões de projeto apresentam as seguintes vantagens e desvantagens:

- **Vantagens:**

- Facilidade de repassar conhecimento entre os desenvolvedores experientes;
- Capturar experiências, tornando-as acessíveis aos novatos;
- Tornar mais rápido o entendimento de sistemas (documentados por meio de padrões);
- Facilitar a evolução do código – quando padrões são usados no desenvolvimento, fica mais fácil entender o código para evoluí-lo;
- Modularidade – código que usa padrões possui maior coesão e, portanto, módulos mais bem definidos, com consequente diminuição da complexidade por quebrar o problema em problemas menores;

- **Desvantagens:**

- Menor Eficiência: Em alguns casos, o uso de um padrão de software pode exigir que seja adicionadas novas classes ou novas camadas de aplicação ao modelo original do sistema, o que pode causar uma sobrecarga na sua execução, tornando-o mais lento;
- Menor Legibilidade: Apesar de serem concebidos para melhorar o entendimento de programas, em certos casos padrões podem diminuir a compreensão de um projeto ou de uma implementação;
- Falta de Motivação da Equipe: Assim como em outras formas de reuso, pode haver resistência ao uso de padrões por parte da equipe.

Para que uma solução seja considerada um padrão, ela deve preencher os seguintes requisitos:

- Deve ser uma solução para um problema em um contexto;
- Você deve ser capaz de dizer ao solucionador do problema o que fazer e como resolver o problema;

- Deve ser maduro, uma solução provada;
- A solução deve ser construída dentro da ótica do solucionador do problema e pode ser implementada milhões de vezes sem se repetir;
- Deve ser capaz de se reproduzir;

Existe uma infinidade de padrões de projeto propostos na literatura, dentre eles podemos citar: Padrões GoF, Padrões POSA e Padrões J2EE.

Dentre eles destacamos os padrões GoF (No Tópico 3 do Capítulo 2 aparece a explicação desta nomenclatura). Quanto ao propósito, eles podem classificar-se em:

- **Padrões de Criação** - Diz respeito ao processo ou ciclo de criação de um objeto.
- **Estruturais** - Diz respeito à composição de objetos e classes.
- **Comportamentais** - Caracteriza o modo como classes e objetos interagem e compartilham responsabilidades.

A organização dos padrões GoF é dada de acordo com a tabela a seguir:

		Propósito		
Escopo	Classe	Criação	Estrutural	Comportamental
Factory Method	Adapter (class)	Interpreter Template Method		
Abstract Factory Buider Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor		

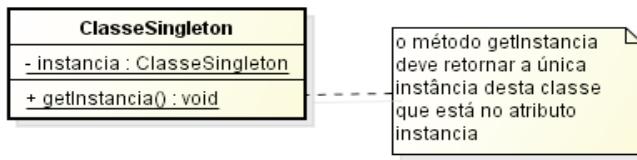
A seguir um padrão de cada propósito será detalhado.

## Singleton

Singleton é um exemplo de padrão GoF criacional que permite que uma classe possibilite apenas uma instância e provê um ponto global de acesso a esta instância.

- **Motivação:** É importante para algumas classes ter exatamente uma instância. Como nós gostaríamos que a classe fosse? E o acesso a esta instância?
- **Descrição do Problema:** Um atributo de instância criado com tipo base da própria classe possibilita a acessibilidade de uma instância a partir da classe base, mas não elimina a possibilidade de instanciar vários objetos a partir dela.
- **Descrição:** Permite que a própria classe base (que possui uma instância de si mesma) controle sua única instância. A classe pode garantir que nenhuma outra instância pode ser criada e pode fornecer uma maneira de acessar a instância.
- **Consequências:** A classe base deve ser responsável por interceptar solicitações para criar novos objetos.

- **Estrutura da Solução:** A estrutura deste padrão é bastante simples. Uma única classe é utilizada, esta classe possui uma instância de si mesma e um método que devolve esta instância. Tanto o atributo (Instância da classe) quanto o método de retorno são estáticos, isto permite que outra classe possa acessar o método sem instanciar a classe diretamente. Observe que o atributo que armazena a instância da classe é privado, neste modo o método de retorno é capaz de garantir que apenas uma instância seja criada. Em linguagens que disponibilizam o conceito de construtor, deve ser criado um construtor privado sem parâmetros para garantir que outras classes tenham acesso à instância da classe base somente através do método `getInstancia`.



**Exemplo:** O exemplo a seguir utiliza o contexto de um sistema universitário, onde é permitido que somente um reitor exista na instituição em determinado momento. Além disto, a classe UAB, recebe a instância da classe ReitorSingleton.



Em Java, o código destas classes ficaria da seguinte forma:

```

// Classe ReitorSingleton

public class ReitorSingleton {
    private static ReitorSingleton instancia;

    public String nome;

    private ReitorSingleton(){
        System.out.println("OK");
    }

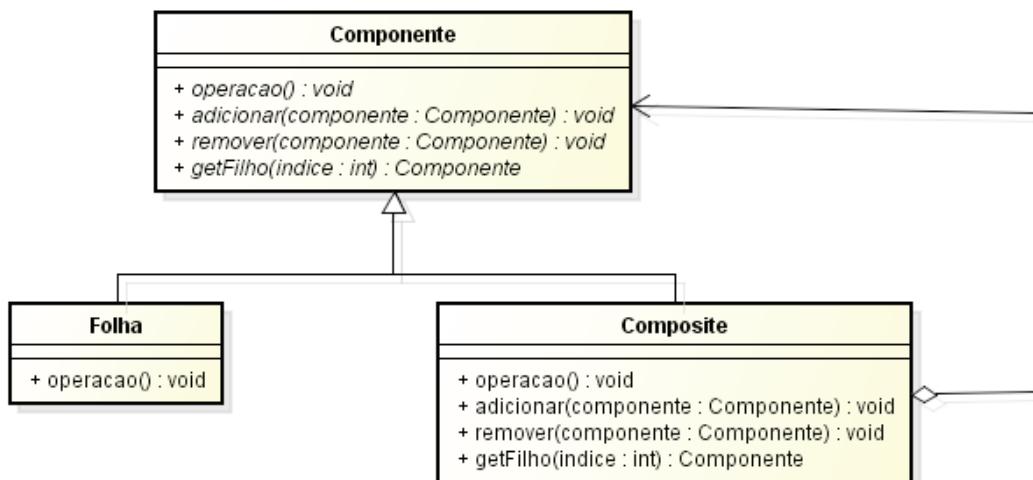
    public static ReitorSingleton getInstancia(){
        if (instancia == null){
            instancia = new ReitorSingleton();
        }
        return instancia;
    }
}

// Classe UAB
public class UAB {
    public static void main(String[] args) {
        ReitorSingleton reitor = ReitorSingleton.getInstance();
    }
}
  
```

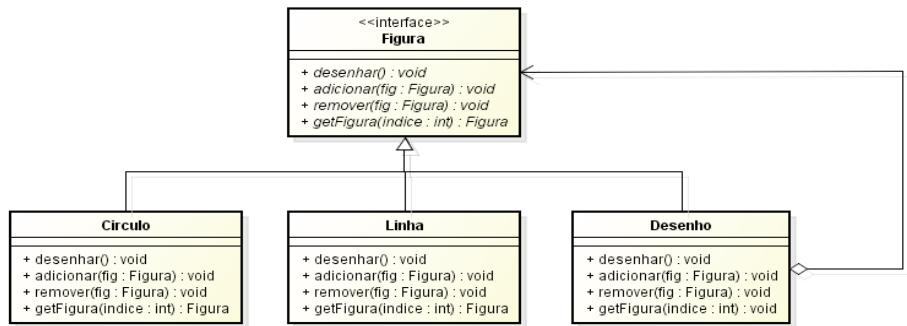
## Composite

Composite é um exemplo de padrão GoF estrutural que compõe objetos em árvores de agregação (relacionamento parte-todo) e permite que objetos agragados sejam tratados como um único objeto.

- **Motivação:** Permitir que as aplicações gráficas como editores de desenho possibilitem que usuários possam construir diagramas complexos a partir de componentes simples.
- **Descrição:** O padrão Composite descreve como usar a composição recursiva para que os clientes não precisem fazer essa distinção.
- **Descrição do Problema:** O utilizador do padrão pode agrupar componentes para formar componentes maiores, que por sua vez podem ser agrupados para formar componentes ainda maiores. Uma implementação simples pode definir classes para componentes gráficos primitivos, tais como texto e linhas mais outras classes que atuam como recipientes para essas primitivas.
- **Consequências:** O código que usa essas classes deve tratar objetos primitivos e recipiente de forma diferente, mesmo que na maioria das vezes o usuário os trata de forma idêntica. Ter que distinguir estes objetos torna a aplicação mais complexa.
- **Estrutura da Solução:** A solução é simples. Várias classes implementam as partes da solução (Folha) e uma classe representa o todo (Composite), adicionalmente uma super-classe ou interface pode ser criada para relacionar as folhas e composite de modo a possibilitar o uso de qualquer folha pelo todo.



**Exemplo:** Um desenho pode ser composto de vários círculos e linhas. Cada um deles é uma figura que pode ser adicionada, removida, desenhada e pode ter seu desenho retornado.



Em Java, o código destas classes ficaria da seguinte forma:

```

// Interface Figura
public interface Figura {
    public abstract void desenhar();
    public void adicionar(Figura fig);
    public void remover(Figura fig);
    public Figura getFigura(int indice);
}

-----
// Classe Círculo

public class Círculo implements Figura{

    public void adicionar(Figura fig) {
        System.out.println("Círculo não tem Filhos.");
    }

    public void desenhar() {
        System.out.println("()");
    }

    public Figura getFigura(int indice) {
        System.out.println("Círculo não tem Filhos.");
        return null;
    }

    public void remover(Figura fig) {
        System.out.println("Círculo não tem Filhos.");
    }
}

-----
//Classe Linha

public class Linha implements Figura{
    public void adicionar(Figura fig) {
        System.out.println("Linha não tem Filhos.");
    }

    public void desenhar() {
        System.out.println("_____");
    }
}
  
```

```

public Figura getFigura(int indice) {
    System.out.println("Linha não tem Filhos, será retornado nulo.");
    return null;
}

public void remover(Figura fig) {
    System.out.println("Linha não tem Filhos.");
}
}

-----
//Classe Desenho
public class Desenho implements Figura {
ArrayList<Figura> filhos = new ArrayList<Figura>();

public void adicionar(Figura fig) {
    this.filhos.add(fig);
}

public void desenhar() {
    Figura f;
    for(int i = 0; i < this.filhos.size(); i++){
        f = this.filhos.get(i);
        f.desenhar();
    }
}

public Figura getFigura(int indice) {
    return (Figura) this.filhos.get(indice);
}

public void remover(Figura fig) {
    this.filhos.remove(fig);
}
}

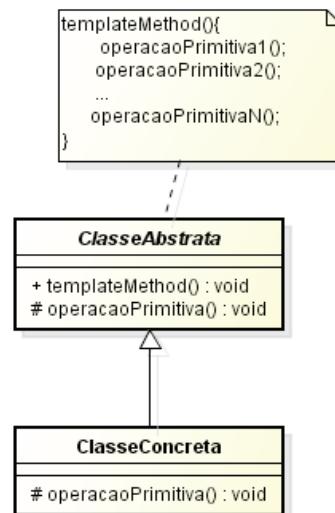
```

## Template Method

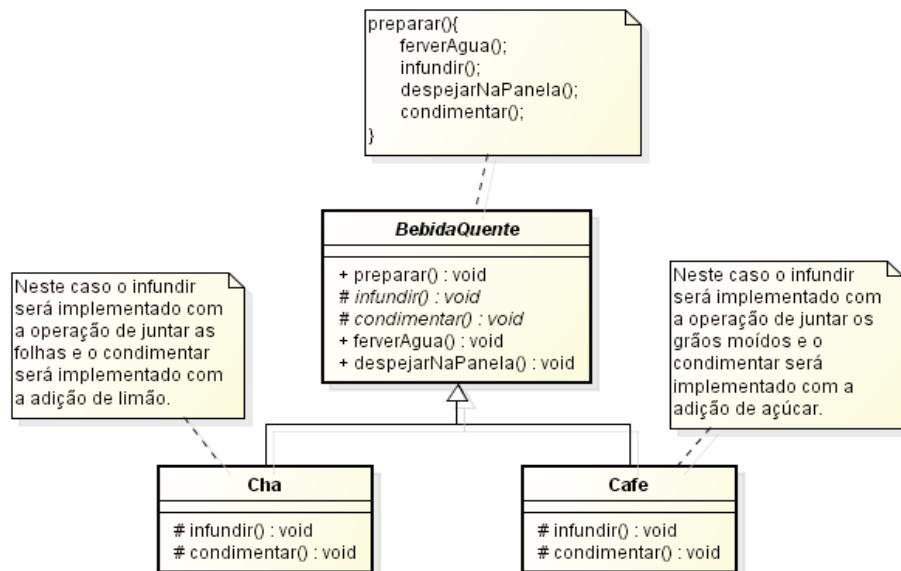
Template Method é um exemplo de padrão GoF comportamental que permite que subclasses componham o algoritmo e tenham a possibilidade de redefinir certos passos a serem tomados no processo, sem contudo alterar sua interface.

- **Motivação:** Você deseja que subclasses implementem versões diferentes, mas com a mesma estrutura de um método da superclasse. O que você faria?
- **Descrição:** Define o esqueleto de um algoritmo em uma operação. O Template Method permite que subclasses componham o algoritmo e tenham a possibilidade de redefinir certos passos a serem tomados no processo, sem contudo, mudá-lo.
- **Descrição do Problema:** Implementar as partes invariantes de um algoritmo uma só vez e as subclasses implementam comportamentos variantes. Quando comportamento comum entre subclasses deve ser fatorado e localizado em uma classe comum para evitar duplicação de código.

- **Consequências:** É importante dizer o que os métodos: devem, podem e não podem ser sobreescritos.
- **Estrutura da Solução:** Uma classe abstrata que possui o método template é definida e utilizada como base para as classes que utilizarão o conjunto de passos do método template e definirão as operações abstratas (cada passo) definidas na classe pai.



**Exemplo:** O exemplo a seguir utiliza a feitura de café e de chá para exemplificar o uso do padrão de Projeto Template Method. Para fazermos um café, é necessário ferver água, infundir os grãos de café, despejar na panela e condimentar com açúcar. Para o preparo do chá, é necessário ferver água, infundir as folhas do chá, despejar na panela e condimentar com limão. Observe que a sequência das operações é semelhante, o que muda são os ingredientes.



Em Java, o código destas classes ficaria da seguinte forma:

```

// Classe BebidaQuente

public abstract class BebidaQuente {
    public void preparar(){
        ferverAgua();
        despejarNaChaleira();
        infundir();
        condimentar();
    }

    public void ferverAgua(){
        System.out.println("Água fervida.");
    }

    public void despejarNaChaleira(){
        System.out.println("Água na chaleira.");
    }

    public abstract void infundir();
    public abstract void condimentar();
}

```

---

```

// Classe Café

public class Café extends BebidaQuente{

    public void condimentar() {
        System.out.println("Açúcar adicionado.");
    }

    public void infundir() {
        System.out.println("Pó de café adicionado.");
    }
}

```

---

//Classe Chá

```

public class Chá extends BebidaQuente{

    public void condimentar() {
        System.out.println("Limão adicionado.");
    }
}

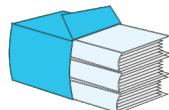
```

```
public void infundir() {  
    System.out.println("Folhas adicionadas.");  
}  
}
```



## ATIVIDADES DE AVALIAÇÃO

1. Qual a importância de padrões de projeto?
2. Consulte outros três padrões de projeto GOF e liste as principais características deles.
3. Consulte mais cenários de aplicação para os padrões apresentados nesta seção.
4. Que padrões você utilizaria para implementar um sistema de uma fábrica de ventiladores que possuem vários tipos de ventiladores, montados a partir de um conjunto específico de peças? Ilustre este cenário.



## SÍNTESE DO CAPÍTULO

Neste capítulo são apresentados em linhas gerais os Diagramas de Componentes, Instalação, Estrutura, Máquina de Estados e Comunicação (ou Colaboração). Através destes diagramas o projetista consegue explicitar vários pormenores que podem ser importantes ao desenvolvimento do sistema. Padrões de projeto são conceituados no Capítulo 2 e alguns padrões GOF são apresentados.



## REFERÊNCIAS

- BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. Elsevier.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: Guia do Usuário**. 2<sup>a</sup> Ed, Editora Campus.
- FOWLER, M. **UML Essencial**. 3<sup>a</sup> Ed, Editora Bookman.
- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1 ed. Addison-Wesley
- UML** - <http://www.uml.org/>



## DADOS DOS AUTORES

### Enyo José Tavares Gonçalves

É Mestre em Ciência da Computação pela Universidade Estadual do Ceará (2009), Bacharel em Ciência da Computação pela Universidade Estadual Vale do Acaraú (2007) e Sun Certified Programmer for the Java 2 Platform 1.4. Tem interesse em pesquisa em Engenharia de Software para Sistemas Multi-Agentes (SMAs), mais especificamente envolvendo os seguintes temas: Modelagem de SMAs, Frameworks de SMAs e Testes de SMAs. Atualmente é Professor Assistente Nível 1 da Universidade Federal do Ceará, onde lidera o Grupo de Engenharia de Software e Sistemas Multi-Agentes (GESMA), adicionalmente co-orienta alunos de mestrado e graduação da Universidade Estadual do Ceará. Atuou profissionalmente como desenvolvedor de sistemas pelo Instituto Atlântico, como professor do curso de Ciências da Computação do Instituto Federal de Educação Ciência e Tecnologia do Ceará e da Universidade Estadual Vale do Acaraú.

### Mariela Inés Cortés

É doutora em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2003) e mestre em Sistemas de Computação pelo Instituto Militar de Engenharia do Rio de Janeiro (1999). Sua alma mater é a Universidade Nacional de La Plata, Argentina, onde completou os estudos de graduação em Ciências da Computação. Especialista na área de Engenharia de Software, atualmente é professora adjunta na Universidade Estadual do Ceará, vinculada ao Curso de Ciências da Computação e com uma ativa participação no Mestrado Acadêmico. Adicionalmente, coordena o Laboratório de Qualidade e Padrões de Software (LAPAQ) e lidera o Grupo de Engenharia de Software e Sistemas Inteligentes (GESSI).

