UniFil
EDUCAÇÃO A DISTÂNCIA

# Unidade 04

## DIAGRAMAS DE SEQUÊNCIA, COMPONENTES, TEMPO E VISÃO GERAL

# Introdução da Unidade

**Olá, amigo(a) discente! Seja bem-vindo(a)!**
Nesta unidade, vamos terminar o Diagrama de Sequência, e serão apresentados os restantes dos diagramas da UML. Com exceção do diagrama de componentes que já existia na UML 1.x, os diagramas de tempo, visão geral e estrutura composta foram criados, especificamente, na UML 2.x.

**Objetivos**

- Demonstrar o Diagrama de Sequência no Estudo de Caso de Locação de Veículos;
- Modelar um diagrama de componente;
- Entender a aplicabilidade dos diagramas de tempo, visão geral e estrutura composta;
- Entender os conceitos de portas, interfaces exigidas e fornecidas.

**Conteúdo Programático**

**Aula 01** – Diagrama de Sequência e Diagrama de Componentes.

**Aula 02** – Diagrama de Tempo, Estrutura Composta e Visão Geral.

Quer assistir às videoaulas em seu celular? Basta apontar a câmera para os **QR Codes** distribuídos neste conteúdo.

Caso necessário, instale um aplicativo de leitura QR Code no celular e efetue o login na sua conta Gmail.

# Diagrama de Sequência e Diagrama de Componentes

**Diagrama de Sequência do Sistema de Locação de Veículos**

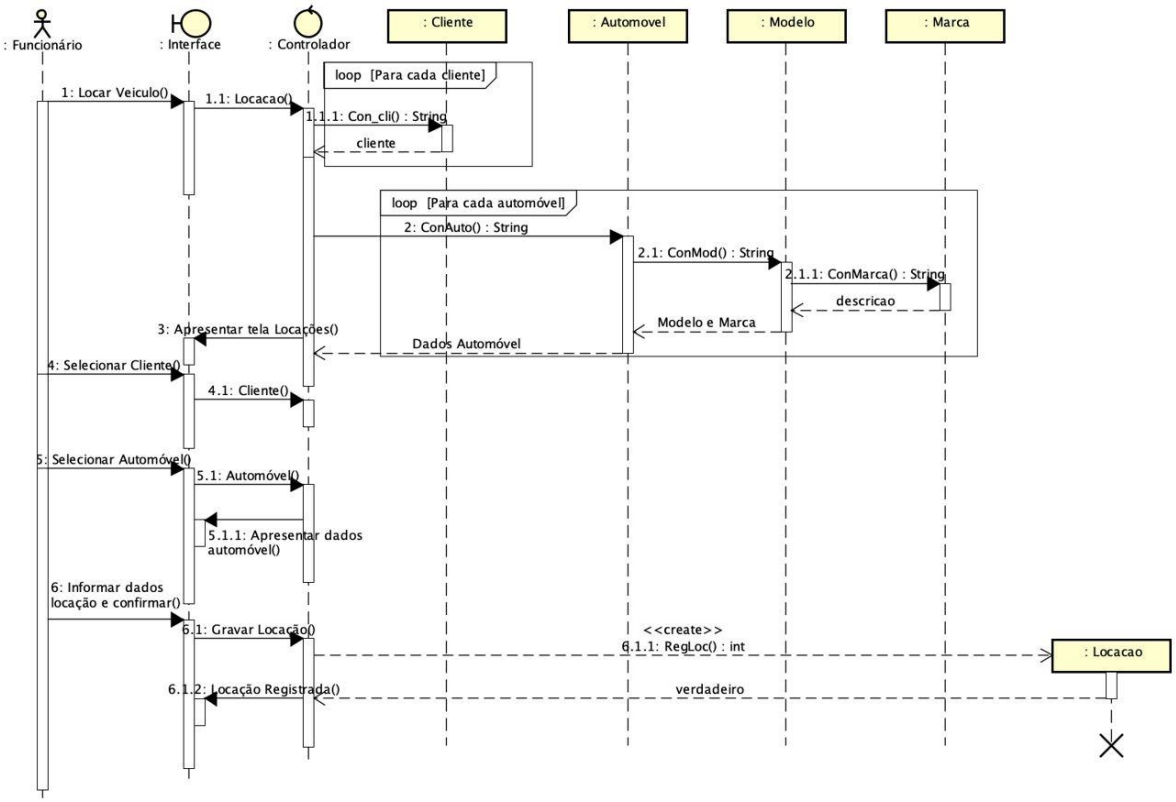
A Figura 01, apresenta o diagrama de sequência do caso de uso *sucLocarAutomóveis.* No processo participa somente o ator **Funcionário**. Neste processo interagem, também, as instâncias da classe de fronteira e de controle, além de objetos das classes de entidade **Cliente, Automovel, Modelo, Marca e Locacao**.

O processo se inicia quando o funcionário seleciona a opção de locar veículo na interface. Esse evento é repassado por ela à controladora, que disparará o método ConCli para consultar cada cliente registrado no sistema, conforme demonstrado, o fragmento combinado do tipo *loop*. Após a execução desse laço, a controladora terá recebido os dados de todos os clientes registrados.

Em seguida, a controladora inicia outro laço para selecionar todos os automóveis disponíveis para locação, disparando o método **ConAuto** para cada automóvel em condições de locação, conforme demonstra o fragmento combinado do tipo método **ConMod** e este, por sua vez, consulta a marca desse modelo por meio do método **ConMarca**. Esses dados são então retornados à controladora, que os manda apresentar na interface juntamente com o formulário de locação de veículos.

O funcionário seleciona o cliente que deseja locar o veículo e em seguida, o veículo desejado. Isso é feito diretamente na interface. A seguir, o funcionário deve informar os dados da locação, como o período de locação, para qual finalidade e para onde o cliente pretende ir. A confirmação da locação na interface fará com que esta avise a controladora da ocorrência desse evento, o que fará com que a controladora dispare o método **RegLoc** para registrar a nova locação, instanciando um novo objeto da classe **Locação** e, caso o método seja executado com sucesso, a controladora mandará a interface informar que a locação foi realizada e que o veículo está liberado.

Figura 01 - Exemplo do Diagrama de Sequência do caso de uso sucLocarAutomóveis




Fonte: Guedes (2009)

O Diagrama de sequência é desenvolvido para as classes existentes. Segue abaixo, os conceitos e notações utilizados para o diagrama de sequência.

**CLIQUE AQUI E CONFIRA**

**Quadro 1 – Conceitos e notações para o diagrama de sequência**

**Videoaula 1**

Utilize o QR Code para assistir!

Agora, assista o vídeo sobre o diagrama de sequência na prática.

**DIAGRAMA DE COMPONENTE**

Na UML 2.0 mudou-se a definição da metaclasse Component. Agora ela é uma subclasse de Class em vez de Classifer. Como uma sublcasse de Classifer, um componente não poderia ter atributos ou métodos próprios.

A finalidade de um componente é explicar os artefatos que são usados para implementar a expressão de projeto lógico nos diagramas Caso de Uso e Classe.

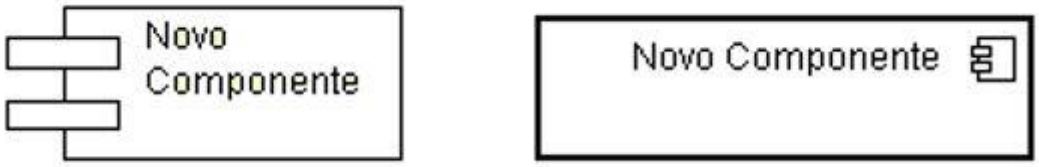
O aninhamento dos elementos dentro de um componente foi definido de modo tanto indistinto por meio de ModelElement. ModelElement, é uma classe básica para praticamente qualquer elemento de diagrama UML. Incluindo associações, classes, objetos, mensagem e outros.

Os componentes permitem representar subsistemas e suas dependências, normalmente organizadas em camadas, mas também os arquivos de código fontes, suas dependências de compilação e os arquivos de aplicação (executáveis, bibliotecas etc.) (LIMA, 2011).

**Criando Componente**

A mudança observável no diagrama de componente é o ícone componente. Na Figura 2, é mostrado o ícone antigo e o novo ícone. A UML optou por usar um retângulo simples com o estereótipo “component” ou como evidenciado abaixo, um ícone retângulo, no qual possui um adorno no canto superior direito do símbolo do componente como uma abreviação para o estereótipo.

Figura 2 - Diferença na UML 1.x com a UML 2.0

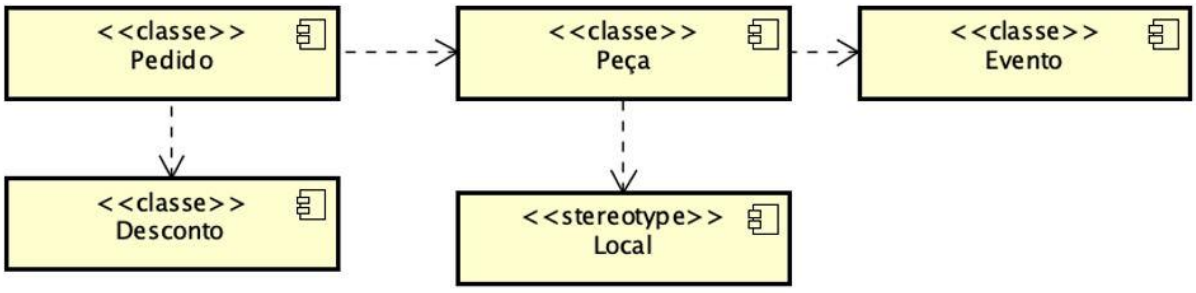



Fonte: o autor (2021)

**Modelando dependências**

No início da modelagem não temos todos os detalhes da interface para os componentes. Às vezes, poderemos ignorar esses detalhes a fim de se concentrar na finalidade de cada componente e seus relacionamentos básicos. Bem, de qualquer forma o foco passa a entender as dependências básicas. Na Figura 3, é apresentado um conjunto de componentes em um nível de visão geral. A interface não aparece no diagrama. E sim, somente dependências.

Figura 3 - Dependências de componentes




Fonte: Guedes (2009)

**Estereótipos no Diagrama de Componentes**

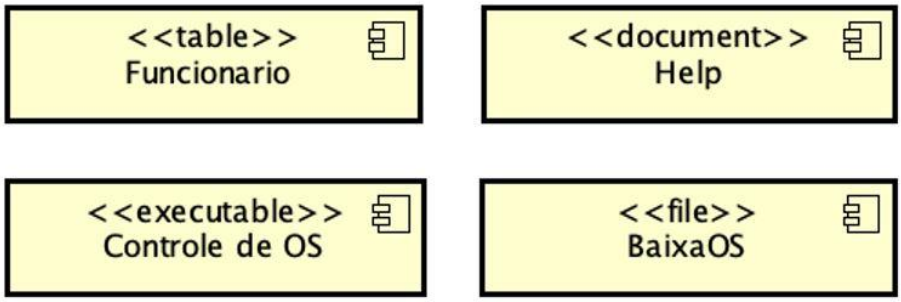
A UML prevê vários estereótipos para o Diagrama de Componentes, porém nada impede que o usuário crie outros conforme a necessidade exige.

Segue abaixo alguns estereótipos pré-definidos:

- **Arquivo (File)**: refere-se a qualquer arquivo que compõe o sistema, como código-fonte.
- **Documento (*Document*)**: refere-se a um arquivo texto utilizado por algum componente do sistema como um arquivo de ajuda.
- **Executável (*Executable*)**: determina que o componente é um arquivo executável, ou seja, que pode executar uma série de instruções. O termo executável é genérico, podendo se referir a módulos compilados em Java, que às vezes não são executáveis no sentido clássico, sendo interpretado pela máquina virtual (GUEDES, 2009).
- **Biblioteca (*library*):** refere-se às bibliotecas fornecidas pela própria linguagem de programação ou até mesmo criadas durante o processo de desenvolvimento.
- **Tabela (*table*):** refere-se a tabela do banco de dados, local físico onde os dados são armazenados.

A Figura 4 demonstra alguns exemplos de estereótipos utilizados nos componentes.

Figura 4 - Exemplo de estereótipos




Fonte: o autor (2021)

**Modelando a realização de componente**

A UML 2.0 inclui um conceito de realização. A realização se refere a qualquer implementação de um requisito. Componente é uma abstração, uma representação de um requisito para o *software* físico. A realização é uma entidade concreta que transforma o requisito em uma realidade ou realiza o requisito. Um componente pode estar associado a qualquer quantidade de realizações.

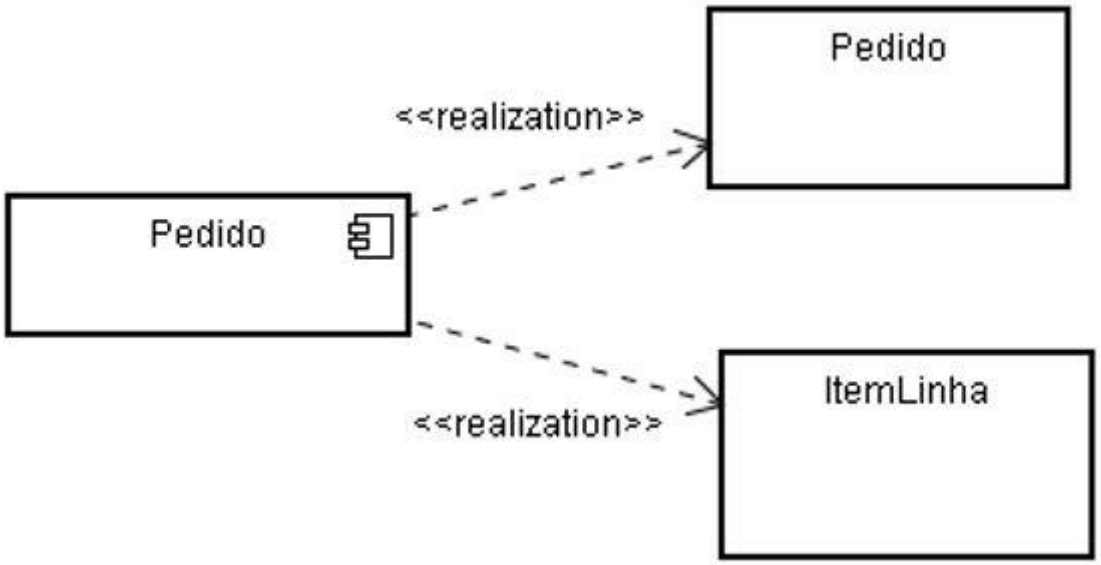
A realização pode se referir a um classificador. O classificador é uma superclasse que descreve qualquer coisa que representa um conjunto de instâncias e pode conter recursos. Eles incluem classes, interfaces, comportamentos e tipos de dados e não estão limitados.

A especificação UML oferece dois estereótipos padrão para o componente. O "*realization*" e o "*specification*" que definem se o componente é usado como uma especificação pura ou como uma implementação.

Um componente “*realization*” define um ou mais conjuntos de classificadores de implementação. Já o componente “*specification*” se refere a um ou mais classificadores de especificação.

Na Figura 5, a realização para o componente pedidos são classes Pedido e ItemLinha. O componente pedido é implementado pela instanciação das classes de pedido e item de linha.

Figura 5 - Modelando realização por dependência




Fonte: o autor (2021)

**Videoaula 2**

Utilize o QR Code para assistir!

Agora, assista o vídeo sobre o diagrama de componentes.

**Modelando interface**

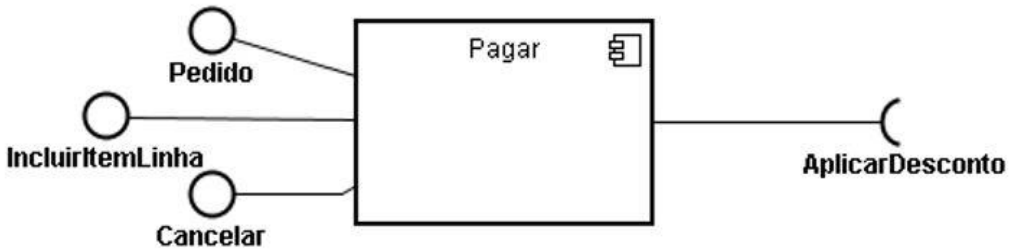
Modelar interface de componente é um recurso fundamental. É a capacidade de definir interfaces. É preciso expor alguns meios para que os componentes se comuniquem com outros componentes. A UML 2.0 aplica os conceitos de interface e porta componentes e as classes.

As interfaces podem ser de dois tipos, as exigidas e fornecidas. As exigidas, os componentes podem definir componentes. Por exemplo, o componente pedido pode oferecer capacidade de incluir itens de linha ao pedido, cancelar o pedido e pagar pelo pedido. O início do projeto do componente define a implementação dos comportamentos. A interface define como outro componente precisa pedir acesso a um serviço fornecido.

A fornecida é quando o componente precisa da ajuda de outros componentes e precisa definir exatamente o que ele precisa. Por exemplo, o pedido precisa ser capaz de descobrir se um desconto se aplica aos itens comprados no pedido. Mas em si o pedido não tem recursos para responder a essa pergunta e por isso ele define uma interface indicando o tipo de ajuda exigida.

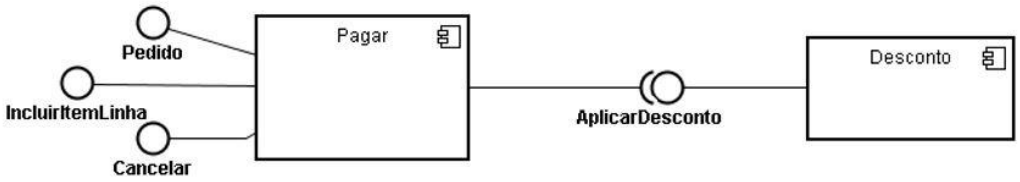
Na Figura 6 e 7, ilustram o componente pedido com interfaces fornecidas, à esquerda para acrescentar itens de linha ao pedido, cancelar o pedido e pagar pelo pedido. Uma interface exigida à direita para aplicar o desconto.

Figura 6 - Interface para interface fornecida e exigida




Fonte: o autor (2021)

Figura 7 - Conectando interfaces exigida e fornecida




Fonte: o autor (2021)

A UML 2.0 admite o uso de classes para descrever as interfaces tendo em vista que as interfaces podem ser mais complicadas do que os exemplos. O uso de classe permite acrescentar as assinaturas de interface explicitamente no diagrama.

A UML 2.0 oferece um meio de representar toda a informação definida até aqui para um componente, incluindo interfaces, realizações e artefatos. Isso é possível, pois o componente agora é um tipo de classe. Uma classe pode ter compartimentos e recursos.

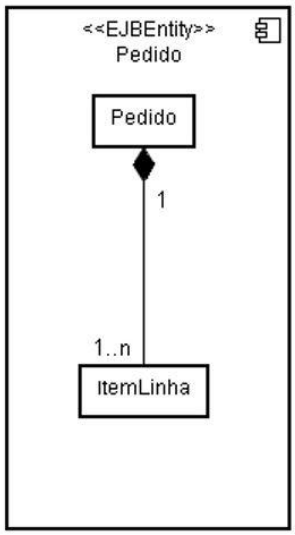
**Realização usando conectores e portas**

A UML 2.0 define o relacionamento entre componentes e realizações como composição. Composição é o tempo de vida dos membros, as realizações, neste caso controlado pelo composto, o componente. As realizações são encapsuladas dentro do componente e ele o controla quando elas são instanciadas.

As interfaces fornecidas, interface exigida realizações e artefato, é chamada de caixa branca, pois podemos ver por meio da caixa suas propriedades internas. A caixa branca ajuda a identificar as realizações encapsuladas. Mas os relacionamentos entre si ou com o componente ela não ajuda a explicar. Para invertermos essa situação, a definição estendida PackagingComponent para um ícone de componente permite que esse componente se abra

para revelar as realizações contidas. A Figura 8 mostra os relacionamentos usando uma visão de contenção.

Figura 8 - Componente contendo realizações

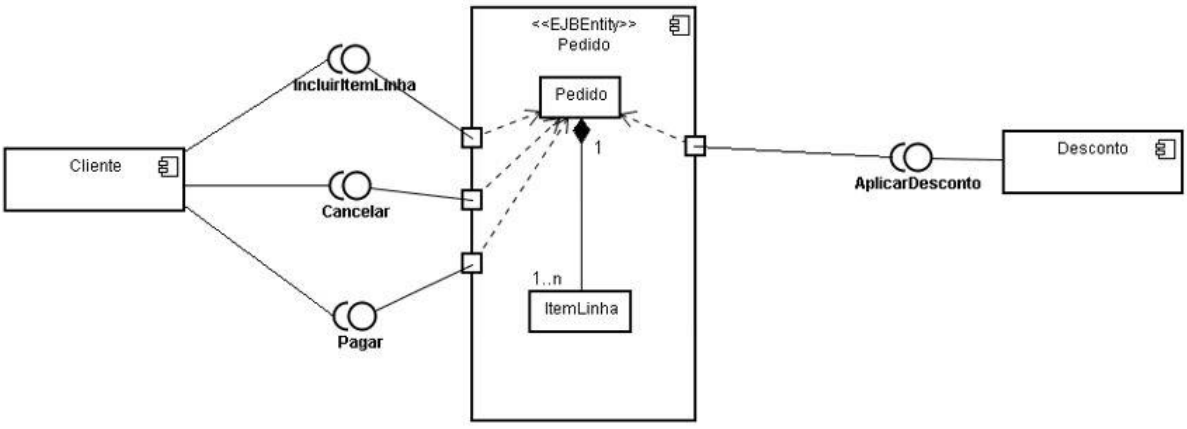



Fonte: o autor (2021)

Na UML 1.x é usada essa representação. Na UML 2.0 acrescenta a capacidade de mapear as interfaces dos componentes e as realizações que lhes dão suporte. Ao fazer isso, o componente precisa de um lugar para fazer a transição entre suas interfaces e suas realizações internas que é uma porta.

Uma porta aparece com um pequeno quadrado na borda do componente. A interface de componente é anexada a uma porta. As dependências estendem-se da porta as realizações que satisfazem os requisitos das interfaces. A Figura 9, modela o pedido e suas interfaces.

Figura 9 - Modelando portas entre interfaces externas e realizações internas




Fonte: o autor (2021)

Um conector de delegação mapeia uma solicitação de uma porta a outra porta ou a uma realização que oferece a implementação para a solicitação. A conexão de porta a porta efetivamente redireciona a invocação. A única restrição é que as portas precisam ser do mesmo tipo. Ambas precisam admitir as interfaces exigidas ou as fornecidas.

Na conexão de porta realização ela possui um relacionamento "*implements*" com a especificação da interface. A interface externa combina com uma assinatura na realização.

Um conector de montagem é usado para mapear a interface exigida a uma interface fornecida. O conector de montagem une as duas interfaces para formar uma conexão do tipo bola e encaixe. Uma espécie de saudação entre os componentes.

Na UML 2.0, ela oferece uma notação abreviada para o conector montagem. Os conectores podem ser mesclados caso os componentes exigirem ou fornecerem as mesmas interfaces.

**Videoaula 3**

Utilize o QR Code para assistir!

Agora, assista o vídeo sobre o diagrama de componentes.

Leitura Obrigatória

Leia o capítulo 12 do livro do Medeiros. *Desenvolvendo Software com UML 2.0*. Acesse o livro no *link* abaixo:

Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/2921/pdf/0. Acesso em: 5 set. 2021.

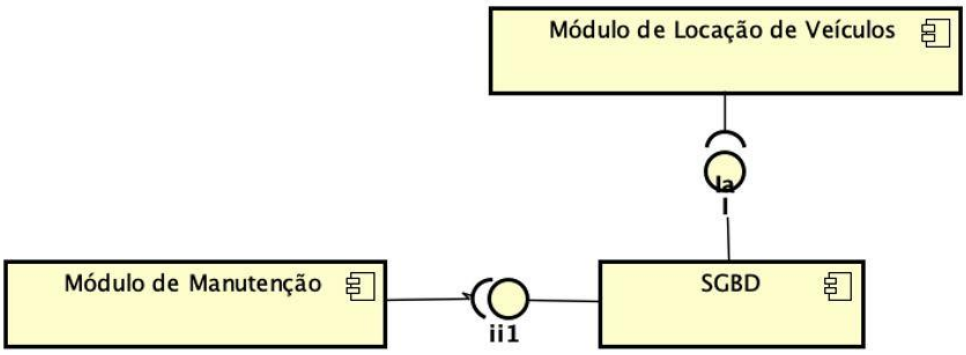
# Diagrama de Tempo, Estrutura Composta e Visão Geral

**Diagrama de Componentes do Sistema de Locação de Veículos**

Este diagrama, representado pela Figura 10 apresenta os seguintes componentes: (GUEDES, 2009)

- **Módulo de Manutenção:** Esse módulo é responsável pela manutenção e cadastros do sistema
- **Módulo de Locação de Veículos:** Esse módulo possui a função de gerenciar as locações de veículos da empresa, bem como suas devoluções.
- **SGBD:** Esse componente representa o sistema gerenciador de banco de dados responsável por manter as informações do sistema. Observe que os dois módulos anteriores apresentam interfaces requeridas com esse componente, uma vez que solicitam operações de gravação e recuperação de dados no SGBD.

Figura 10 - Diagrama de Componentes




Fonte: Guedes (2009)

**Videoaula 1**

Utilize o QR Code para assistir!

Agora, assista o vídeo sobre o Diagrama de Componentes.

**Diagrama de Tempo**

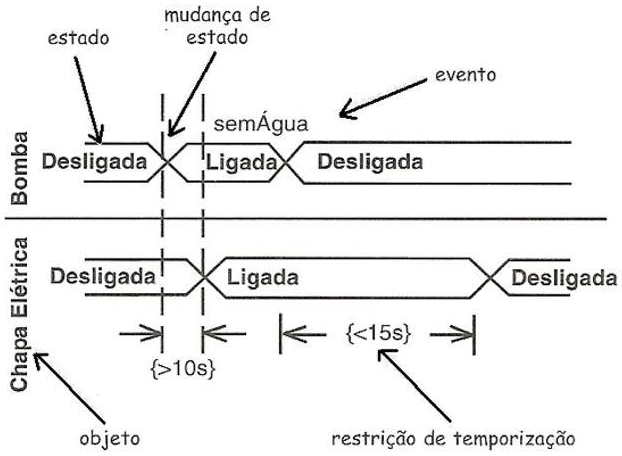
Os diagramas de tempo demonstram as mudanças de estado de um objeto ao longo do tempo em resposta aos eventos externos. De muita utilidade para modelar sistemas de tempo real e pouca utilidade para modelar aplicações comerciais. Os diagramas de interação formam um

quarteto de: sequência, comunicação, visão geral e tempo. Os diagramas de sequência e comunicação já existiam na UML 1.0, ao contrário do diagrama de tempo que fora criado na UML 2.0.

O diagrama de tempo é utilizado por engenheiros elétricos no uso da demonstração do estado de máquinas digitais. Pode ser utilizado em programação de máquinas de controle numérico, na indústria, tipo CNC.

A proposta primária é apresentar a evolução no tempo ao longo de um eixo com suas variações (alterações) e de forma linear. O foco da atenção se dá no tempo da ocorrência dos eventos que causam as mudanças em estados. Resumindo, esse diagrama explora o comportamento de um ou mais objetos em um dado período de tempo, mostrando suas alterações de estado. A Figura 11 apresenta um exemplo de um diagrama de tempo modelando uma chapa elétrica e uma bomba.

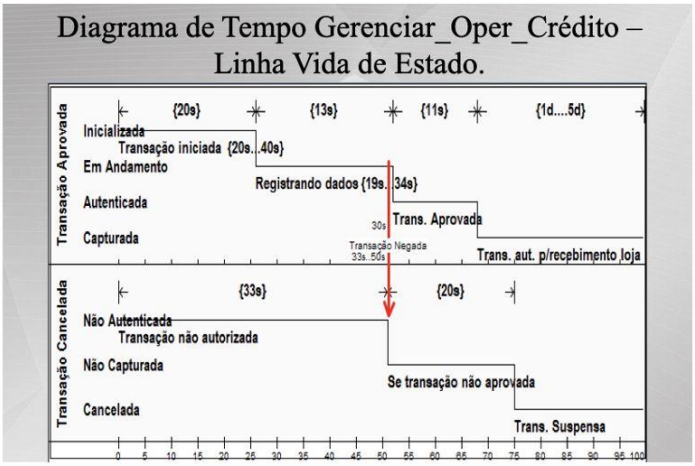
Figura 11 - Modelagem de uma chapa elétrica e uma bomba




Fonte: Guedes (2009)

A Figura 12 apresenta um exemplo de um diagrama de tempo em uma modelagem de mais alto nível para um sistema de gerenciador de operações de crédito, pouco utilizado, comercialmente.

Figura 12 - Modelagem de uma chapa elétrica e uma bomba




Fonte: Guedes (2009)

**Quando criar um diagrama de tempo?**

Caso tenha mais facilidade de entendimento sobre o que está acontecendo com determinado objeto durante um determinado tempo, utilizando esse diagrama (em conjunto ou não ao diagrama de estado e/ou sequência), esse diagrama pode ser usado.

**Videoaula 2**

Utilize o QR Code para assistir!

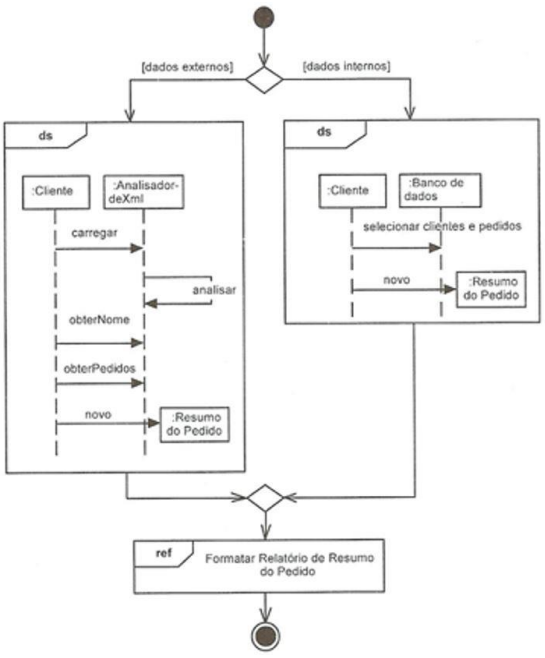
Agora, assista o vídeo sobre o Diagrama de Tempo.

**Diagrama de Visão Geral**

O diagrama de visão geral, é um diagrama novo na UML 2.0, é uma variação do diagrama de atividade. Seu objetivo é fornecer uma visão geral do controle de fluxo, esse diagrama permite a captura de uma perspectiva estática dos fluxos de interação entre os componentes do sistema de forma geral.

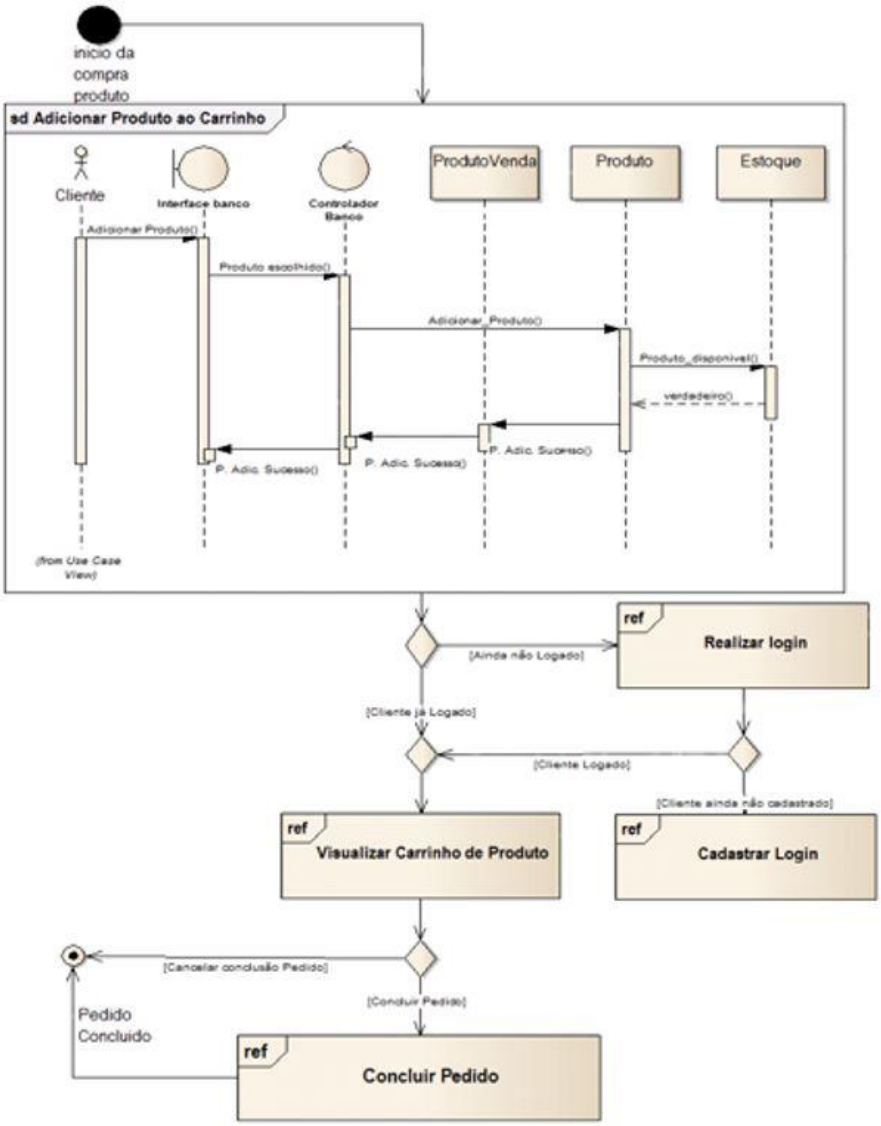
Este diagrama congrega uma determinada visão que pode englobar vários diagramas. Os diagramas que podem estar envolvidos são os diagramas de sequência, comunicação e tempo. Muito eficaz em reuniões e demonstrações de situações complexas, devemos utilizá-lo de forma intensiva, pois não existe outro diagrama que nos dê uma visão tão completa dos passos que uma situação, classe ou objeto possa passar.

Figura 13 - Exemplo de diagrama de Visão Geral




Fonte: Guedes (2009)

Figura 14 - Diagrama de Visão Geral




Fonte: Guedes (2009)

**Videoaula 3**

Utilize o QR Code para assistir!

Agora, assista o vídeo sobre o Diagrama de Visão Geral.

Leitura Obrigatória

Leia o capítulo 15 e 17 do livro do Medeiros. *Desenvolvendo Software com UML 2.0.* Acesse no *link* abaixo*:*

Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/2921/pdf/0. Acesso em: 5 set. 2021.

Atividades Avaliativas e de Fixação

**Prezado(a) aluno(a)!**
Chegou a hora de colocarmos em prática os conhecimentos adquiridos até o momento!
As avaliações nos possibilitam perceber se estamos obtendo bons resultados e quais são os conteúdos que requerem nossa atenção.

Clique em "**Módulos**" no "**Menu Lateral**" e acesse a **Atividade Fixação e Atividade Avaliativa**.

Atente-se aos prazos! Após a data limite, a avaliação será encerrada. Bons estudos!

**Videoaulas**

**Prezado(a) aluno(a)!**
Você também poderá encontrar todas as videoaulas, clicando em "**Módulos**" no "**Menu Lateral**" e acessar a página de vídeos.

**Encerramento**

Nesta unidade, foi abordado a prática do Diagrama de Sequência, novos diagramas da UML como o Diagrama de Tempo e Visão Geral. Espero que tenham gostado do conteúdo e que ele possa agregar ainda mais em seus conhecimentos.

**Bom trabalho!**

**Videoaula Encerramento**

Utilize o QR Code para assistir!

Assista agora ao vídeo de encerramento de nossa disciplina.

**Videoaula Encerramento Disciplina**

Assista agora ao vídeo de encerramento de nossa disciplina.

**Encerramento da Disciplina**

Chegamos ao final da disciplina de Análise e Projeto Orientado a Objetos. Iniciamos com os conceitos básicos sobre a orientação a objetos, UML, requisitos e o processo de desenvolvimento de *software*. Fizemos a modelagem de um estudo de caso de ponta a ponta, sempre, observando a integração e a rastreabilidade entre eles. Além do estudo de caso apresentado, vocês puderam modelar seus respectivos projetos aplicando na prática os

conceitos abordados utilizando uma ferramenta CASE. Espero que todos tenham tido uma ótima aprendizagem e me coloco à disposição para sanar possíveis dúvidas futuras. Um grande abraço a todos!

**Referências**


BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML – Guia do Usuário**. 2. ed. Rio de Janeiro: Editora Campus, 2006.

FOWLER, M. **UML Essencial**. 3. ed. Porto Alegre: Bookman, 2007.

GUEDES, G. T. A. **UML 2.0: uma abordagem prática.** São Paulo: Novatec, 2009.

IBM. **RUP – Rational Unified Process (Software**) Versão 7.0. USA: IBM Rational, 2006.

LARMAN, C. **Utilizando UML e Padrões**. 3. ed. Porto Alegre: Bookman, 2007.

LIMA, A. S. **UML 2.3 Do Requisito a Solução.** São Paulo: Erica, 2011.

MEDEIROS, E. **Desenvolvendo Software com UML 2.0**. São Paulo: Pearson, 2004.

OMG. OBJECT MANAGEMENT GROUP. UML 2.0. Disponível em:www.omg.org. 2020.

PAGE-JONES, M. **Fundamentos do Desenho Orientado a objetos com UML**. São Paulo: Makron books, 2001.

PENDER, T. **UML 2.0 - A Bíblia**. Rio de Janeiro: Editora Campus, 2004.

TANAKA, S. S**. O poder da tecnologia de workflow e dos mapas conceituais no processo de ensino e aprendizagem da UML**. Dissertação de Mestrado, UEL. Londrina, 2011.


UniFil
EDUCAÇÃO A DISTÂNCIA
UNIFIL.BR