
UNIFIED
MODELING
LANGUAGE
TM

O que é?

UML é uma linguagem padrão da OMG para visualização, especificação, construção e documentação de software orientado a objetos.

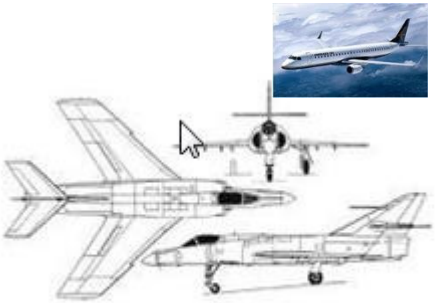
- O ponto importante a observar aqui é que a UML é uma **linguagem** para especificar e não uma metodologia ou procedimento.

- A UML é usada para definir um sistema de software; para detalhar os **artefatos** do sistema, para **documentar** e **construir**.

- É a linguagem que permite que a "planta" do sistema seja criada; permite a criação de **modelos**.

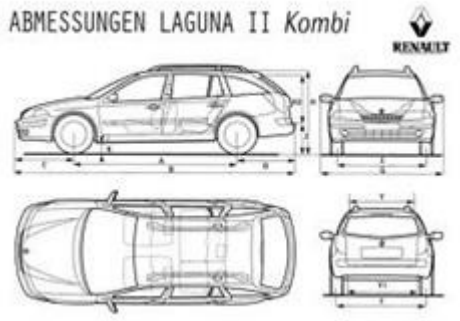
- A modelagem é uma técnica de engenharia aprovada e bem-aceita.

- Construímos modelos de arquitetura de casas e de grandes prédios para auxiliar seus usuários a visualizar qual será o produto final. A modelagem não faz parte apenas da indústria de construção.

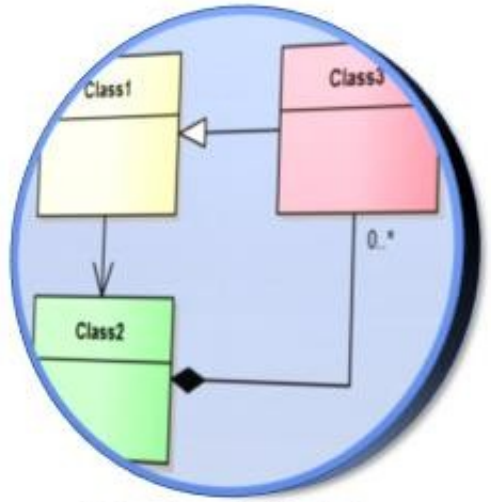


- Seria inconcebível fornecer um novo avião ou automóvel sem primeiro construir os respectivos modelos – desde modelos de computadores, modelos físicos de túneis de vento até protótipos em larga escala.

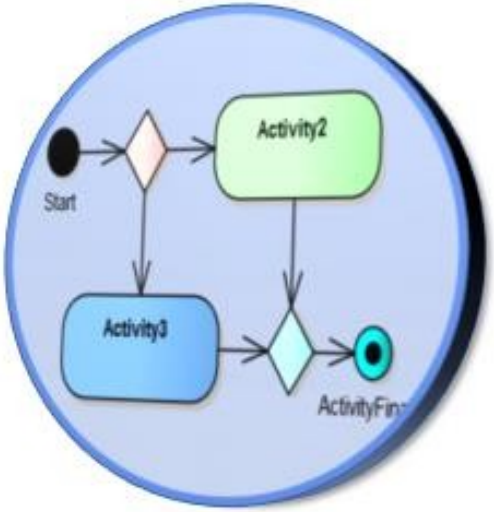
- Novos dispositivos elétricos, desde microprocessadores a sistemas de telefonia, demandam algum grau de modelagem com o propósito de **permitir uma melhor compreensão do sistema e a comunicação dessas ideias a outras pessoas.**



- Um modelo é uma **simplificação da realidade**, que fornece uma cópia do projeto de um sistema.
- Os modelos poderão abranger planos detalhados, ou planos mais gerais com uma visão panorâmica do sistema considerado.
- **Um bom modelo inclui componentes que têm ampla repercussão e omite os componentes que não são relevantes em determinado nível de abstração.**
- Todos os sistemas podem ser descritos sob **diferentes aspectos**, com a utilização de **modelos distintos**, e cada modelo será, portanto, uma abstração semanticamente específica do sistema.
- Os modelos podem ser **estruturais**, dando ênfase à **organização** do sistema, ou podem ser **comportamentais**, dando ênfase à **dinâmica** do sistema.



Structural Models

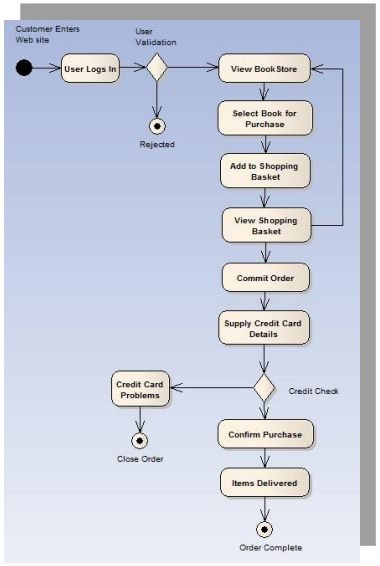


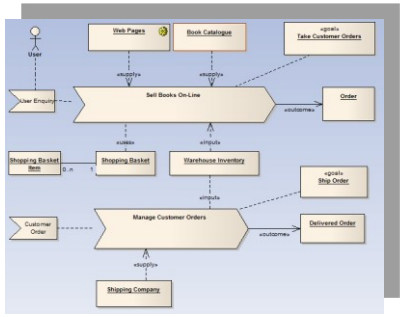
Behavioral Models

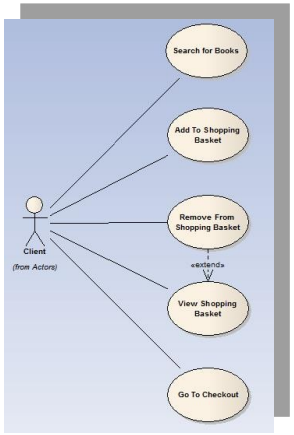
Projetos de software precisam de modelagem?

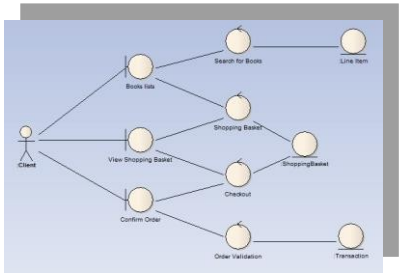
- Projetos de softwares mal sucedidos **falham** em relação a aspectos **únicos** e **específicos** de cada projeto, mas todos os projetos **bem-sucedidos** são **semelhantes** em diversos aspectos.

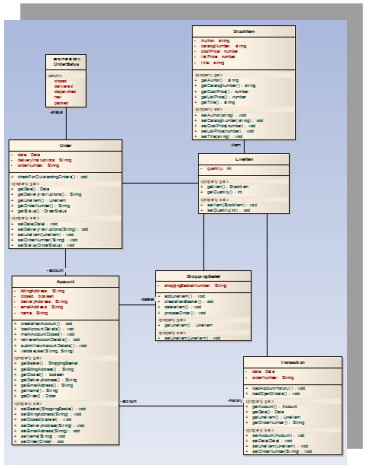
- Existem muitos elementos que contribuem para uma empresa de software de sucesso; um desses componentes é a utilização da modelagem.











Exemplos de diagramas UML

Todos os tipos de projetos de software precisam de modelagem?

- A modelagem não se restringe a grandes sistemas: softwares bem simples poderão receber os benefícios da modelagem. Porém, é absolutamente verdadeiro que, quanto maior e mais complexo for o sistema, maior será a importância da modelagem, por uma razão muito simples:

Construímos modelos de sistemas complexos porque não é possível compreendê-los em sua totalidade

Existem limites para a capacidade humana de compreender complexidades. Com a ajuda da modelagem, delimitamos o problema que estamos estudando, **restringindo nosso foco a um único aspecto por vez.**
Em essência, esse é o procedimento de "dividir-e-conquistar" que Edsger Dijkstra falava há anos: **"Ataque um problema difícil, dividindo-o em vários problemas menores que você pode solucionar."**

Histórico

Método de Booch

Object Modeling Technique - OMT Jacobson

Object Oriented Sw Engineering Rumbaugh

Início dos anos 90

Rational

Unified Modeling Language (UML) *OMG*

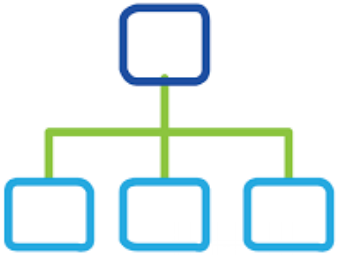
1997

Composição da UML

- Formada por 13 diagramas classificados em diagramas estruturais e comportamentais;

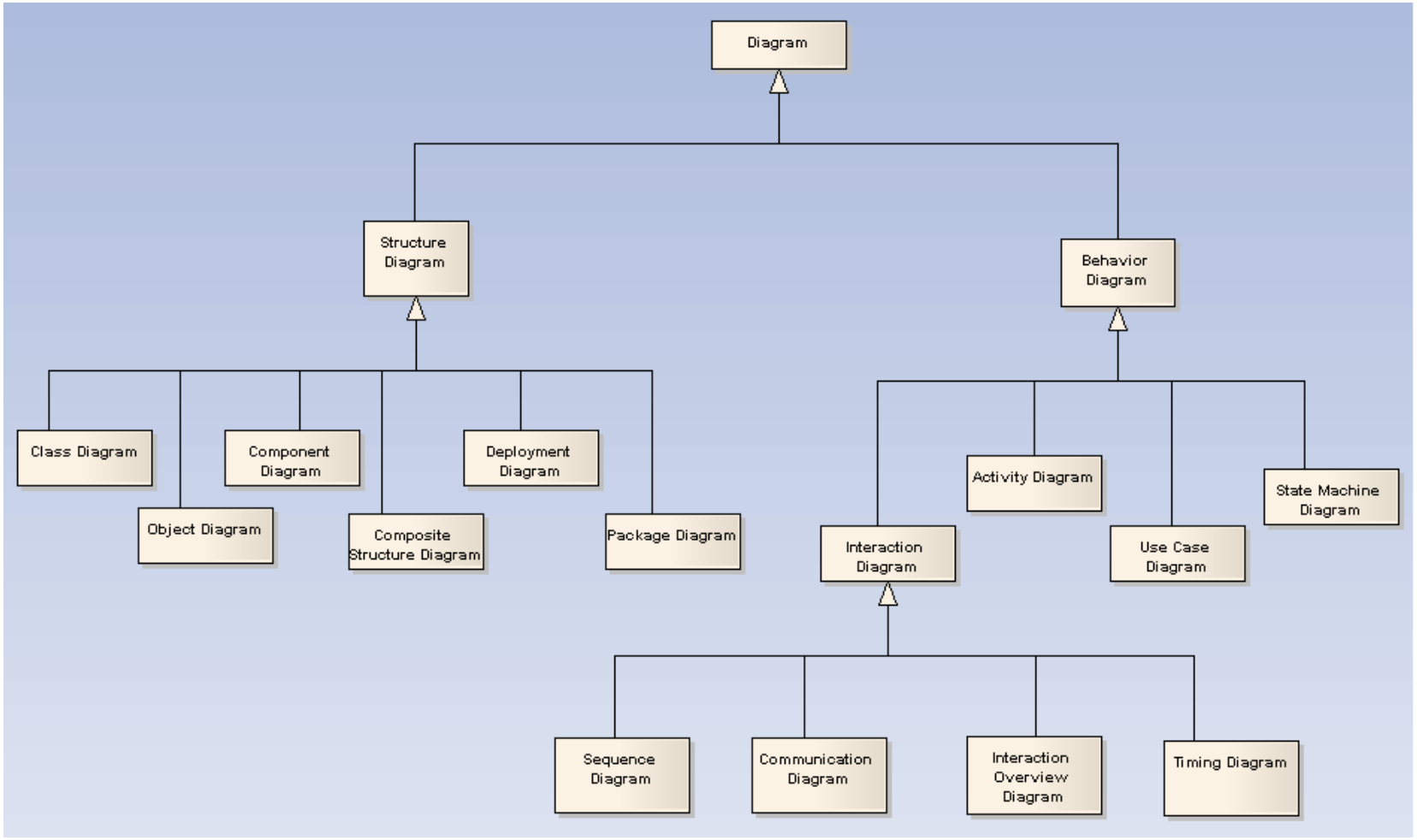
- Cada diagrama apresenta uma visão diferente sobre o sistema:

  – Estruturais - visão estática do modelo



  – Comportamentais - visão dinâmica do modelo

Composição da UML



Fonte: UML Superstructure Specification, v2.1.1, figure A.5, p. 680

Como usar a UML?

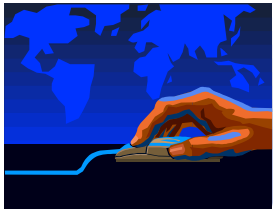
A UML é uma linguagem para especificar e não uma metodologia ou procedimento.
A UML é normalmente usada como parte de um processo de desenvolvimento de software, com o apoio de uma ferramenta CASE, para definir os requisitos, as interações e os elementos do sistema.

A natureza exata do processo depende da metodologia de desenvolvimento utilizada. Como exemplo de processo poderia ter-se algo como o seguinte:

Entender o problema.

Identificar as características que a solução deverá prover – O QUÊ?

Analisar a solução – COMO?



Projetar e Construir a solução

Entregar a solução

Exemplo de uso da UML

Entender o problema.

Identificar as características que a solução deverá prover – O QUÊ?

Analisar a solução – COMO?

Projetar e construir a solução

Entregar a solução

Entendimento do domínio do problema	Especificação: Requisitos, Use Cases	Análise	Projeto	Entrega

Exemplo de uso da UML

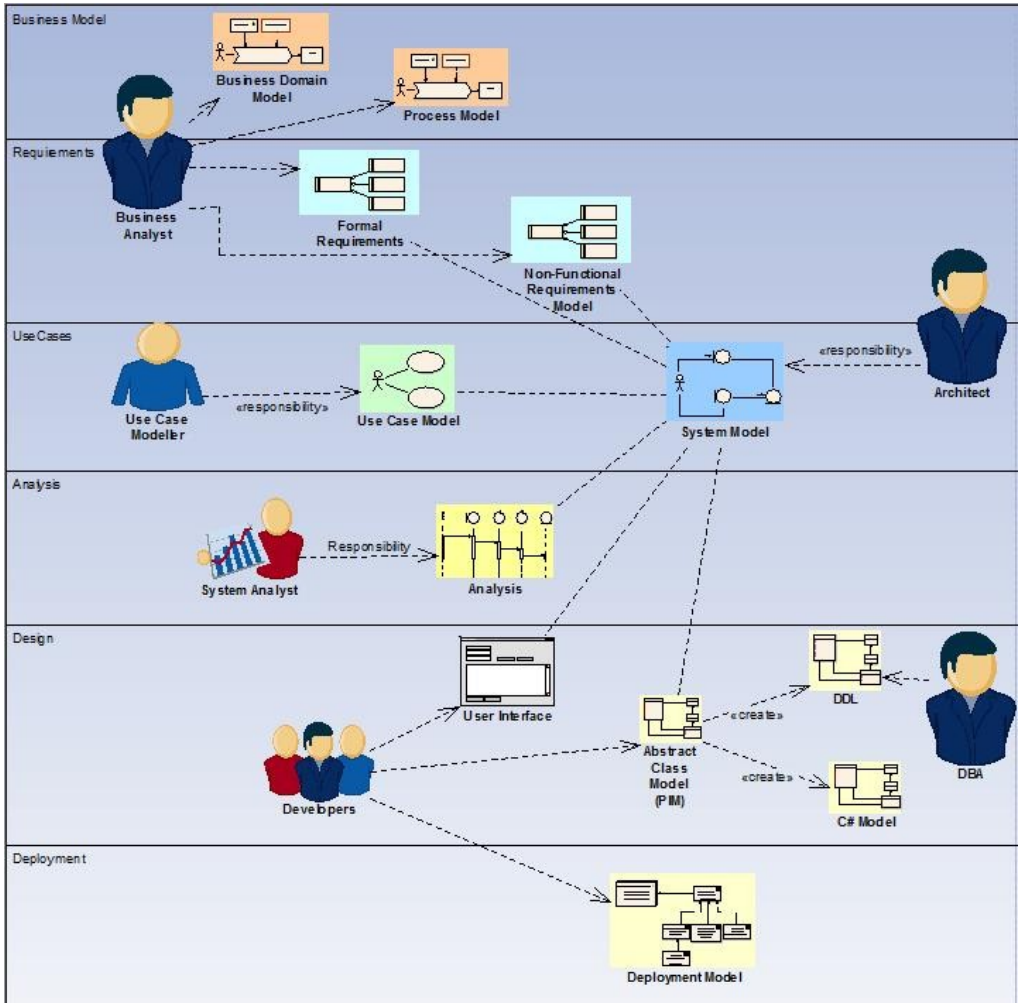
Entendimento do domínio do problema

Especificação

Análise

Projeto

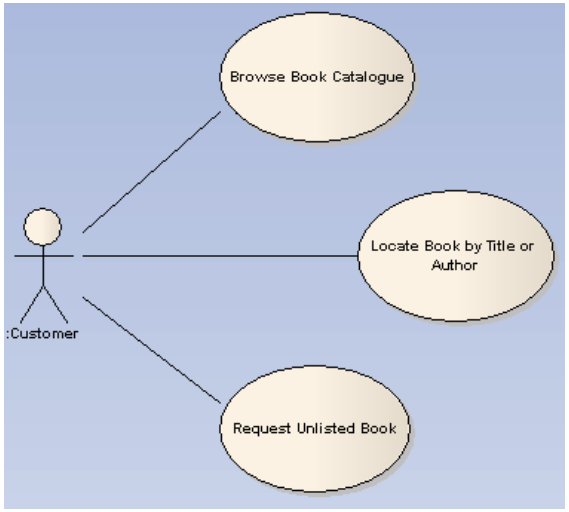
Entrega



Diagramas - Comportamentais

- **Casos de Uso**
- **Atividades**
- **Máquina de Estados**
- **Sequência**
- **Visão geral da interação**
- **Comunicação**
- **Tempo**

➢Normalmente usado durante levantamento e análise de requisitos.
➢ Identifica “atores” que de alguma forma vão interagir com o software e como o sistema irá responder a eles.
➢ Identifica funcionalidades a serem disponibilizadas aos atores.



Diagramas - Comportamentais

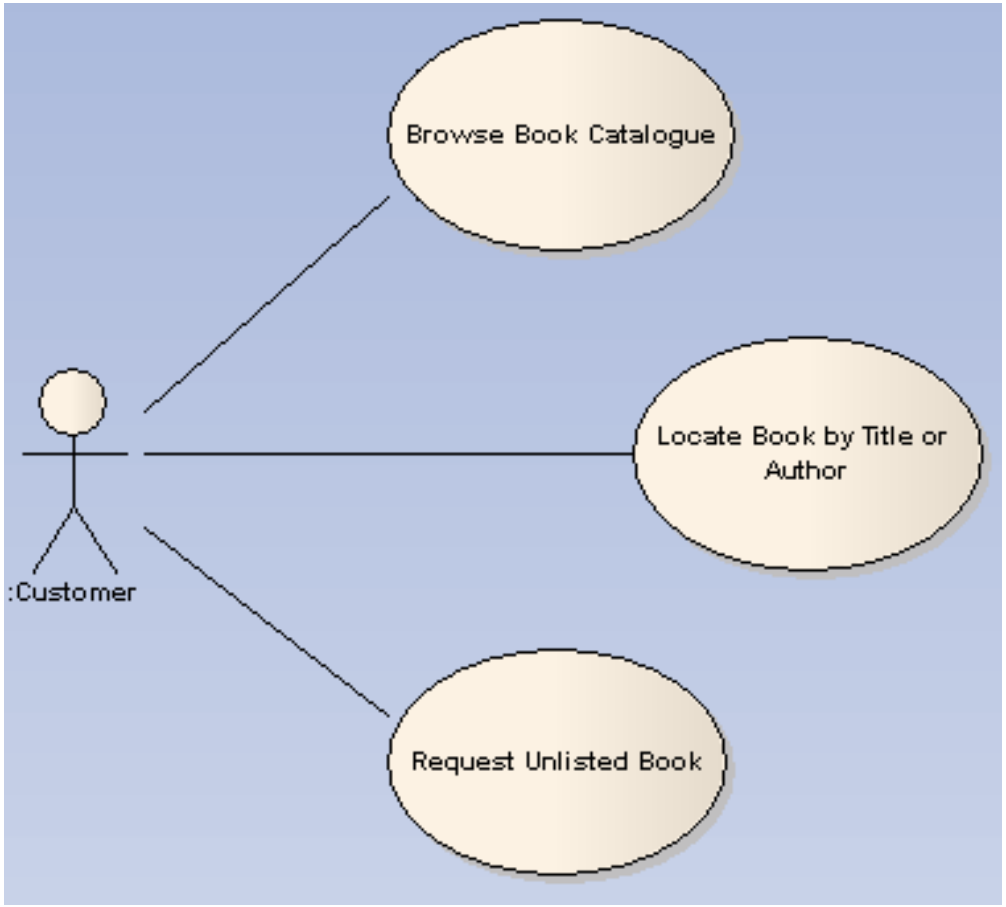
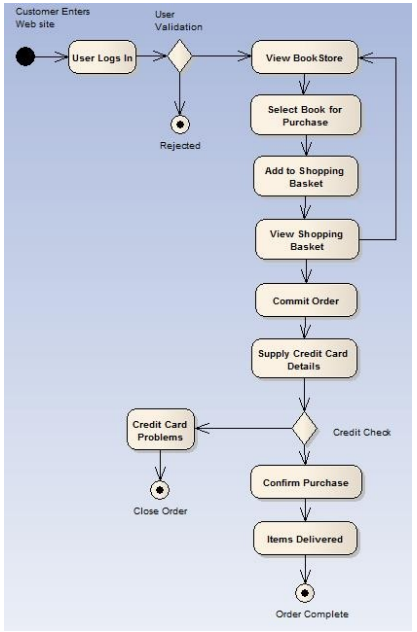


Diagrama de Casos de Uso (Use Case diagram)

Diagramas - Comportamentais

- **Casos de Uso**
- **Atividades**
- **Máquina de Estados**
- **Sequência**
- **Visão geral da interação**
- **Comunicação**
- **Tempo**

➢ Descreve passos a serem percorridos até a conclusão de uma atividade.
➢ Concentra-se no fluxo de controle de uma atividade.



Diagramas - Comportamentais

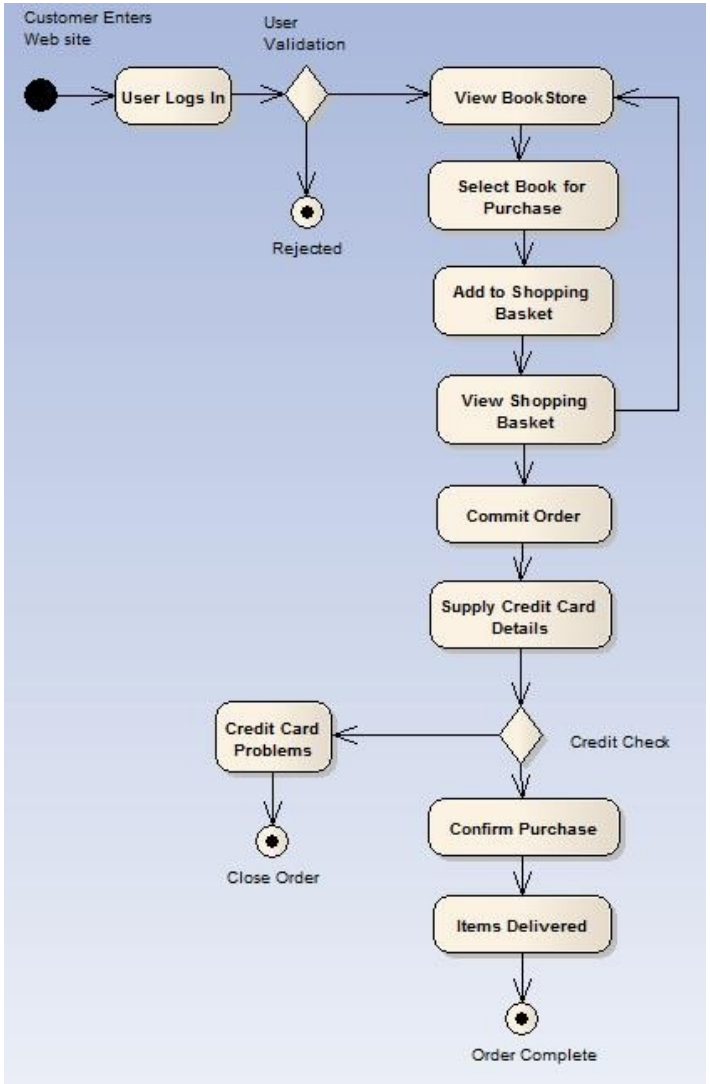
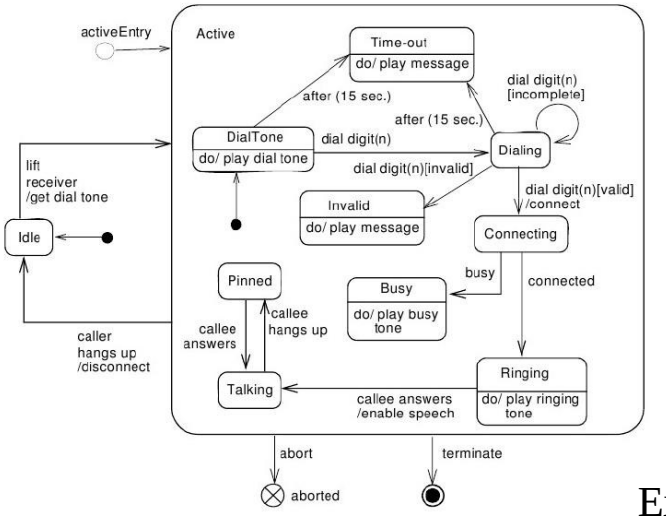


Diagrama de Atividades
(Activity diagram)

Diagramas - Comportamentais

- **Casos de Uso**
- **Atividades**
- **Máquina de Estados**
- **Sequência**
- **Visão geral da interação**
- **Comunicação**
- **Tempo**

➢Também conhecido como diagrama de estados.
➢Acompanha as mudanças sofridas por um objeto dentro de um determinado processo.
➢ Acompanha os estados pelos quais passa uma instância de uma classe.

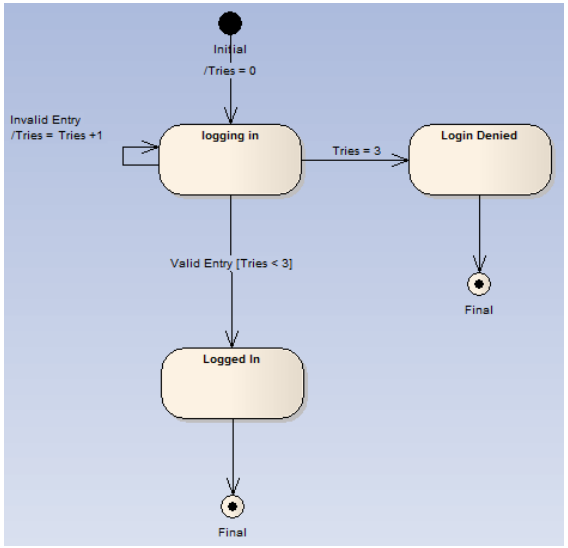


Exemplo 1

Diagramas - Comportamentais

- **Casos de Uso**
- **Atividades**
- **Máquina de Estados**
- **Sequência**
- **Visão geral da interação**
- **Comunicação**
- **Tempo**

➢Também conhecido como diagrama de estados.
➢Acompanha as mudanças sofridas por um objeto dentro de um determinado processo.
➢ Acompanha os estados pelos quais passa uma instância de uma classe.



Exemplo 2

Diagramas - Comportamentais

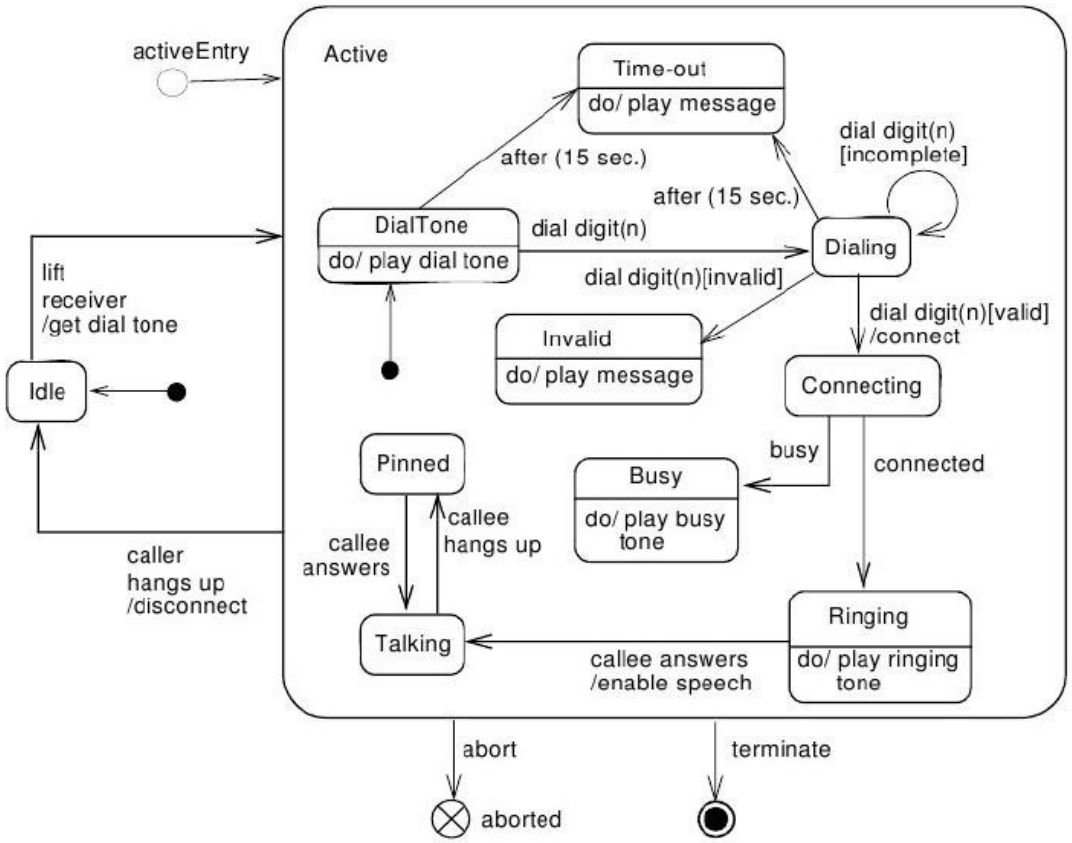


Diagrama de Máquina de Estados
(State Machine diagram)

Exemplo 1

Diagramas - Comportamentais

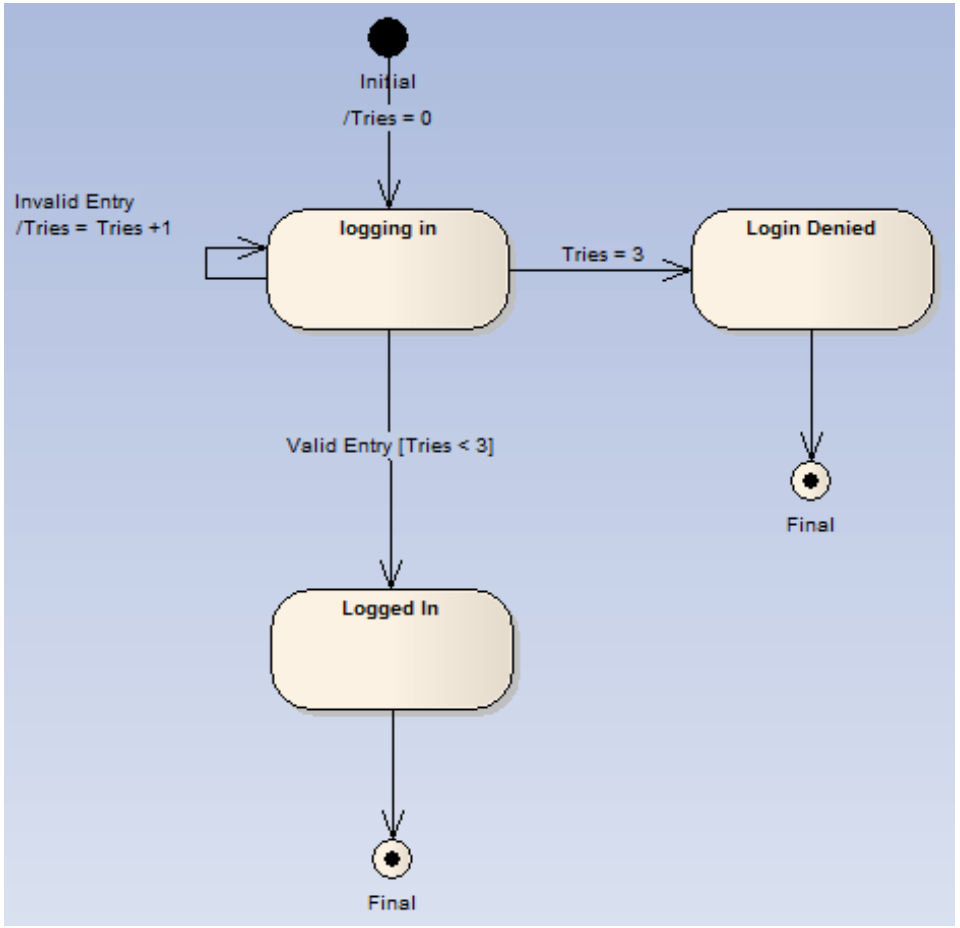
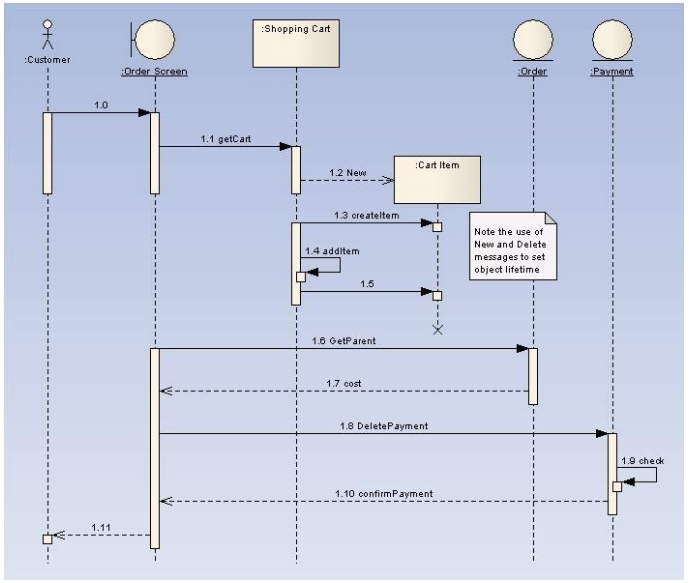


Diagrama de Máquina de Estados
(State Machine diagram)

Exemplo 2

Diagramas - Comportamentais

- Casos de Uso
- Atividades
- Máquina de Estados
- Sequência
- Visão geral da interação
- Comunicação
- Tempo



➢ Representa o comportamento do sistema ou parte do sistema como uma série de passos ao longo do tempo;
➢Ele é usado para descrever o fluxo de trabalho, a passagem de mensagens e como elementos cooperam entre si para alcançar um resultado.

Diagramas - Comportamentais

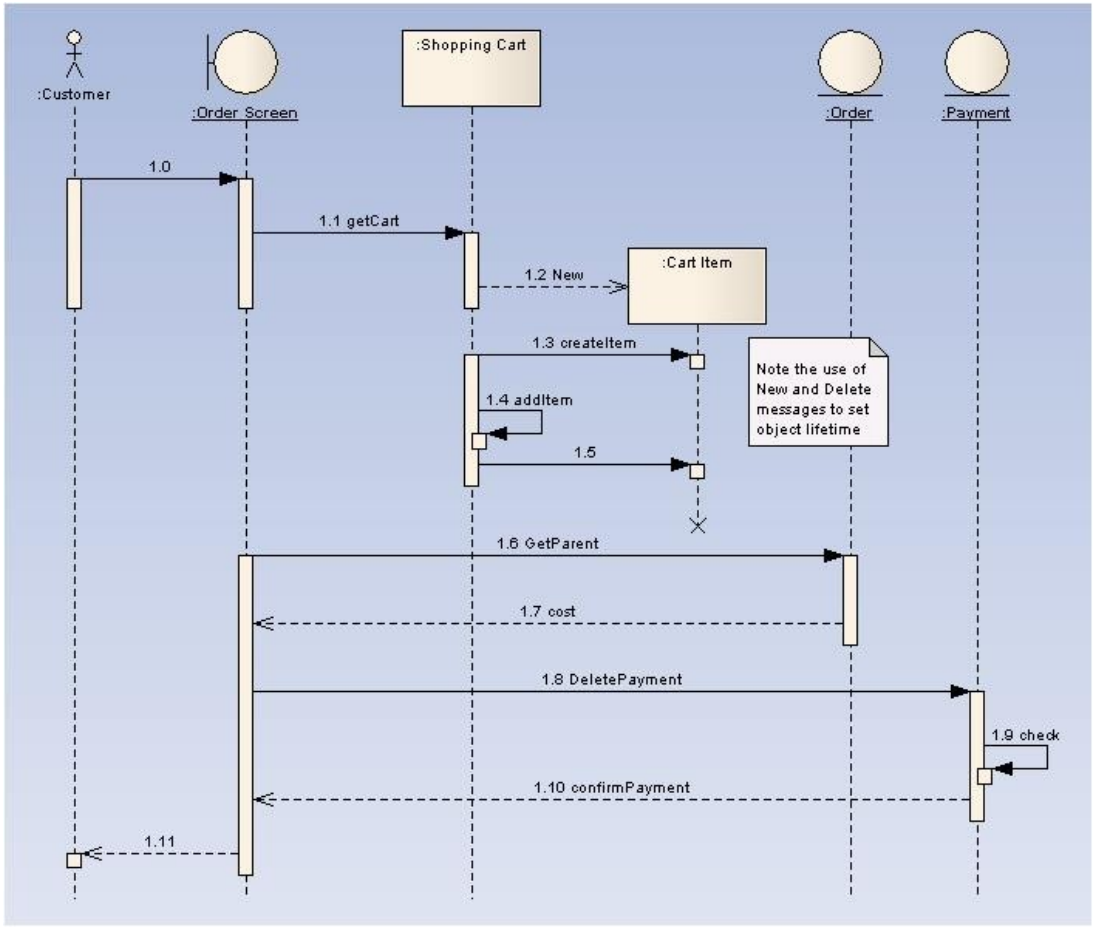
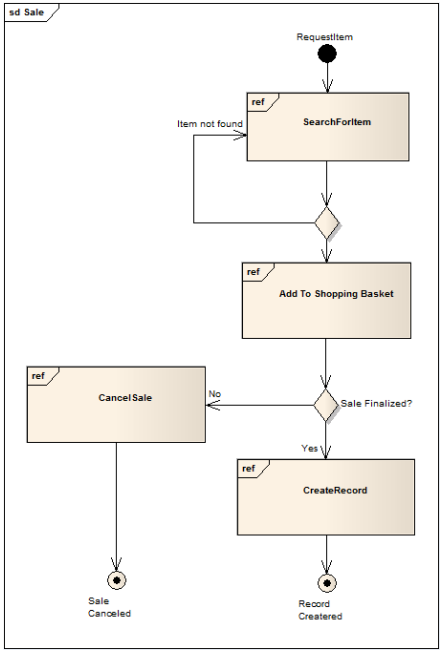


Diagrama de Sequência

(Sequence diagram)

Diagramas - Comportamentais

- Casos de Uso
- Atividades
- Máquina de Estados
- Sequência
- Visão geral da interação
- Comunicação
- Tempo



➢É uma variação do diagrama de atividades;
➢Permite visualizar a cooperação entre outros diagramas de interação ilustrando o fluxo de controle de um propósito mais abrangente.

Diagramas - Comportamentais

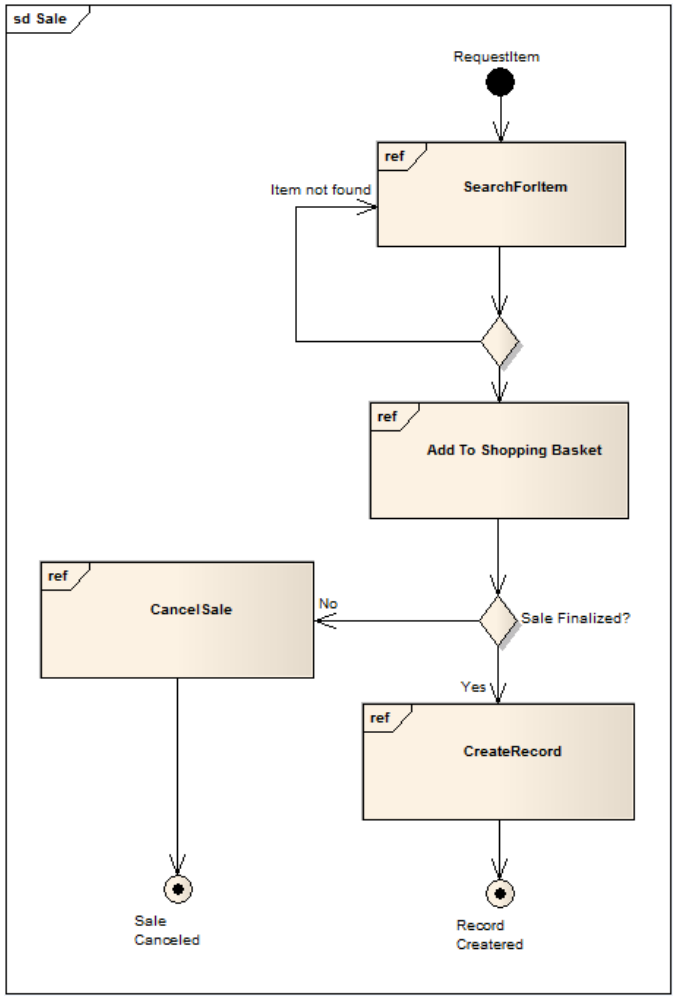
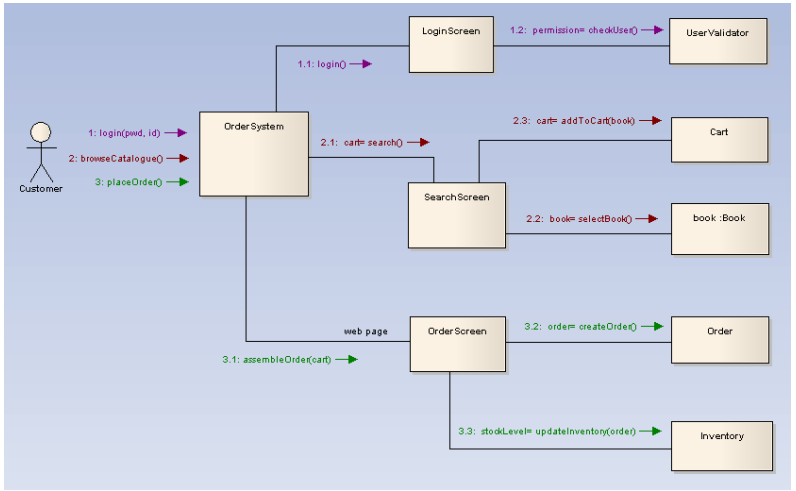


Diagrama de Visão Geral da interação (Interaction Overview diagram)

Diagramas - Comportamentais

- Casos de Uso
- Atividades
- Máquina de Estados
- Sequência
- Visão geral da interação
- Comunicação
- Tempo



➢ Mostra a interação entre elementos em tempo de execução.
➢ É similar ao diagrama de sequência, no entanto são usados para visualizar as relações entre objetos, enquanto os diagramas de seqüência são mais eficazes na visualização do processamento ao longo do tempo.

Diagramas - Comportamentais

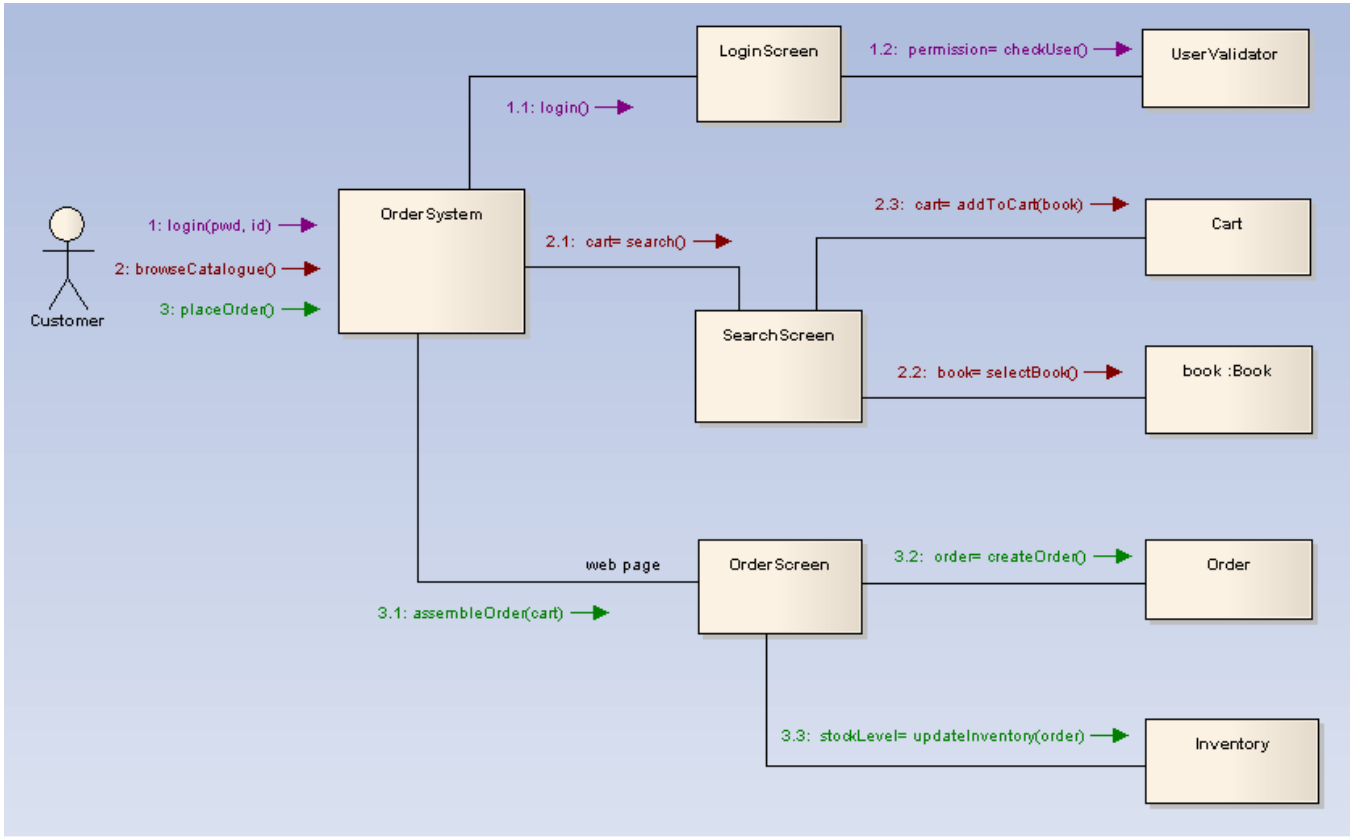
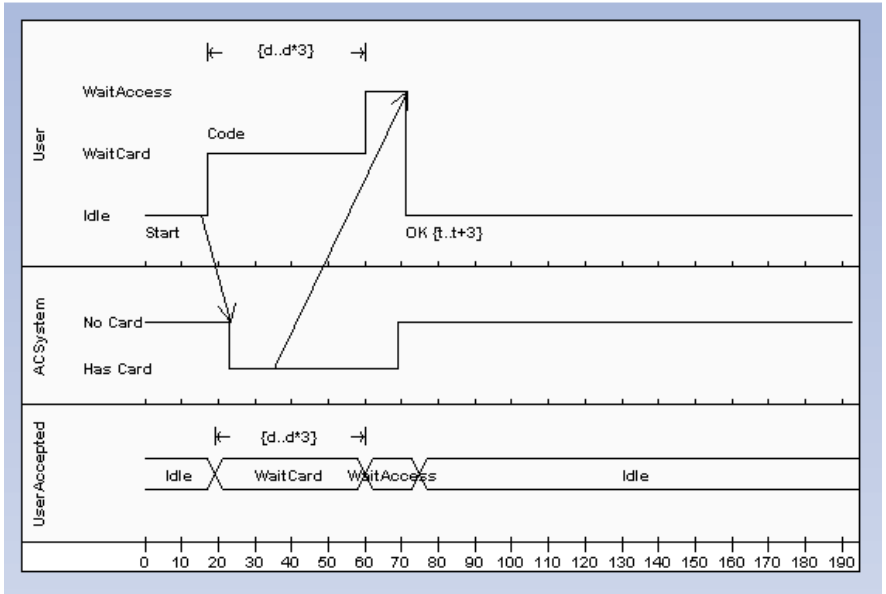


Diagrama de Comunicação

(Communication diagram)

Diagramas - Comportamentais

- Casos de Uso
- Atividades
- Máquina de Estados
- Sequência
- Visão geral da interação
- Comunicação
- Tempo



➢ Define o comportamento de diferentes objetos dentro de uma escala de tempo.
➢Permite a visualização da mudança de estado dos objetos ao longo do tempo.

Diagramas - Comportamentais

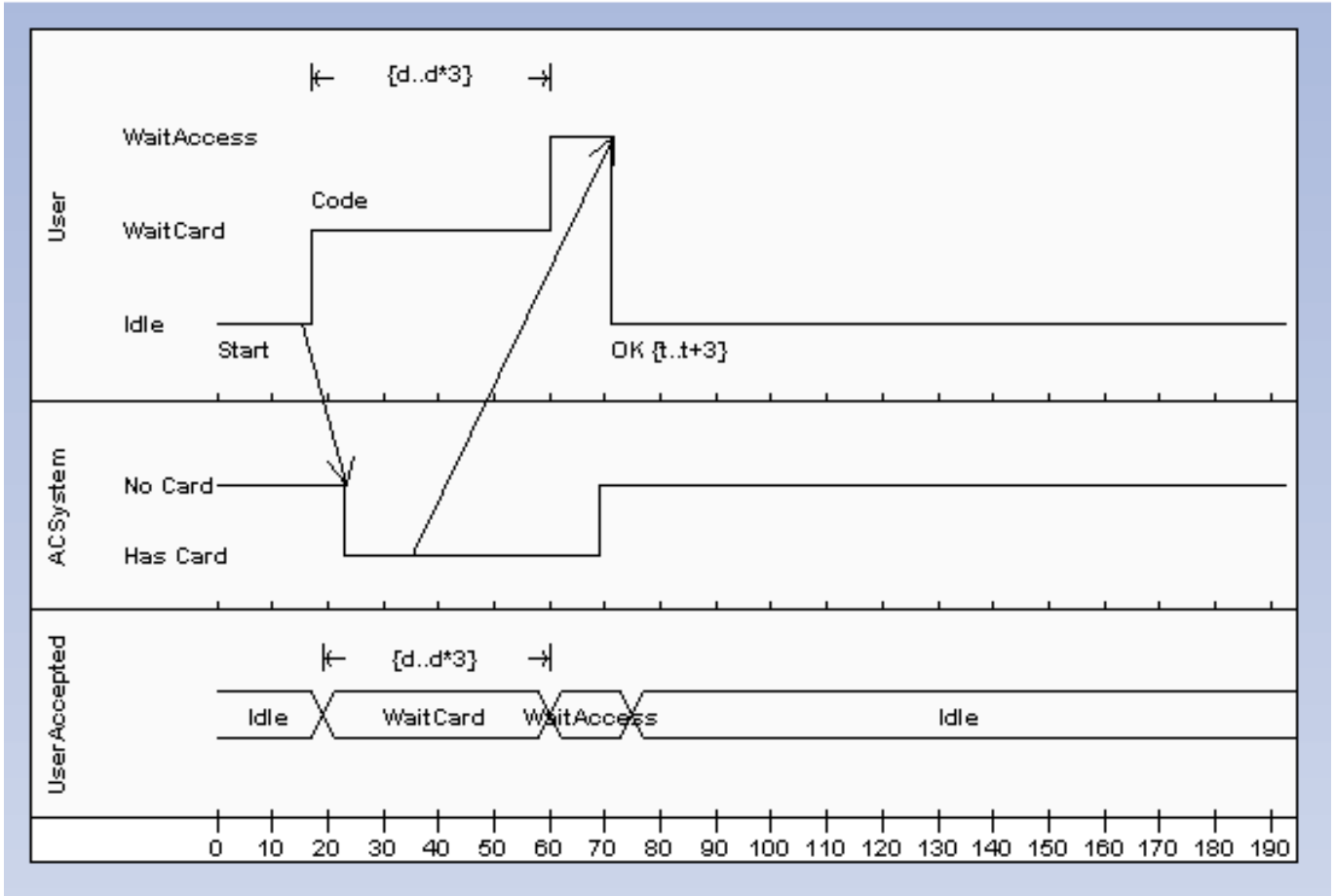
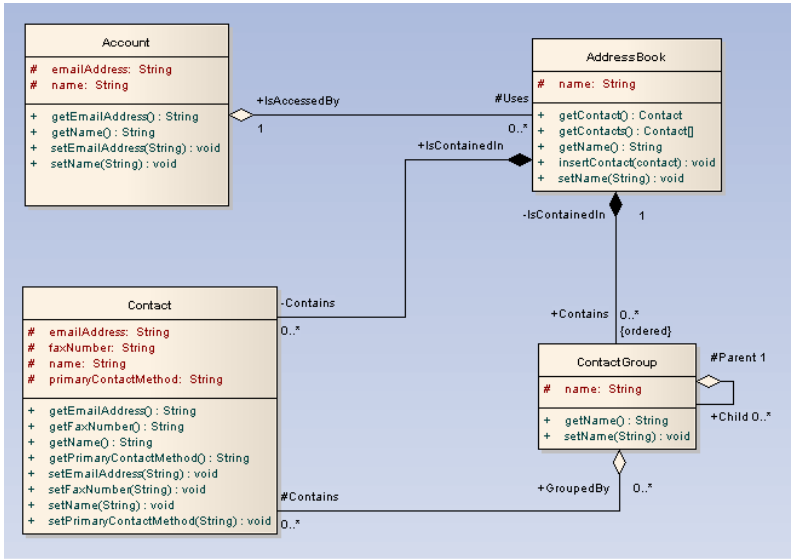


Diagrama deTempo (Timing diagram)

Diagramas - Estruturais

- Classes
- Objetos
- Estrutura Composta
- Componentes
- Pacotes
- Implantação

➢ Captura a estrutura lógica do sistema, as classes que compõem o modelo.
➢É um modelo estático, descrevendo o que existe, quais atributos e comportamentos um elemento possui, ao invés de como algo é realizado.
➢Diagramas de classe são mais úteis para ilustrar as relações entre classes e interfaces.



Diagramas - Estruturais

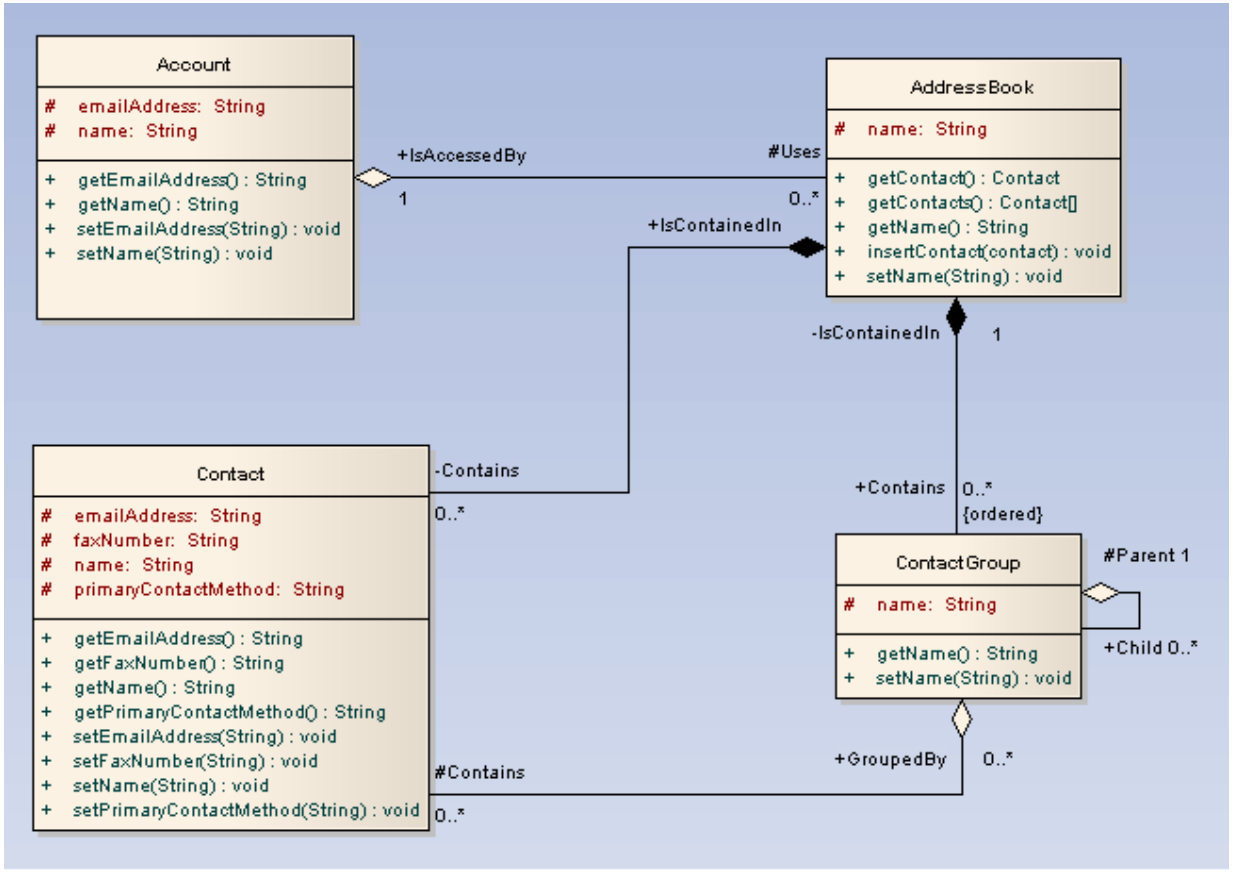


Diagrama de Classes (Class diagram)

Diagramas - Estruturais

- **Classes**
- **Objetos**
- **Estrutura Composta**
- **Componentes**
- **Pacotes**
- **Implantação**

➢ Um diagrama de objeto está intimamente relacionado com um diagrama de classes, com a distinção que retrata instâncias de objetos de classes e seus relacionamentos em um ponto no tempo.

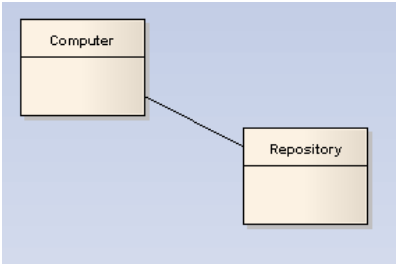


Diagrama de classe

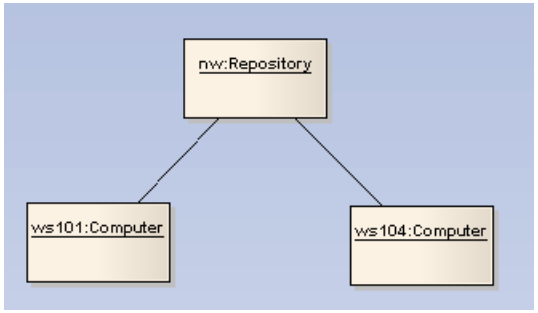


Diagrama de objetos

Diagramas - Estruturais

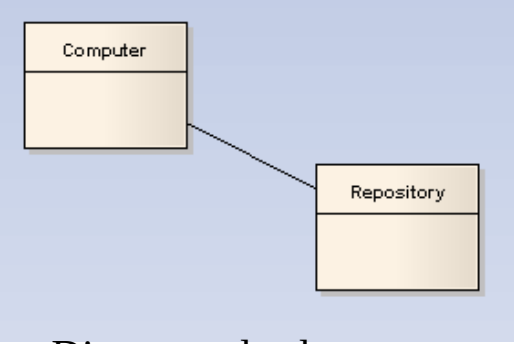


Diagrama de classe

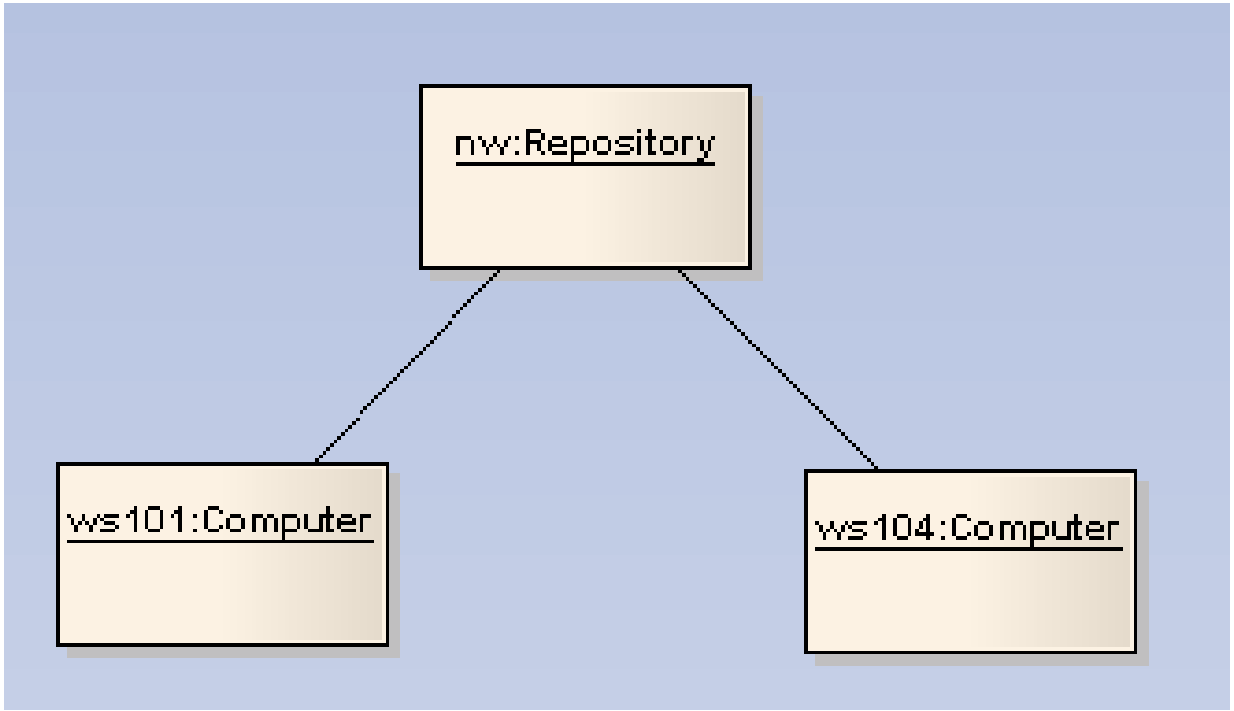
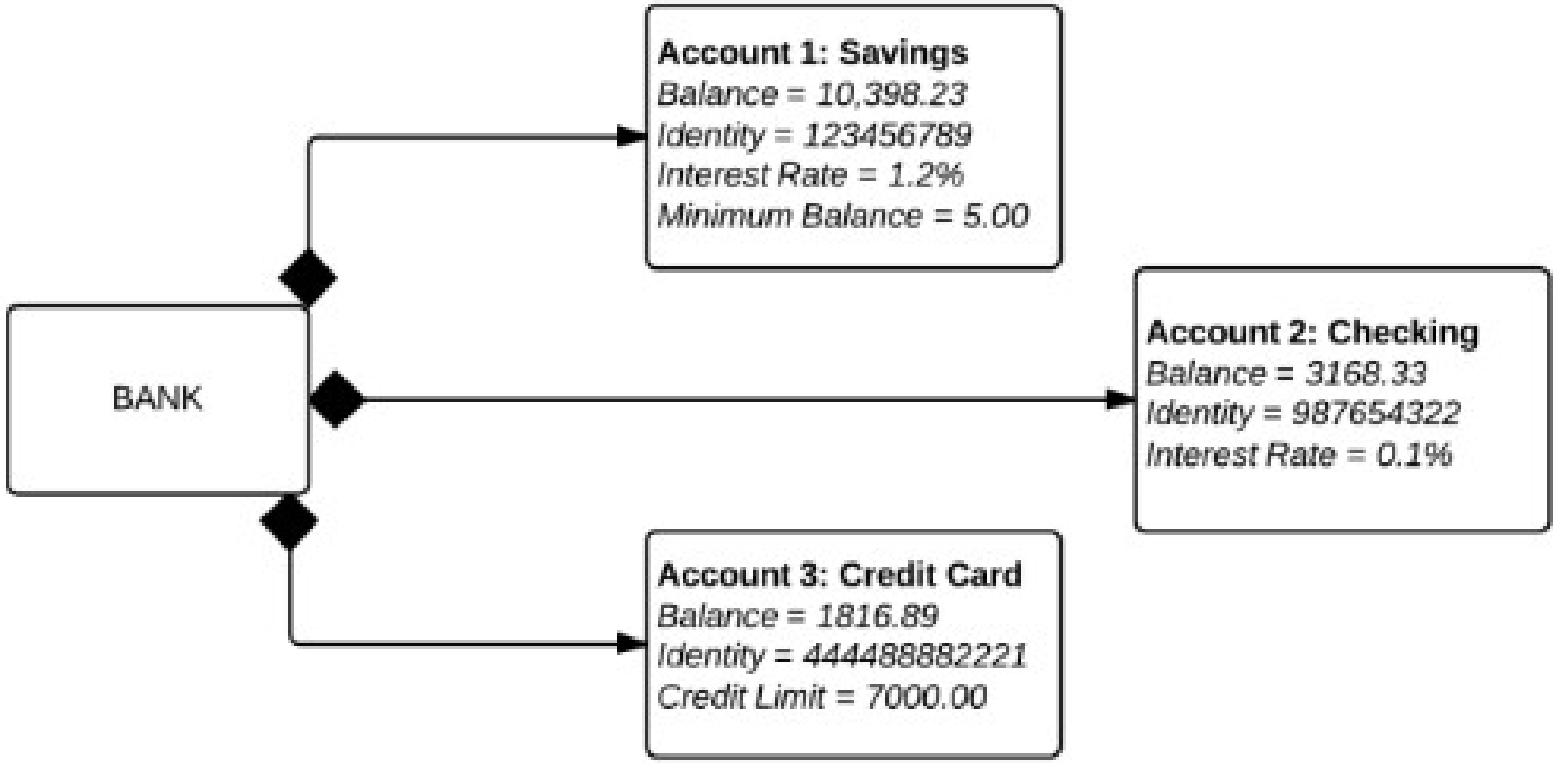


Diagrama de Objetos (Object diagram)

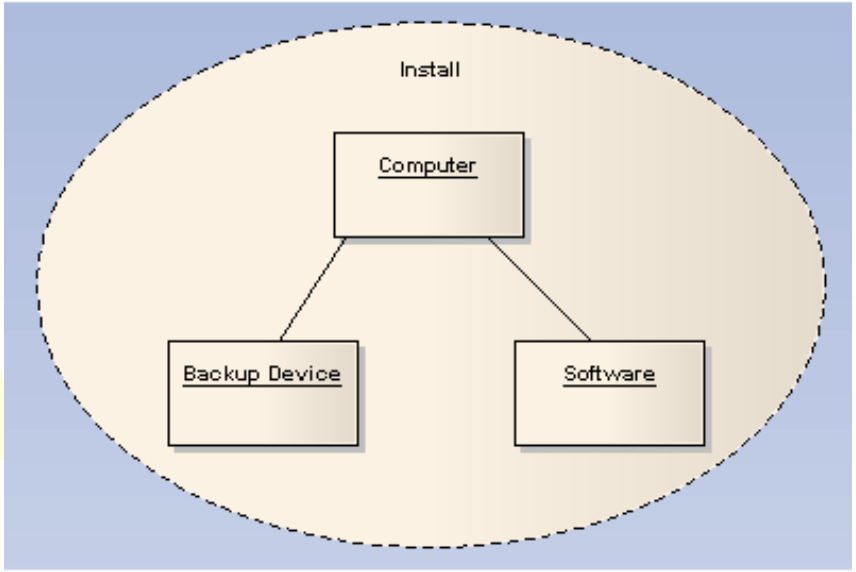
Diagramas - Estruturais



Diagramas - Estruturais

- Classes
- Objetos
- Estrutura Composta
- Componentes
- Pacotes
- Implantação

➢ Um diagrama de estrutura composta reflete a colaboração interna de classes, interfaces ou componentes para descrever a funcionalidade.



Diagramas - Estruturais

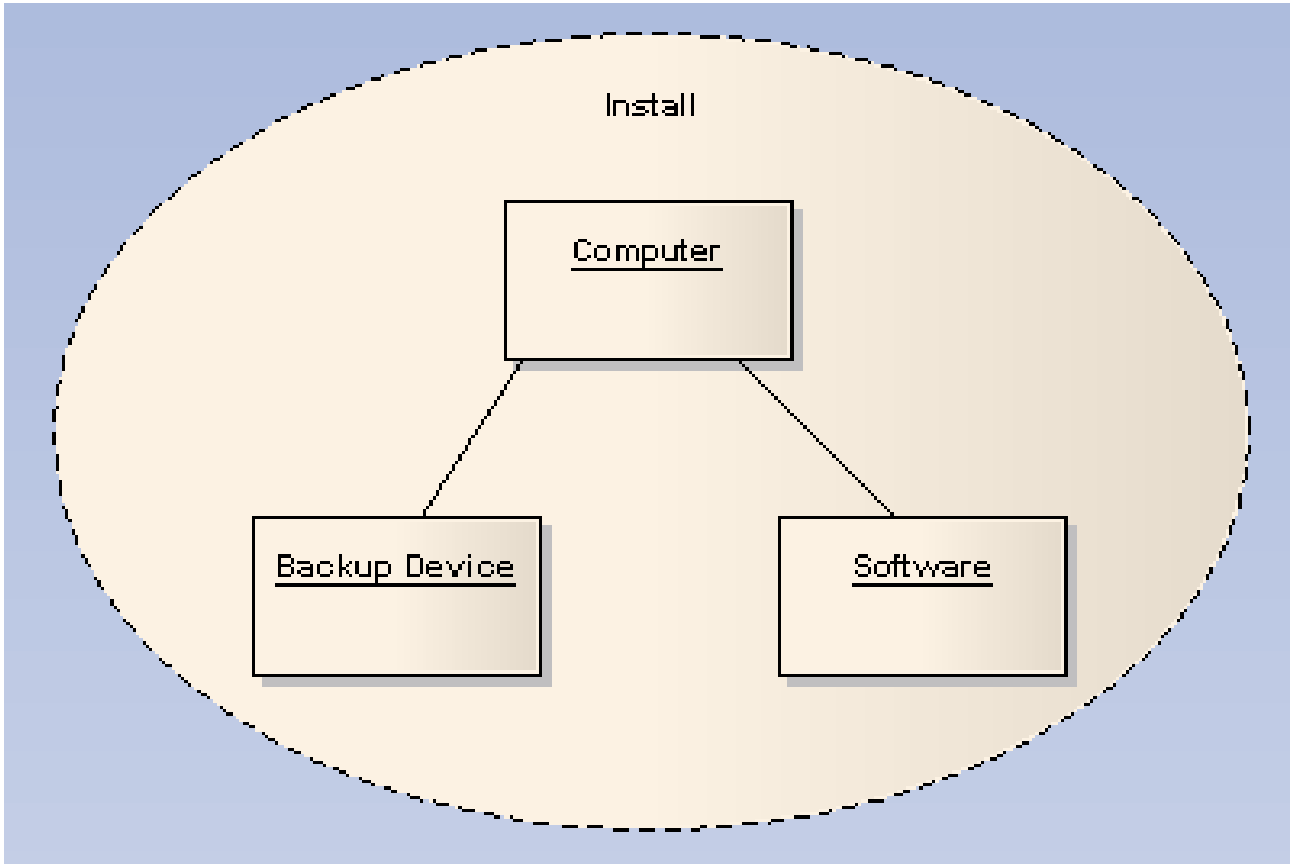
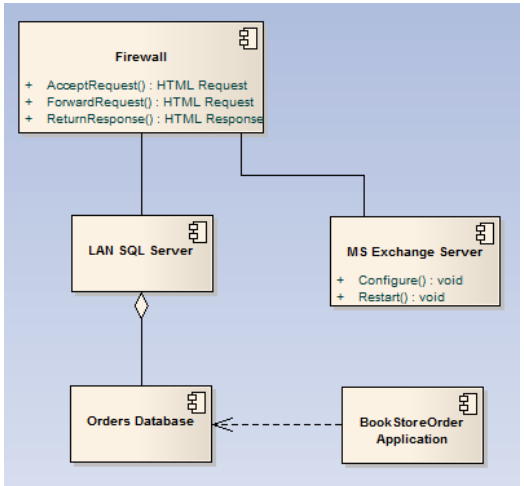


Diagrama de Estrutura Composta (Composite Structure diagram)

Diagramas - Estruturais

- Classes
- Objetos
- Estrutura Composta
- Componentes
- Pacotes
- Implantação



➢ Um diagrama de componente mostra as partes do software, sua organização e dependências, que formam o sistema.

➢O diagrama de componentes possui um nível maior de abstração que um diagrama de classes; geralmente um componente é implementado por uma ou mais classes (ou objetos) em tempo de execução.

➢Eventualmente um componente pode abranger uma grande parte de um sistema.

Diagramas - Estruturais

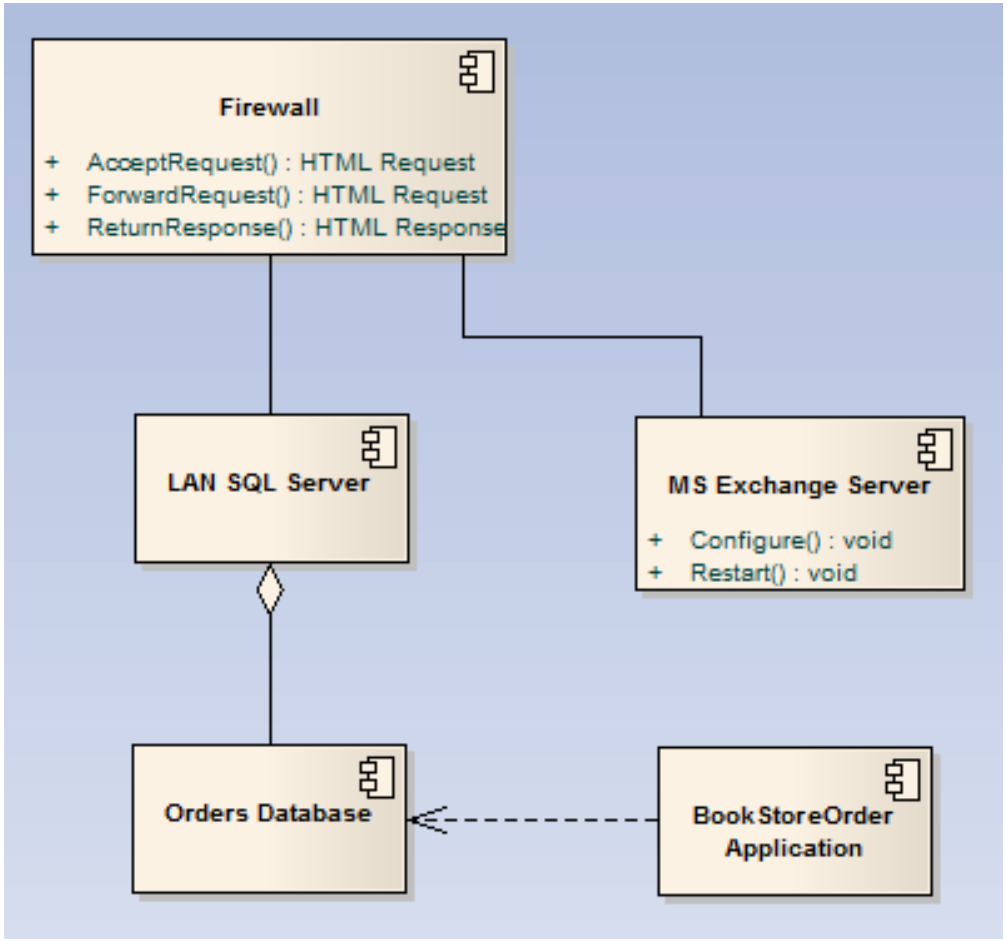
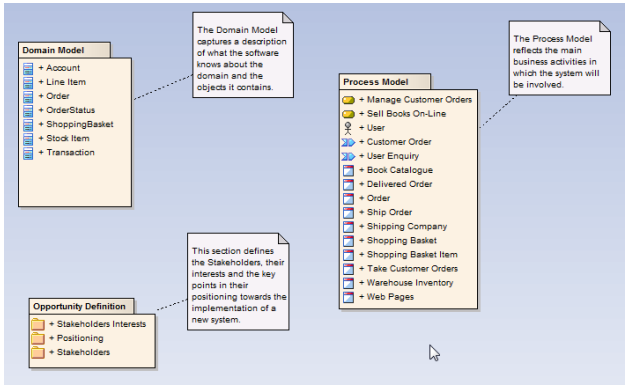
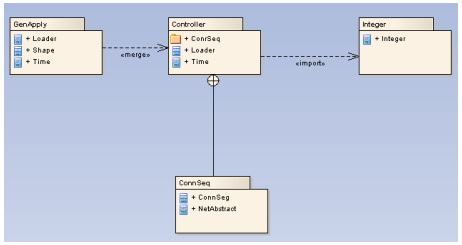


Diagrama de Componentes (Component diagram)

Diagramas - Estruturais

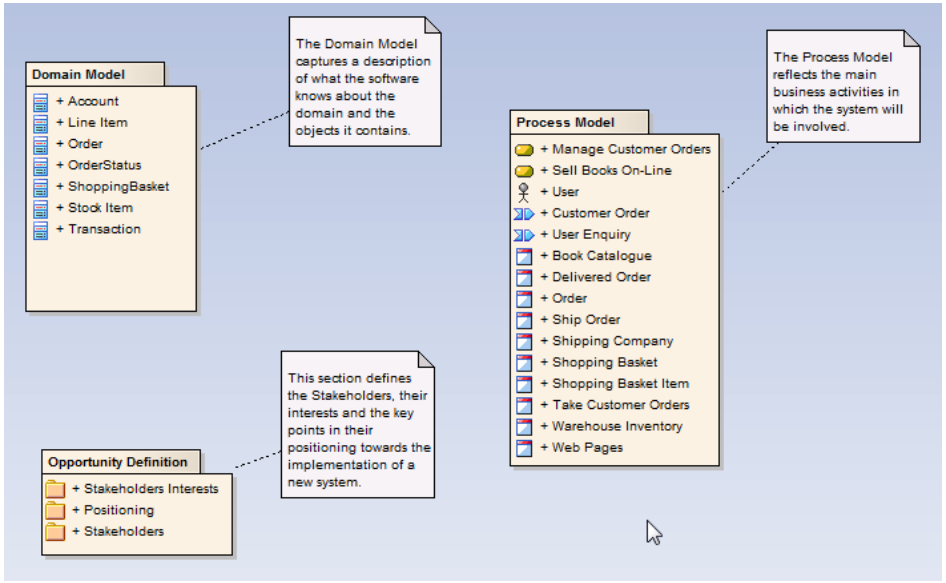
- Classes
- Objetos
- Estrutura Composta
- Componentes
- Pacotes
- Implantação





➢ Apresenta a organização de elementos de modelo e suas dependências;
➢ Permite a visualização dos "namespaces" correspondentes.

Diagramas - Estruturais



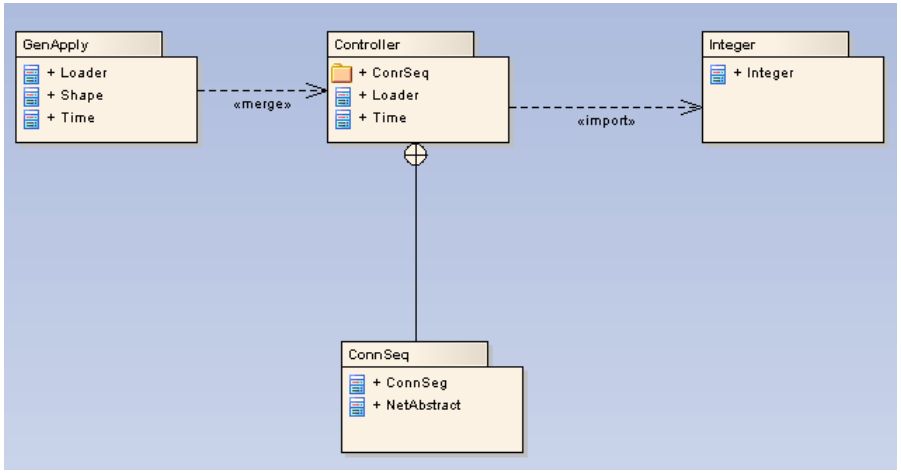
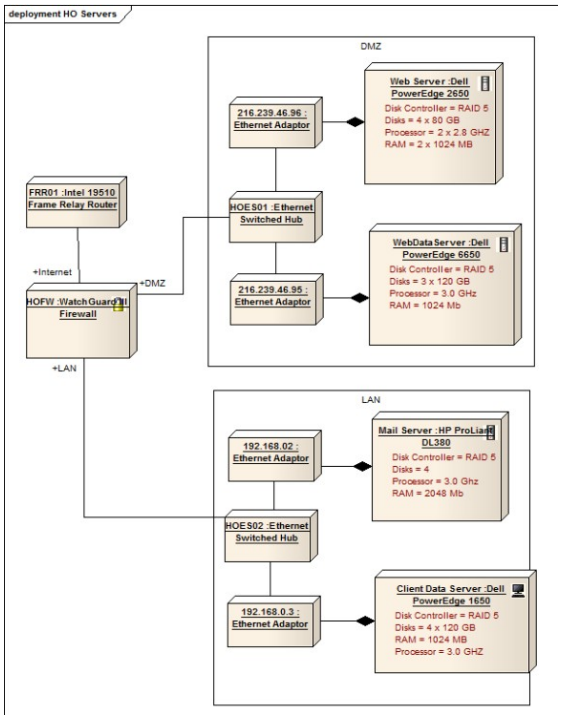


Diagrama de Pacotes (Package diagram)

Diagramas - Estruturais

- Classes
- Objetos
- Estrutura Composta
- Componentes
- Pacotes
- Implantação



➢Um diagrama de implantação mostra como e onde o sistema será implantado, ou seja, sua arquitetura de execução

➢Necessidades de hardware, características físicas: servidores, estações, topologia, protocolos, etc.

Diagramas - Estruturais

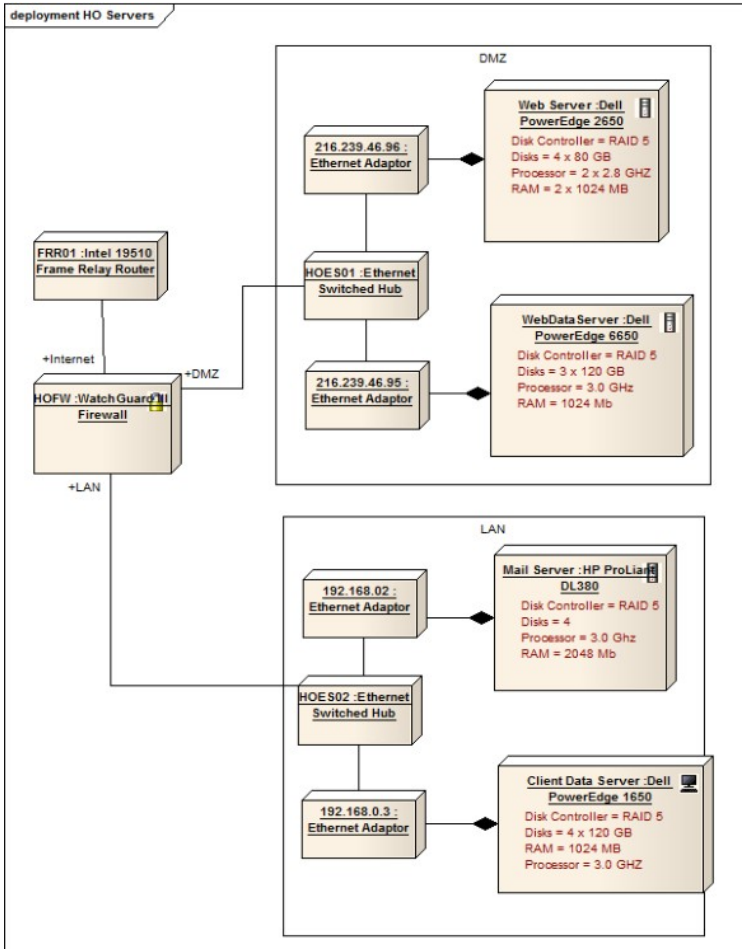


Diagrama de Implantação (Deployment diagram)


UNIFIED
MODELING
LANGUAGE
TM

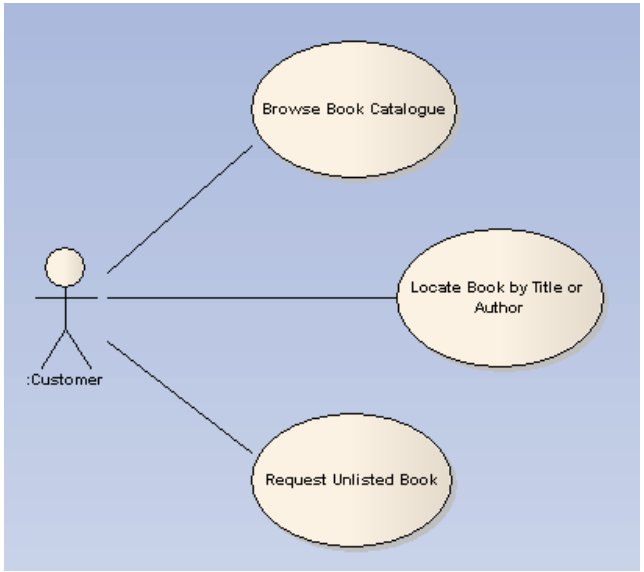
Diagrama de Use Case

O diagrama de Use Cases captura as Use Cases (casos de uso) e o relacionamentos com os atores.

Ele descreve os requisitos funcionais do sistema e a maneira pela qual os atores interagem com o sistema.

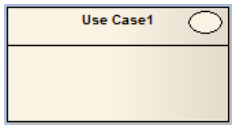
Um Caso de Uso descreve a funcionalidade proposta de um novo sistema.

Um Caso de Uso representa uma unidade discreta da interação entre um usuário (humano ou máquina) e o sistema. Essa interação é uma única unidade de trabalho significativo, como **Browse Book Catalogue** ou **Locate Book by Title or Author.**



Elementos do diagrama de Use Case

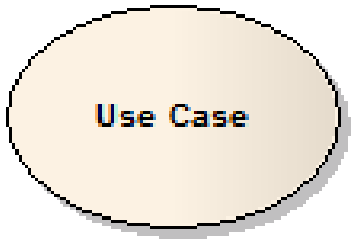
Use Case ou Caso de Uso

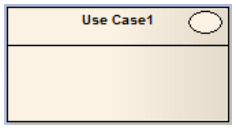


Representada por uma elipse e um nome (que pode estar dentro ou abaixo). Também pode ser representada por um retângulo com uma elipse no topo direito.

Descreve uma funcionalidade do sistema que é descrita por um fluxo de eventos ou cenário.

Elementos do diagrama de Use Case

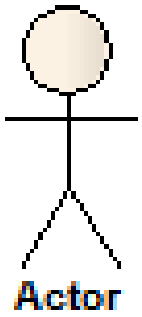


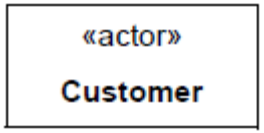


A descrição de uma Use Case geralmente inclui:

- um identificador único;
- uma descrição geral da funcionalidade que a Use Case representa;
- os requisitos associados;
- os atores associados;
- pré-condições para que o fluxo de eventos ocorra;
- pós-condições atingidas após a ocorrência do fluxo;
- cenários:
    - cenário básico;
    - cenários alternativos (casos de exceção e fluxos alternativos);

Elementos do diagrama de Use Case





Ator

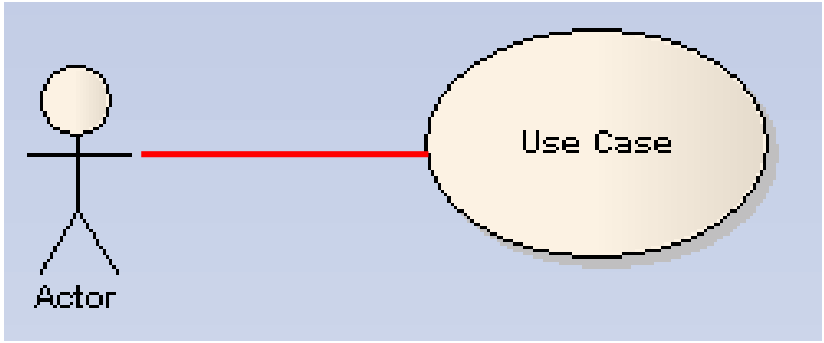
Representado por um boneco, um retângulo (com o estereótipo <<actor>> ou um ícone.

Um ator é um usuário do sistema: o usuário pode ser uma pessoa, uma máquina, ou mesmo outro sistema.

Qualquer coisa que interaja com o sistema a partir de sua fronteira é chamado de ator.

Conectores principais

Associação:



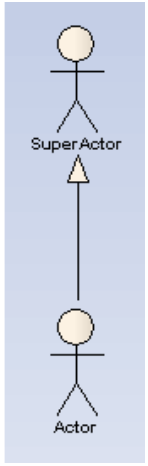
Um conector do tipo **Associação** mostra que dois ou mais elementos do modelo têm um relacionamento.

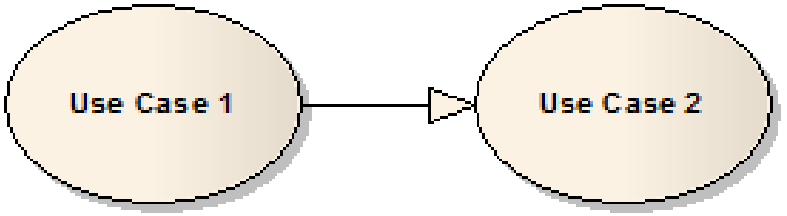
Esse conector pode incluir nome de papéis no seu final, multiplicidade, direção e restrição.

Uma associação entre um Ator e um Caso de Uso demonstra que o Ator utiliza-se da função do sistema representada pelo Caso de Uso.

Conectores principais

Generalização:





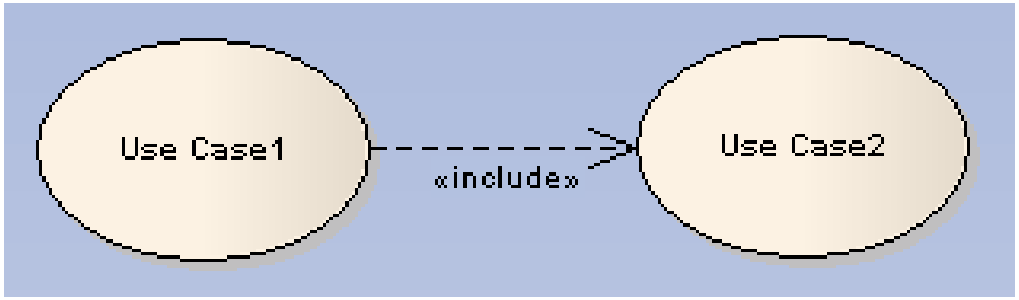
Um conector do tipo **Generalização** é utilizado para indicar herança.

É representado por uma seta partindo de uma fonte para um alvo: implica que a fonte **herda** as **características** do elemento alvo.

Pode existir entre Use Cases e entre Atores.

Conectores principais

Inclusão:



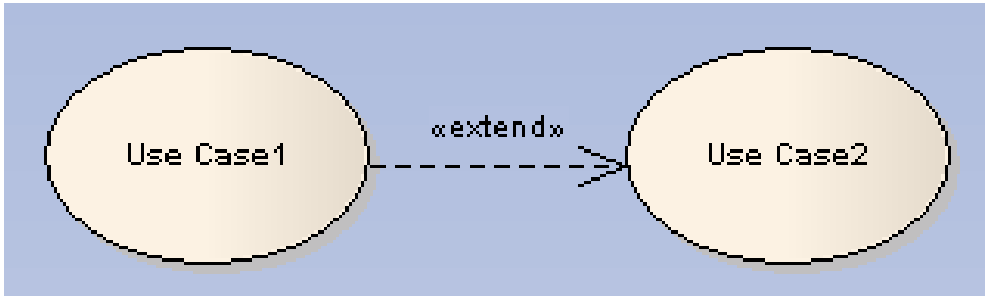
Um conector do tipo ***Include*** é utilizado para indicar inclusão de comportamento.

É representado por uma seta tracejada partindo de em elemento fonte para um alvo, com o estereótipo “include”; implica que o elemento fonte **inclui** as **funcionalidades** do elemento alvo.

Pode existir entre Use Cases e é utilizado para evitar que o mesmo comportamento tenha que ser repetido em várias Use Cases.

Conectores principais

Extensão:



Um conector do tipo ***Extend*** é utilizado para indicar extensão de comportamento.

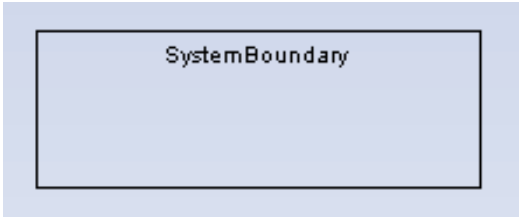
É representado por uma seta tracejada partindo de em elemento base para outro, com o estereótipo “extend” : implica que um elemento **estende** as **funcionalidades** do elemento base.

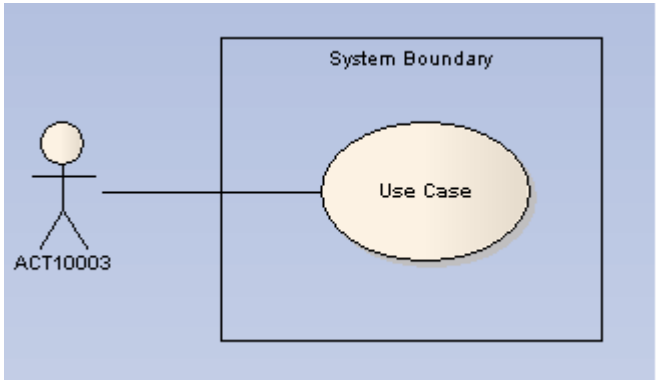
Pode existir entre Use Cases.

É utilizado para representar fluxos complementares; o comportamento do elemento base pode existir sem o comportamento estendido.

Fronteira

Fronteira do sistema:



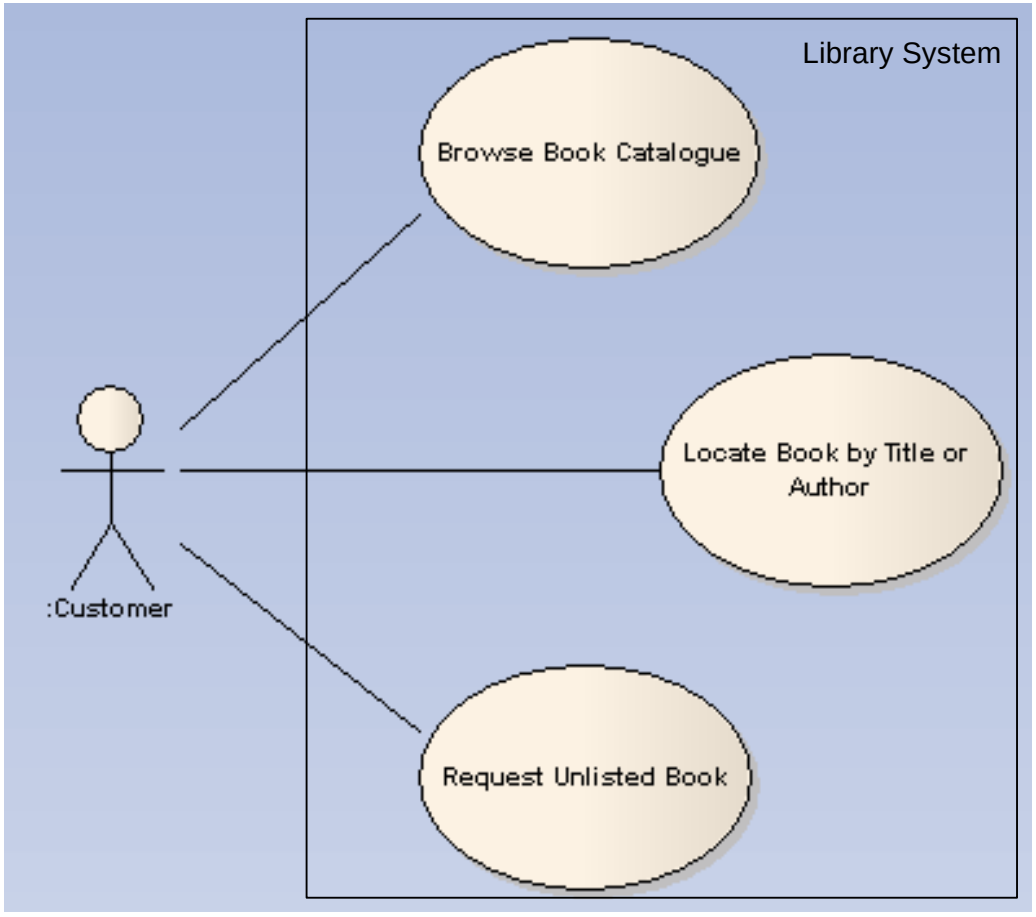


Fronteira (*System Boundary*) é um elemento classificador no qual atuam as Use Cases nela incluídas.

É representada por um retângulo e o nome do “assunto” (que classifica o sistema).

Mais utilizado quando existem mais de um sistema (fronteira) a ser representado e portanto para indicar quais use cases estão contidas em cada um deles.

Exemplo 1



Fonte: Enterprise Architect

Exemplo 2

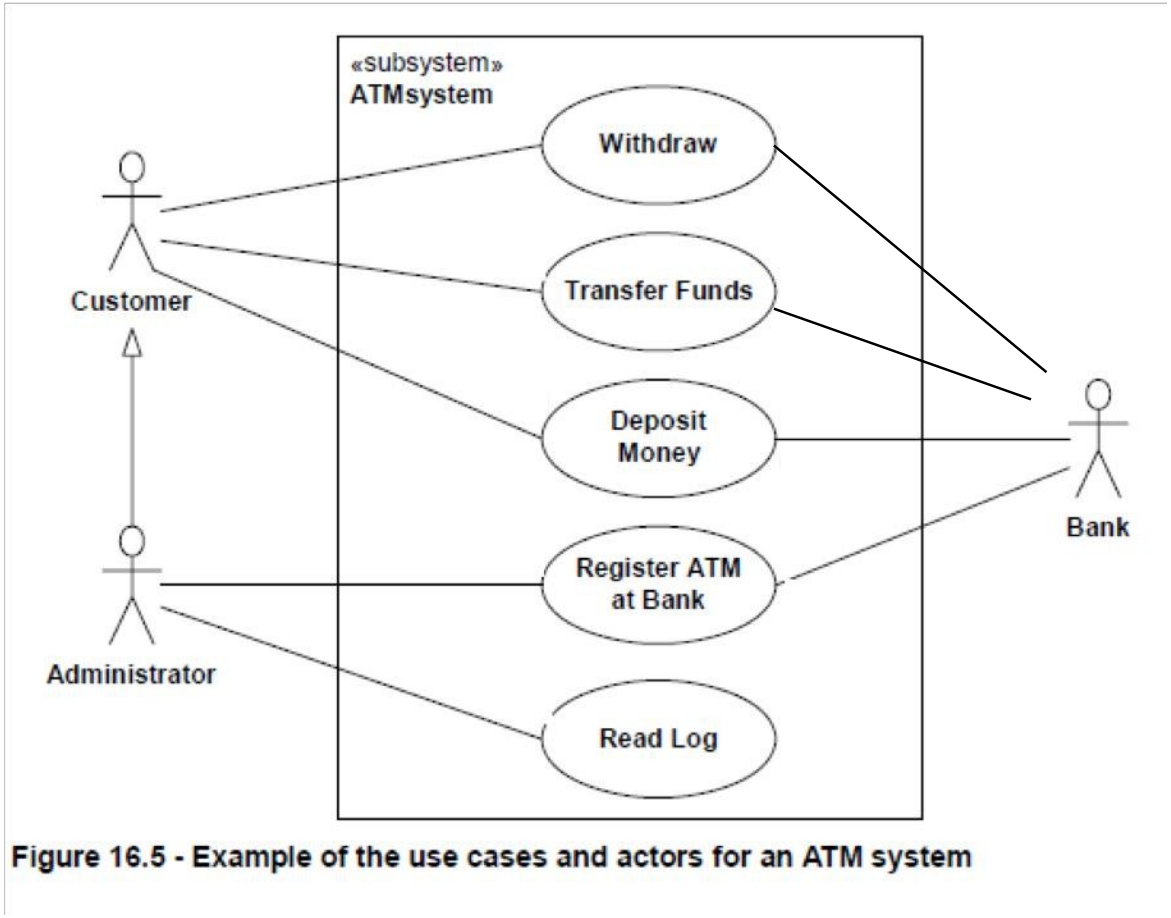
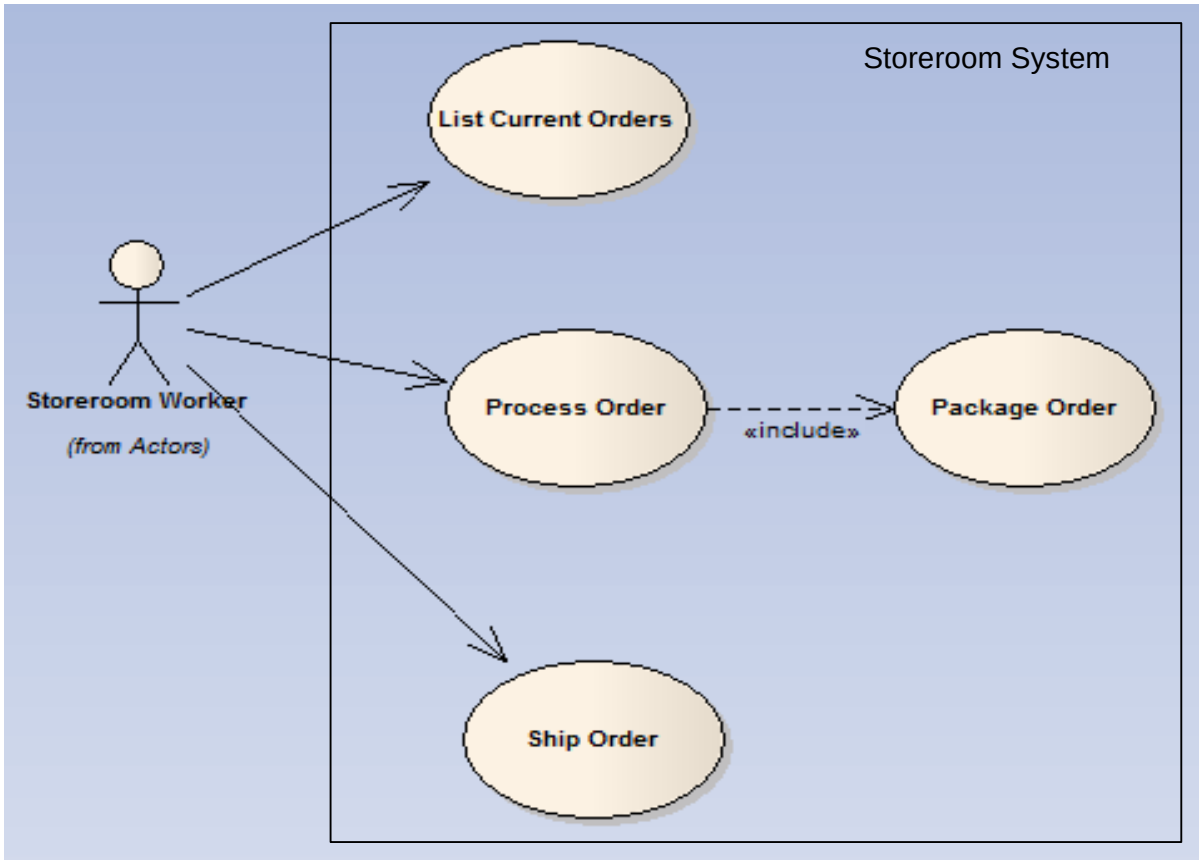


Figure 16.5 - Example of the use cases and actors for an ATM system

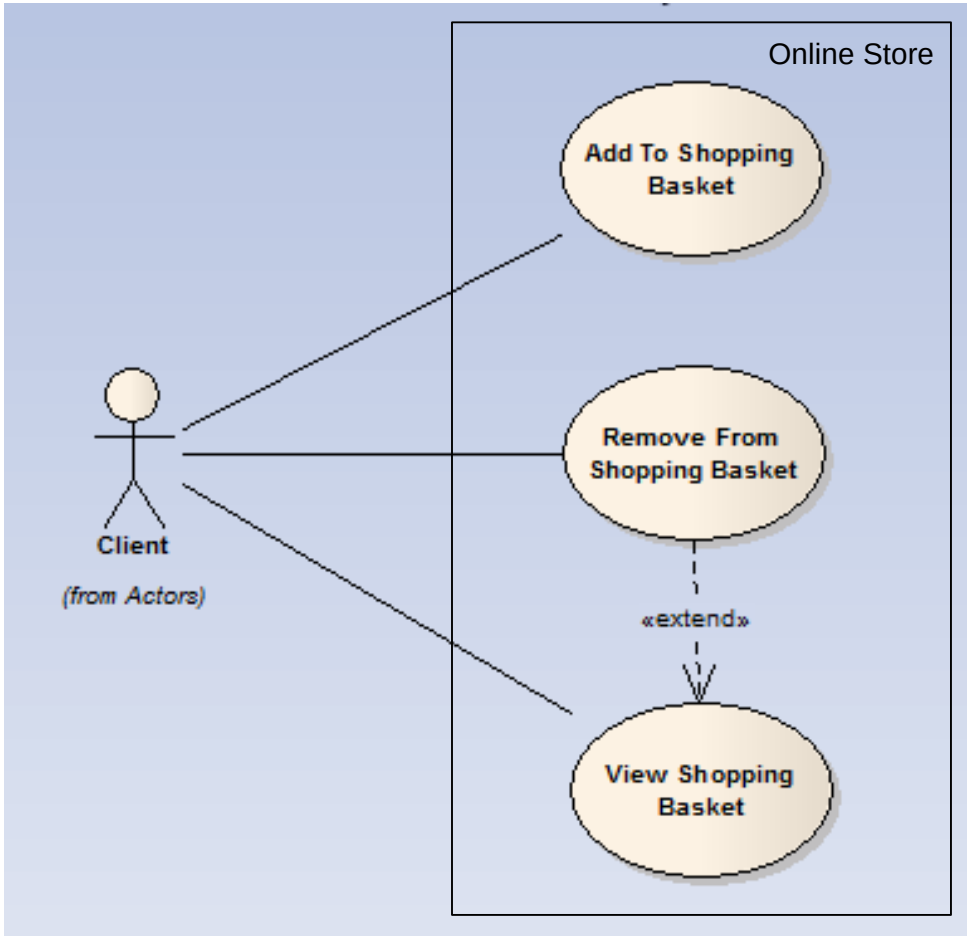
Fonte: UML Superstructure Specification, v2.3 – page 614 (adaptado)

Exemplo 3



Fonte: Enterprise Architect

Exemplo 4



Fonte: Enterprise Architect

Diagrama de Casos de Uso

Fluxo de Eventos

- **Casos de Uso – Documentação**

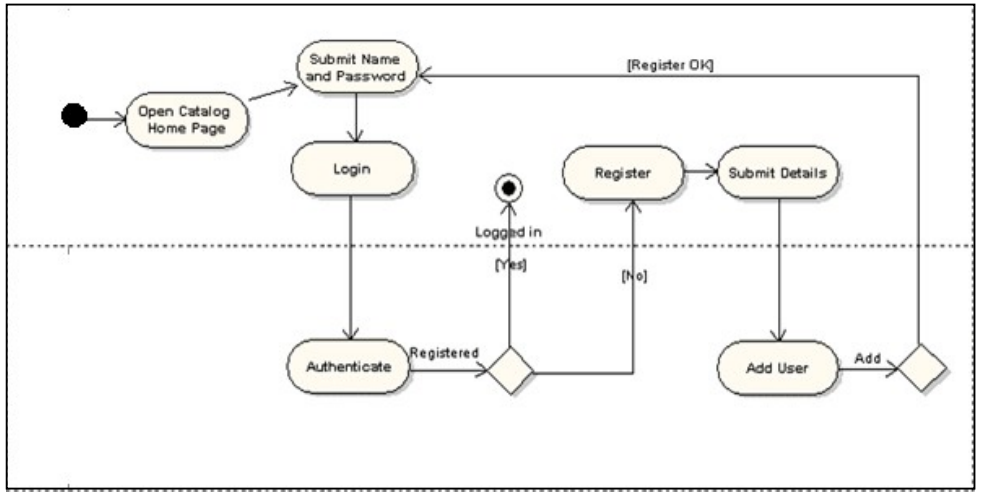
Nome do Caso de Uso	Sacar Dinheiro
Caso de Uso Geral	
Ator Principal	Cliente
Atores Secundários	Não há
Resumo	Descreve o funcionamento do sistema para o saque de dinheiro em conta corrente efetuado por um cliente.
Pré-Condições	Conta bancária precisa estar ativa.
Pós-Condições	Debitar valor sacado da conta corrente.
Ações Recebidas	**Ações Realizadas**
1. Cliente solicita o saque.	
	2. Sistema solicita que ele passe o cartão.
3. Cliente passa o cartão solicitado.	
	4. Sistema verifica dados do cartão.
	5. Se dados estiverem corretos, sistema solicita a senha.
6. Cliente digita a senha.	
	7. Sistema verifica senha.
	8. Se senha estiver correta, sistema solicita que o cliente digite valor.
9. Cliente digita o valor.	
	10. Sistema verifica saldo.
	11. Se saldo for suficiente, sistema libera valor solicitado.
	12. Sistema informa valor sacado ao módulo de controle de saldo.
	13. Caso de uso finalizado.

Diagrama de Casos de Uso – Exercício 1

- Uma agência de viagens pretende disponibilizar um sistema de software via web para que as pessoas possam acessar informações sobre pacotes de viagem, preços, sugestão de destino, promoções, etc, que serão cadastradas por um funcionário da agência.

Diagrama de Casos de Uso – Exercício 2

- O sistema Catalog é um sistema web que permite a pesquisa de obras de arte do museu Louvre. Qualquer pessoa pode acessar o sistema, mas para iniciar uma pesquisa é necessário realizar um login, onde é passado o nome do usuário e sua senha. Uma vez autenticado, o usuário tem disponível uma série de opções de busca e informações sobre as obras. Se o usuário digitar um usuário e senha inválidos (caso não tenha ou caso tenha esquecido) o sistema solicita o registro de seus dados para que um novo nome e senha possam ser utilizados. Veja o diagrama de atividades do módulo “Acessar sistema”:



Fonte: http://hoirul.its-sby.edu/ADBO/UseCase_Document_Template.doc

Pede-se:
faça o diagrama de use cases e a sua documentação, isto é, as tabela(s) de fluxo de eventos.

Diagrama de Casos de Uso

Em que etapa de um processo de desenvolvimento de software o diagrama de use cases mostra-se mais eficaz?

Na etapa de especificação, auxiliando no entendimento e visualização dos requisitos do sistema.


UNIFIED
MODELING
LANGUAGE
TM

Diagrama de Classes

O diagrama de Classes captura a estrutura lógica do sistema, ou seja, como o sistema está organizado para prover todas as suas funcionalidades;

Um diagrama de classe pode ser criado com diferentes níveis de abstração, ou seja, normalmente os seus detalhes são inseridos conforme as etapas de desenvolvimento do sistema avançam;

Desta forma, pode-se criar diagrama de classes de domínio, de análise e de projeto.

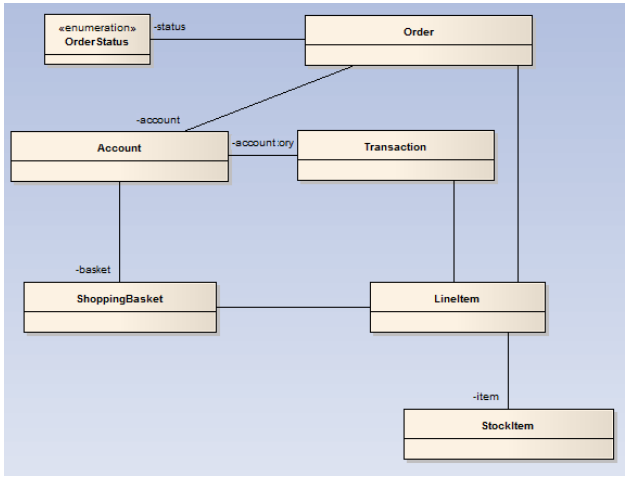


Diagrama de Classes – níveis de abstração

Em nível de **domínio** podem ser exibidos somente os nomes das classes e seus relacionamentos:

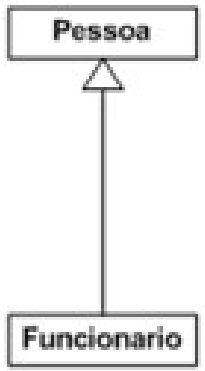


Diagrama de Classes – níveis de abstração

Em nível de **análise** podem ser exibidos os nomes das classes, seus atributos e relacionamentos:

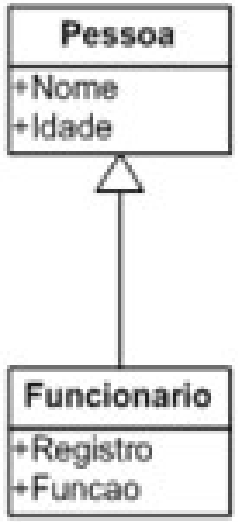
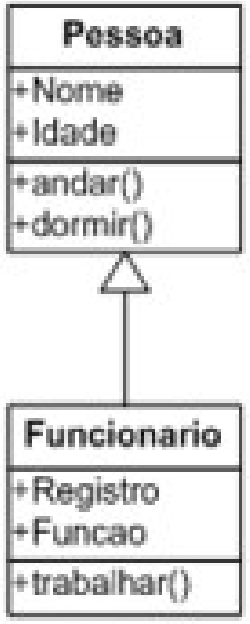
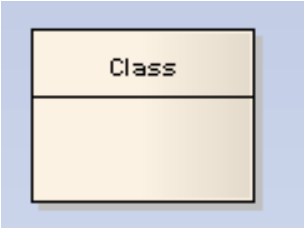


Diagrama de Classes – níveis de abstração

Em nível de **projeto** podem ser exibidos os nomes das classes, seus atributos, métodos e relacionamentos:

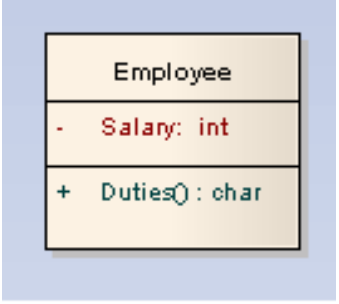


Elementos do Diagrama de Classes



Classe: representada por um retângulo com o ser nome na parte superior;

Uma classe pode ter atributos (que representam os dados) e métodos ou operações (que representam comportamento);



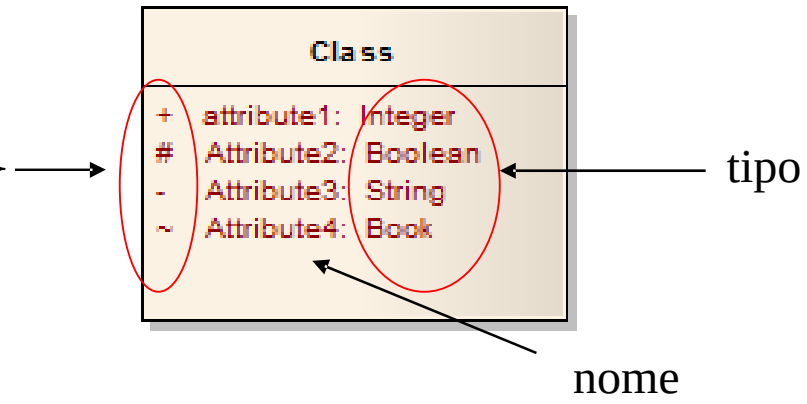
Normalmente a classe é mostrada com 3 compartimentos: o compartimento central que apresenta a lista de atributos e o compartimento inferior que apresenta a lista de operações.

Elementos do Diagrama de Classes

Os atributos da classe possuem características importantes, tais como, como escopo (visibilidade) e tipo.

Escopo (visibilidade)

	Modifier	Class	Package	Subclass	World
(+)	public	Y	Y	Y	Y
(#)	protected	Y	Y	Y	N
(~)	*no modifier*	Y	Y	N	N
(-)	private	Y	N	N	N



Fonte: https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Elementos do Diagrama de Classes

As operações de uma classe representam o comportamento ou serviços que ela fornece.

Também possuem características importantes tais como escopo (visibilidade), tipo.

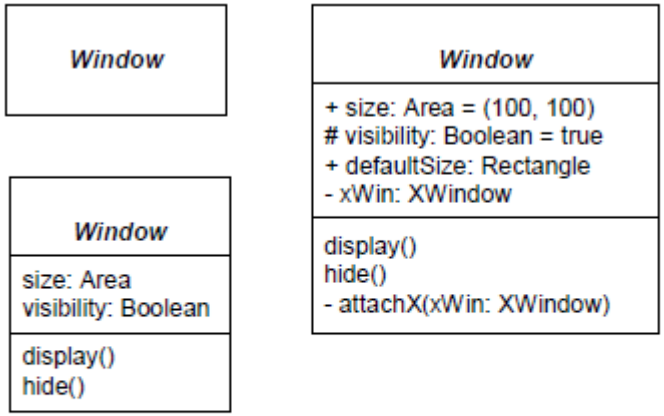


Figure 7.28 - Class notation: details suppressed, analysis-level details, implementation-level details

Fonte: UML Superstructure Specification, v2.3 – page 52

Elementos do Diagrama de Classes

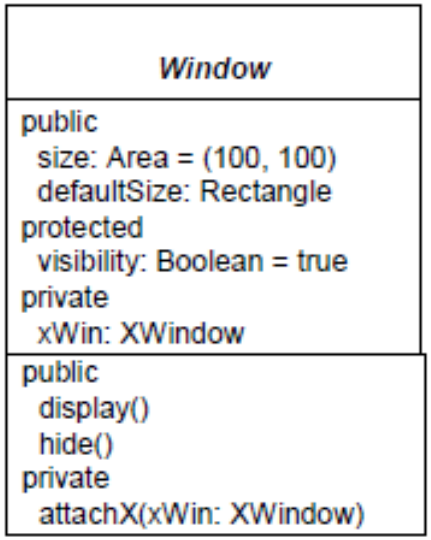
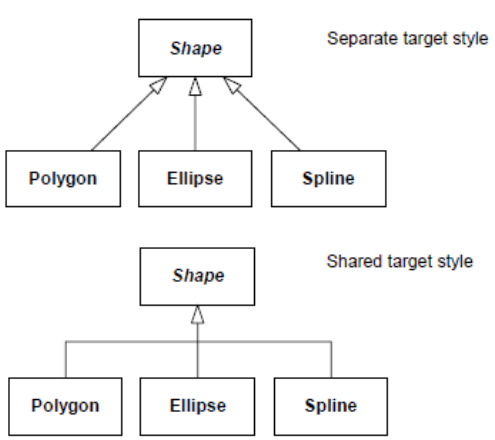


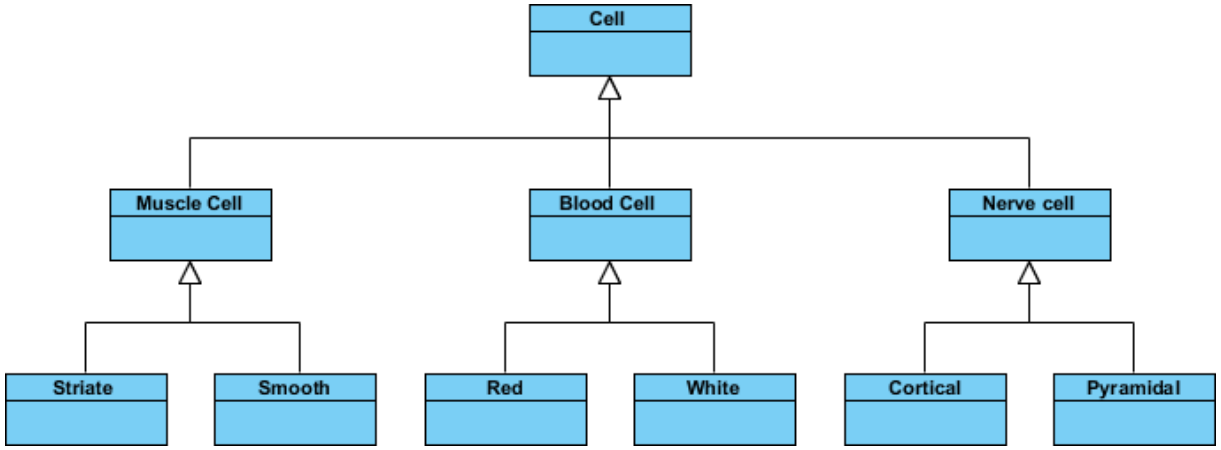
Figure 7.29 - Class notation: attributes and operations grouped according to visibility

Fonte: UML Superstructure Specification, v2.3 – page 52

Conectores principais

Herança (ou Generalização):



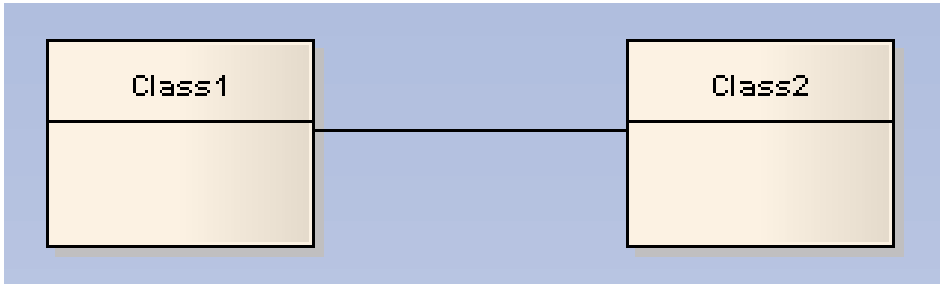


- Um conector do tipo **Generalização** é utilizado para indicar herança.
- A generalização relaciona uma classe específico para uma classe geral, portanto, cada instância da classe específico é também uma instância da classe geral.
- Dessa forma as características especificadas para as instâncias da classe geral são implicitamente especificadas para as instâncias da classe específica.

Fonte: UML Superstructure Specification, v2.3 – page 91 e Fonte: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

Conectores principais

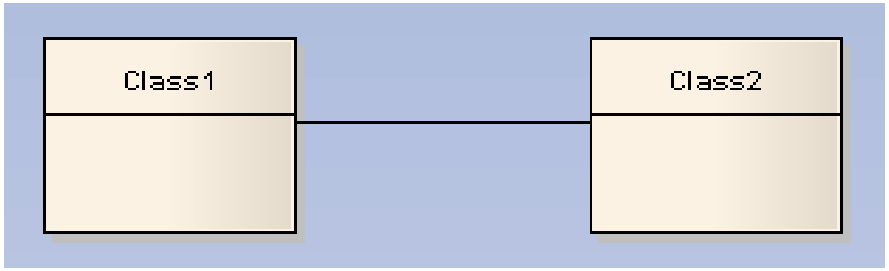
Associação:



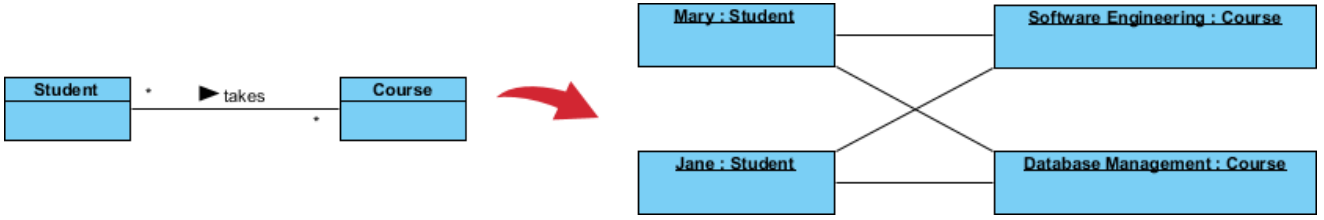
- Um conector do tipo **Associação** mostra que dois ou mais elementos do modelo possuem um relacionamento estrutural.
- Pode incluir nome de papéis no seu final, multiplicidade (exatamente 1, zero ou 1, muitos, um ou mais, quantidade exata etc), direção e restrição.
- Uma associação entre duas classes é normalmente implementada como uma variável de instância em uma classe.

Conectores principais

Associação:



- Um conector do tipo **Associação** mostra que dois ou mais elementos do modelo possuem um relacionamento estrutural.
- Pode incluir nome de papéis no seu final, multiplicidade (exatamente 1, zero ou 1, muitos, um ou mais, quantidade exata etc), direção e restrição.

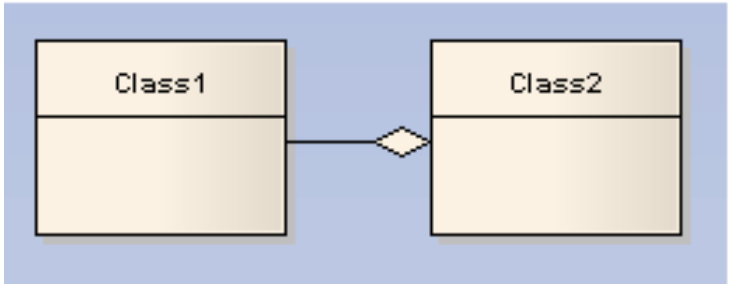


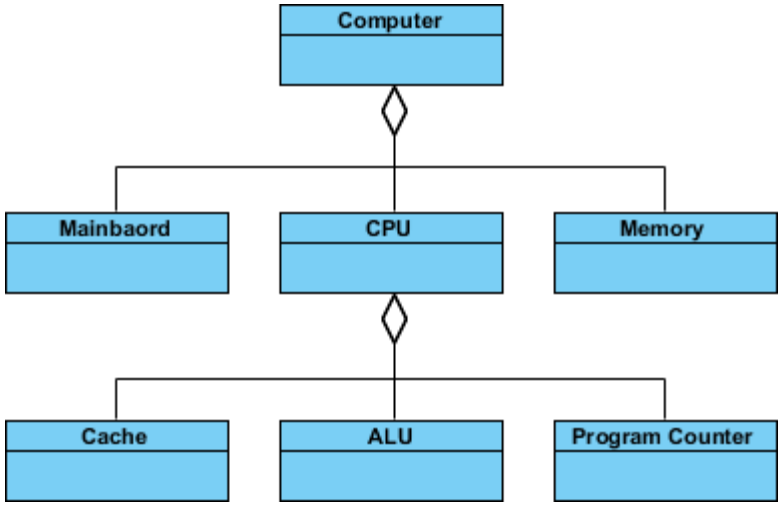
- Uma associação entre duas classes é normalmente implementada como uma variável de instância em uma classe.

Fonte: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

Conectores principais

Agregação:



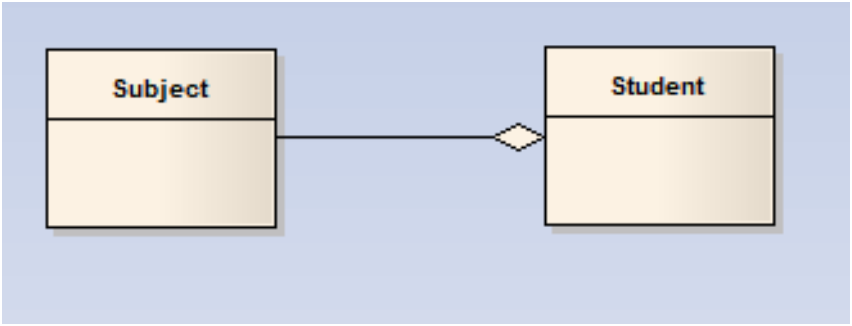


- Um conector do tipo **Agregação** é um tipo especial de associação que mostra que um elemento "é parte de" outro elementos (acima, Class2 é parte de Class1).
- A agregação é representada por uma linha com um losango em sua extremidade (apontada para o todo).
- Utilizada para indicar que um elemento mais complexo é construído a partir de uma coleção de elementos mais simples. Ex: Um Carro é composto por Rodas, Motor, e assim por diante.

Fonte: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

Conectores principais - Agregação

- Exemplo: suponha uma classe Student que armazena informações sobre cada estudante da escola. Agora suponha que exista uma classe Subject que armazena detalhes sobre um determinado assunto (por exemplo, história, geografia, etc).
- Se definirmos que a classe Student contém um objeto Subject, então pode-se dizer que o objeto Subject está contido no objeto Student (ou é parte-de).



Fonte: http://java.about.com/od/a/g/aggregation.htm

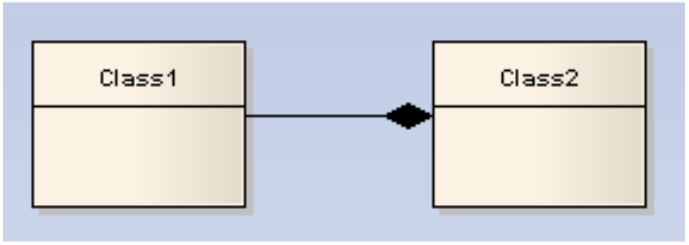
Conectores principais - Agregação

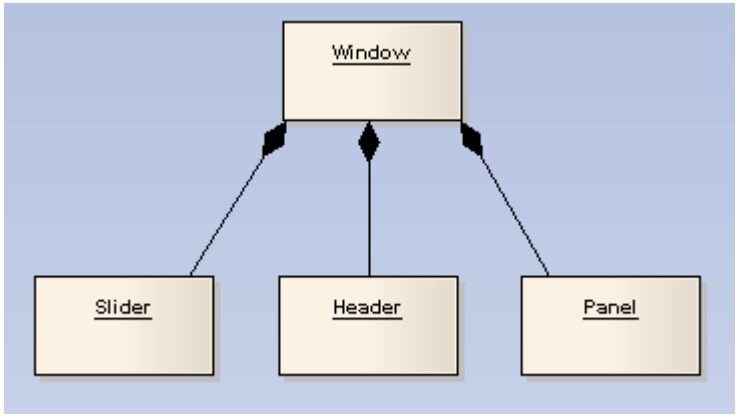
```
public class Subject {

  private String name;

  public void setName(String name)
  {
    this.name = name;
  }

  public String getName()
  {
    return name;
  }
}

public class Student {

  private Subject[] studyAreas = new Subject[10];

  //the rest of the Student class
}
```

Fonte: http://java.about.com/od/a/g/aggregation.htm

Conectores principais

Composição:

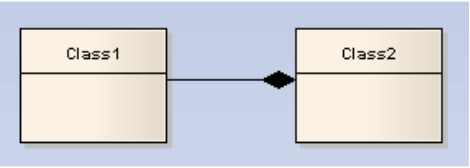




- Um conector do tipo **Composição** é um tipo especial de agregação na qual o todo é destruído quando todas suas partes são destruídas.
- A composição é representada por uma linha com um losango preenchido em sua extremidade (apontada para o todo).
- Se o Todo for excluído, geralmente todas as suas partes são eliminados com ele; no entanto, uma parte pode ser removida individualmente a partir de uma composição sem ter que apagar toda a composição.

Conectores principais - Composição

Exemplo: Suponha uma classe Student class que guarda informações sobre cada estudante da escola. Uma parte dessa informação armazenada é data de nascimento do estudante e para isso é utilizado um objeto da classe GregorianCalendar:

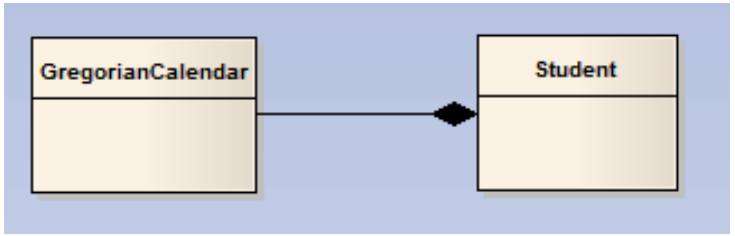



```
import java.util.GregorianCalendar;

public class Student {

  private String name;
  private GregorianCalendar dateOfBirth;

  public Student(String name, int day, int month, int year)
  {
    this.name = name;
    this.dateOfBirth = new GregorianCalendar(year, month, day);
  }

//rest of Student class..

}
```

Fonte: http://java.about.com/od/c/g/composition.htm

Conectores principais - Composição

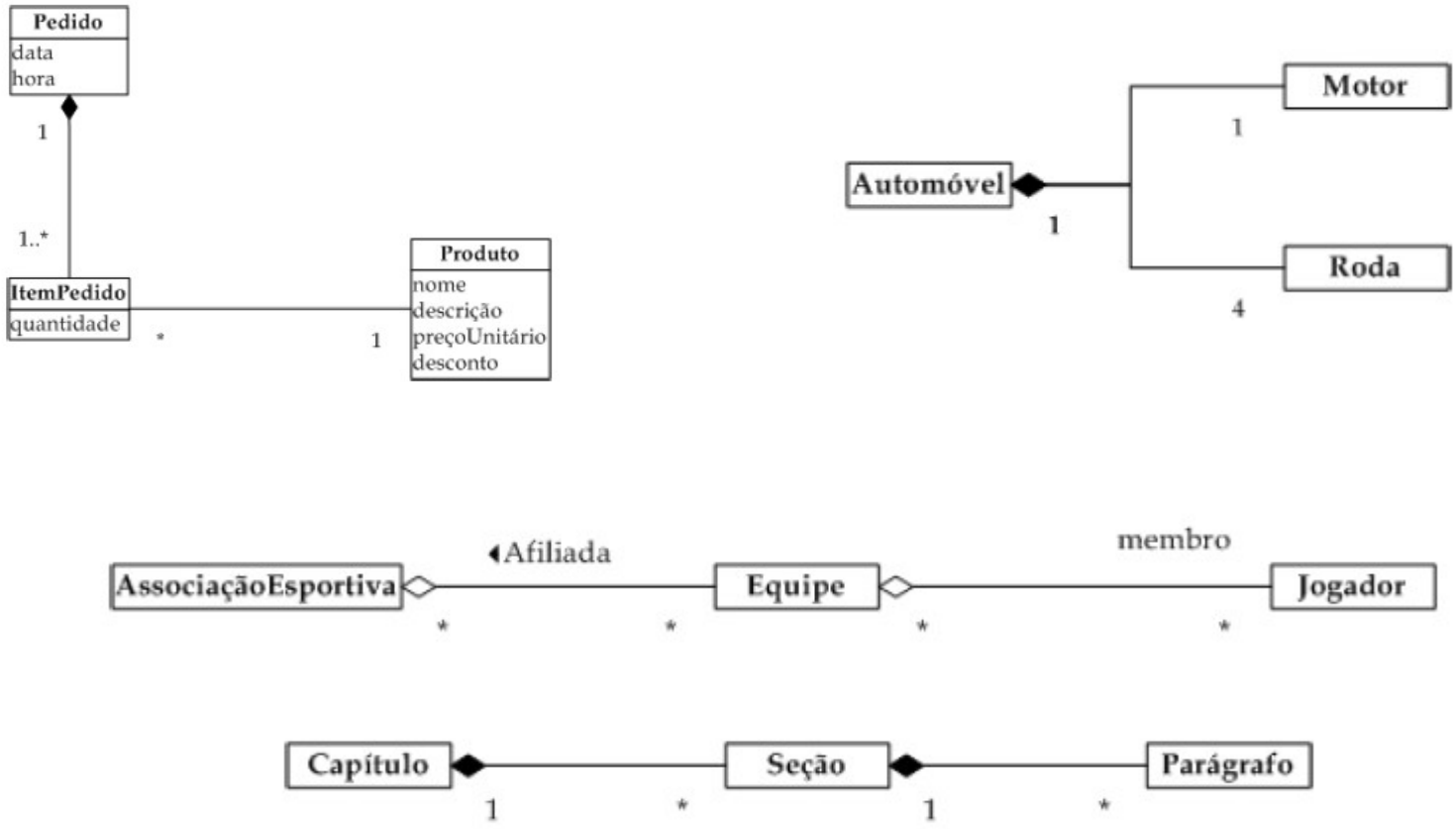
Observe que a classe Student é responsável pela criação e destruição do objeto GregorianCalendar (ou seja, se o objeto Student não existir também não existirá o objeto GregorianCalendar).




```
import java.util.GregorianCalendar;

public class Student {

  private String name;
  private GregorianCalendar dateOfBirth;

  public Student(String name, int day, int month, int year)
  {
    this.name = name;
    this.dateOfBirth = new GregorianCalendar(year, month, day);
  }

//rest of Student class..

}
```

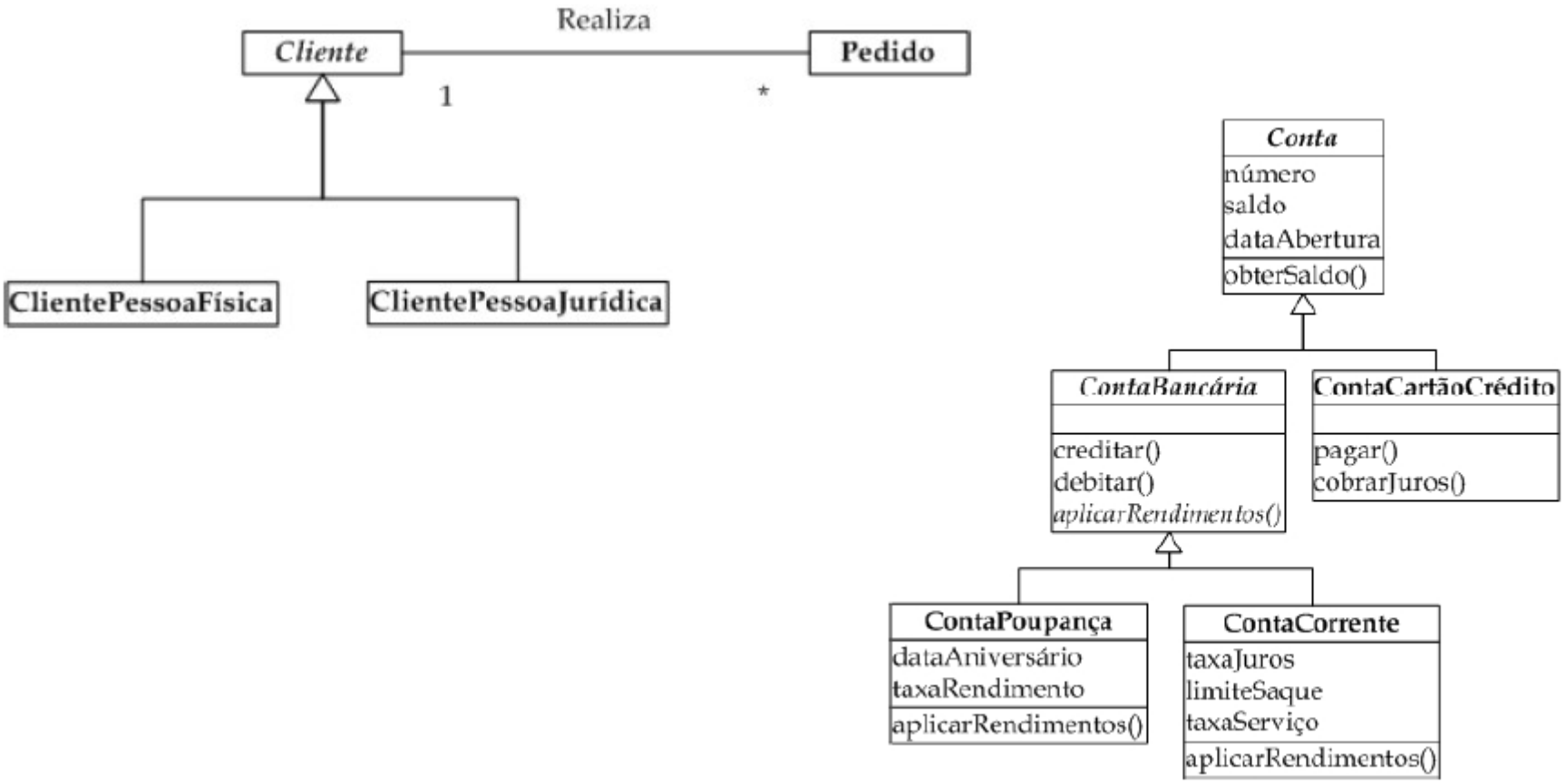
Fonte: http://java.about.com/od/c/g/composition.htm

Exemplo 1



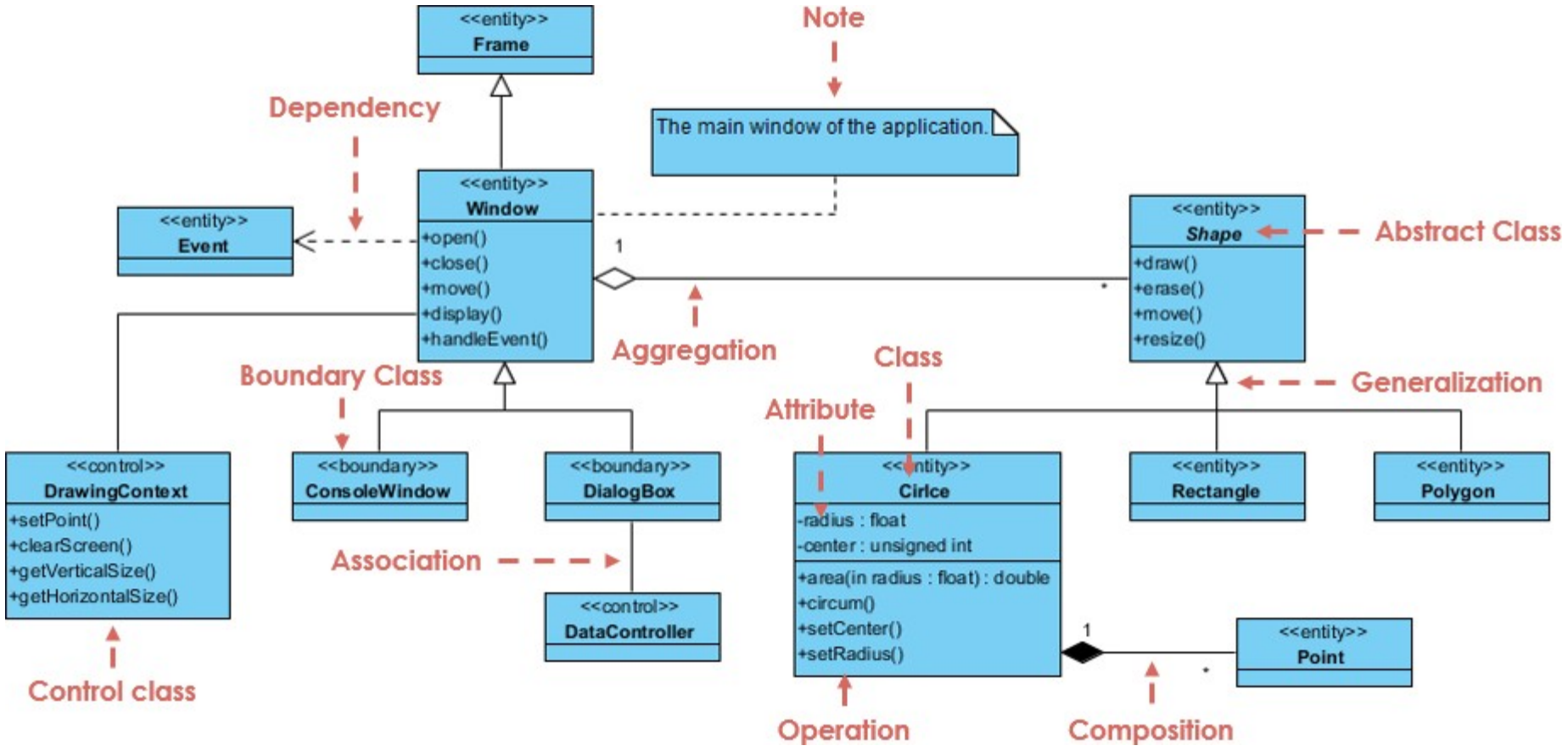
Fonte: Princípios de Análise e Projeto de Sistemas com UML - 2ª edição

Exemplo 2



Fonte: Princípios de Análise e Projeto de Sistemas com UML - 2ª edição

Exemplo 3



Fonte: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/


UNIFIED
MODELING
LANGUAGE
TM

Diagrama de Classes

Como identificar as classes
que irão compor um sistema?

Encontrando as classes do sistema...

A partir da análise do fluxo de cada Use Case é possível identificar **classes de análise**, ou seja, que poderão vir a ser as classes de "projeto" do sistema;

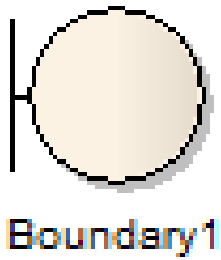
Com as classes de análise, inicia-se o processo de **distribuição do comportamento da use case em classes**, que inicialmente pode apresentar uma visão macro, e após repetidas análises, uma visão bem detalhada;

Dessa forma, **classes de análise representam uma abstração de uma ou mais classes de projeto**;

As classes de análise podem ser categorizadas conforme os seguintes estereótipos da UML: <<boundary>>, <<entity>>, <<control>>

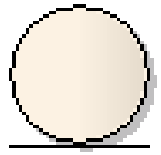
Classes de Análise

- Classe de Fronteira: <<boundary>>
  - Responsável pela comunicação entre Caso de Uso e Ator.
  - Troca de informações entre Caso de Uso e Ator.
  - Geralmente associada a janelas, formulários, sensores, terminais, APIs, ...



Classes de Análise

- Classe de Entidade: <<entity>>
  - Responsável por manter ou persistir as informações.



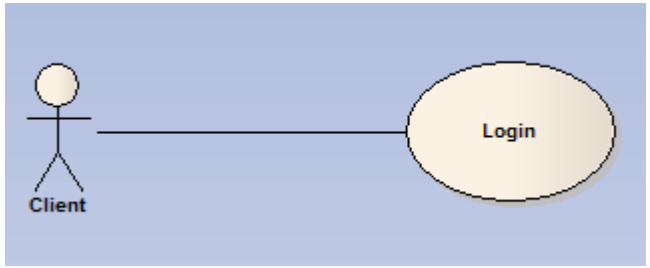
- Classe de Controle: <<control>>
  - Atuam como uma “ponte de comunicação” entre objetos de fronteira e objetos de entidade.
  - Responsável pela lógica de execução da Use Case.

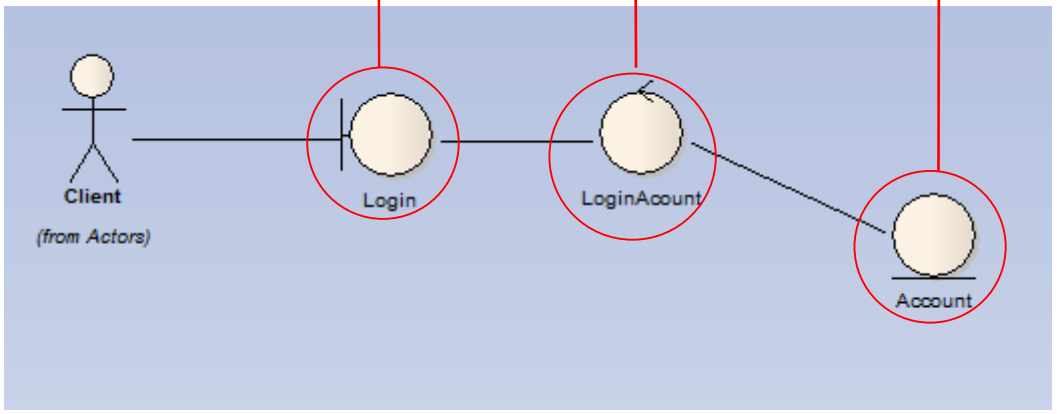
Exemplo 1

<<boundary>>
Responsável pela comunicação entre Caso de Uso e Ator.
Geralmente associada a janelas, formulários, APIs, etc.

<<control>>
Atuam como uma "ponte" entre fronteira e entidade.
Responsável pela lógica de execução da Use Case.

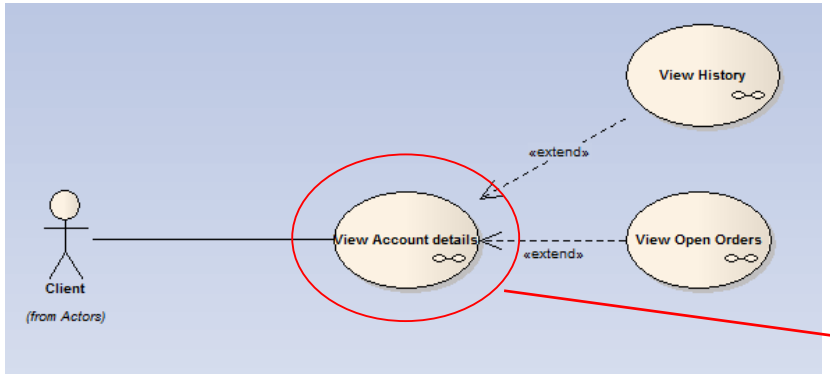
<<entity>>
Responsável por manter
ou persistir as informações.

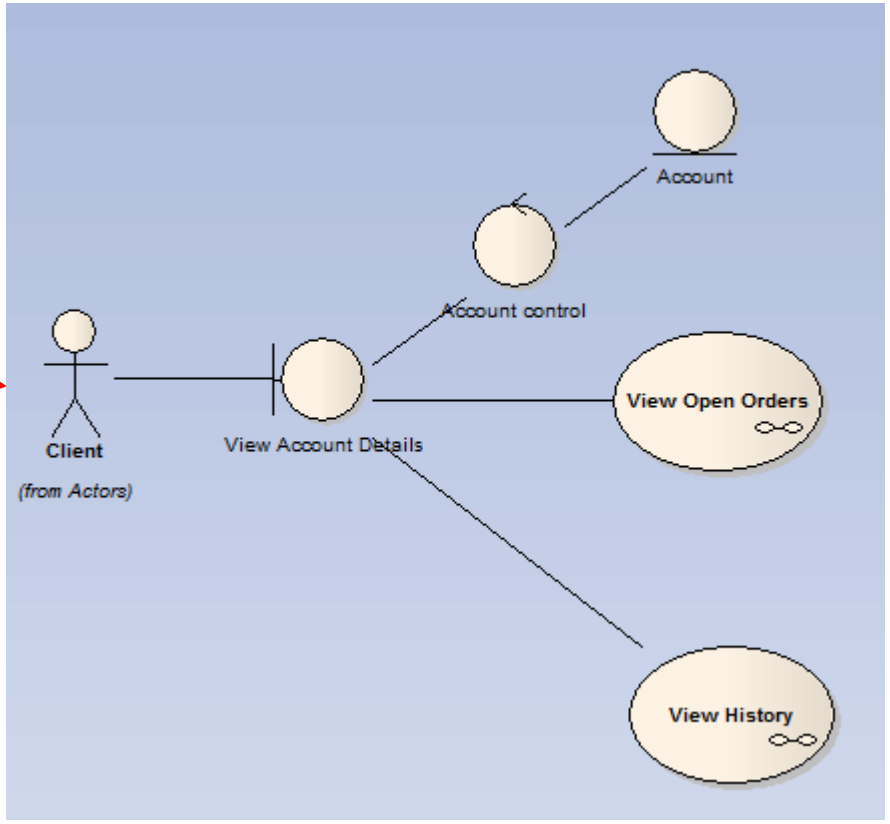




Fonte: Enterprise Architect

Exemplo 2 – parte 1





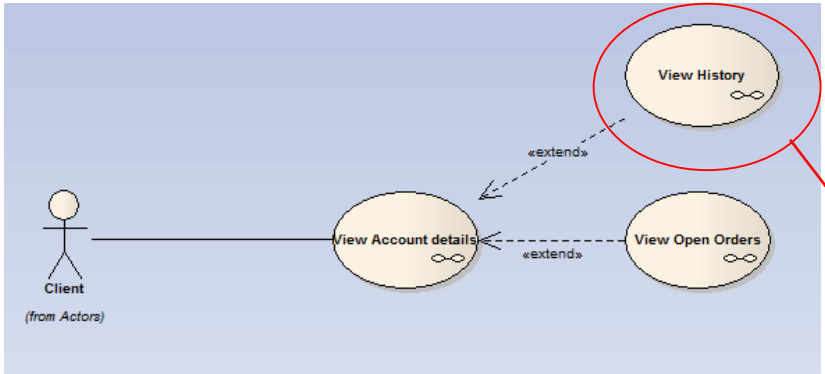
Basic Path: This use case begins when a Client requests a display of their account details. General account information is displayed (i.e. ID, username, billing address, delivery address etc). Command buttons are displayed to allow the user to view Open Orders or History. The use case terminates when the user selects the Exit or Back button.

View Open Orders
If the View Open Orders command is selected, the View Open Orders use case is executed.

View History
If the user selects the View History command, the View History use case is executed.

Fonte: Enterprise Architect

Exemplo 2 – parte 2



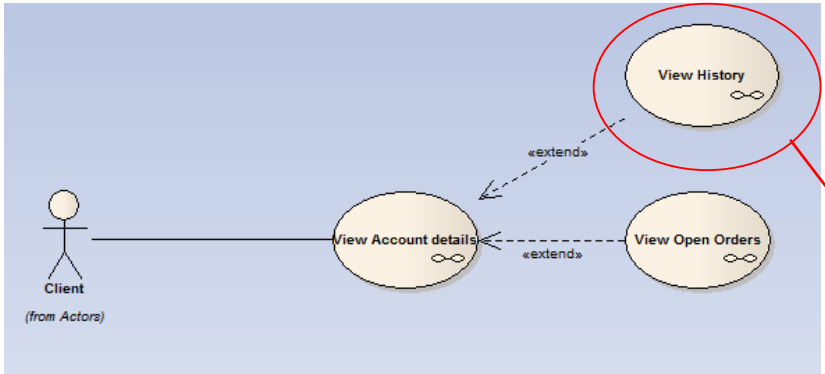
No History
If the database search finds no previous transactions, then you don't display a sorted list of nothing: instead you put a message saying nothing was found.
Note there is no need to put up an error message dialog.

Basic Path
This use case begins when the user requests to view a history of transactions against their account. The account ID is used as a key to lookup the appropriate records in the database. The results are then displayed sorted in date order.

Vamos tentar?

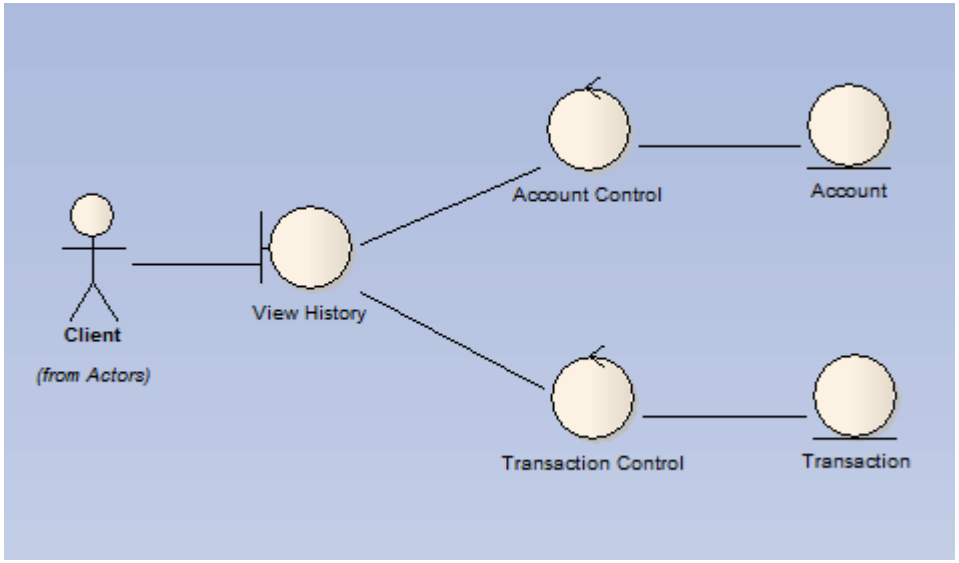
Fonte: Enterprise Architect

Exemplo 2 – parte 2



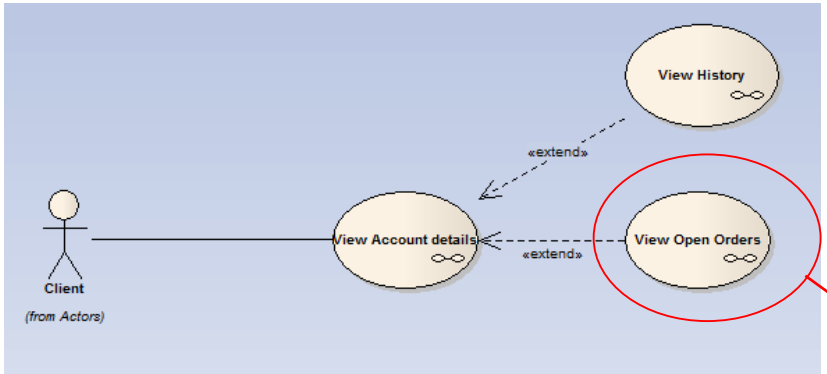
No History
If the database search finds no previous transactions, then you don't display a sorted list of nothing: instead you put a message saying nothing was found.
Note there is no need to put up an error message dialog.

Basic Path
This use case begins when the user requests to view a history of transactions against their account. The account ID is used as a key to lookup the appropriate records in the database. The results are then displayed sorted in date order.



Fonte: Enterprise Architect

Exemplo 2 – parte 3



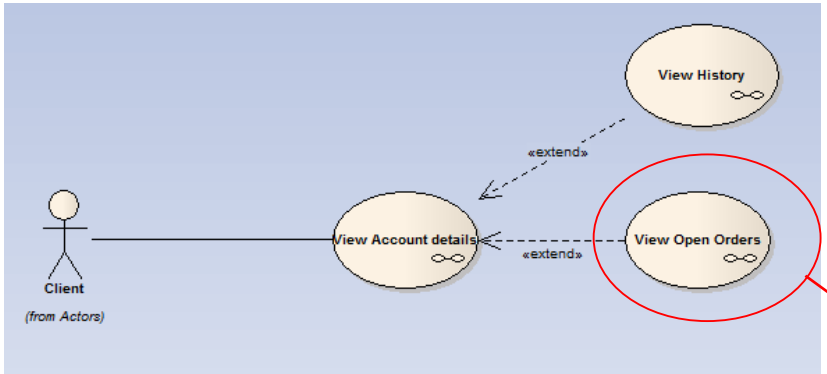
No Current Transactions
If the database search finds no current transactions, then you don't display a sorted list of nothing: instead you put a message saying nothing was found. Note there is no need to put up an error message dialog.

Basic Path
This use case begins when the user request to view a list of current transactions against their account. The account ID is used as a key to lookup the appropriate records in the database. The results are then displayed sorted in date order.

Vamos tentar?

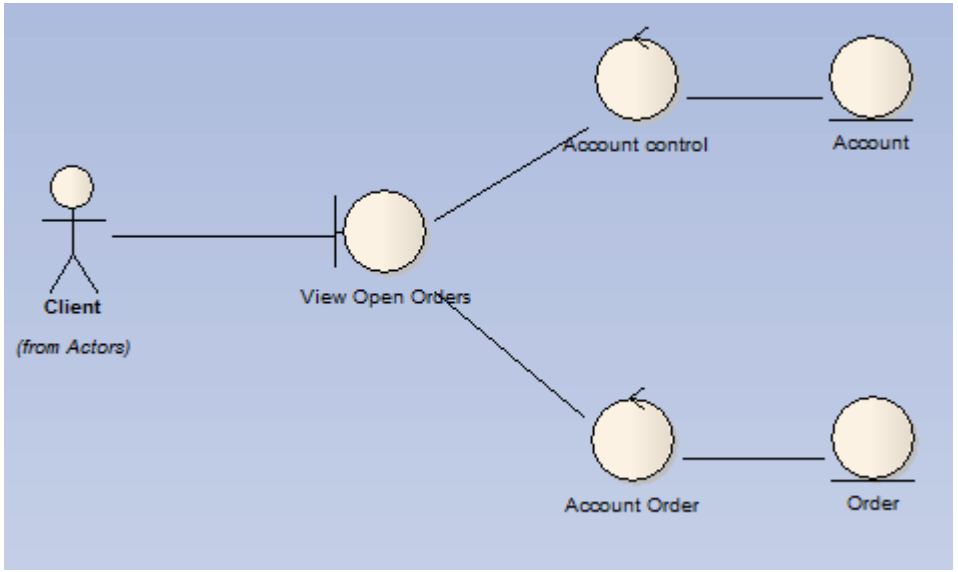
Fonte: Enterprise Architect

Exemplo 2 – parte 3



No Current Transactions
If the database search finds no current transactions, then you don't display a sorted list of nothing: instead you put a message saying nothing was found. Note there is no need to put up an error message dialog.

Basic Path
This use case begins when the user request to view a list of current transactions against their account. The account ID is used as a key to lookup the appropriate records in the database. The results are then displayed sorted in date order.



Fonte: Enterprise Architect


UNIFIED
MODELING
LANGUAGE
TM

Diagrama de Sequência

Um diagrama de sequência é uma representação do comportamento do sistema como uma **série de etapas sequenciais ao longo do tempo**.

Ele é usado para descrever o fluxo de trabalho, a passagem de mensagens e como os elementos em geral cooperam ao longo do tempo para alcançar um resultado.

Cada elemento é organizado em uma sequência horizontal, com mensagens que passam para trás e para frente entre os elementos.

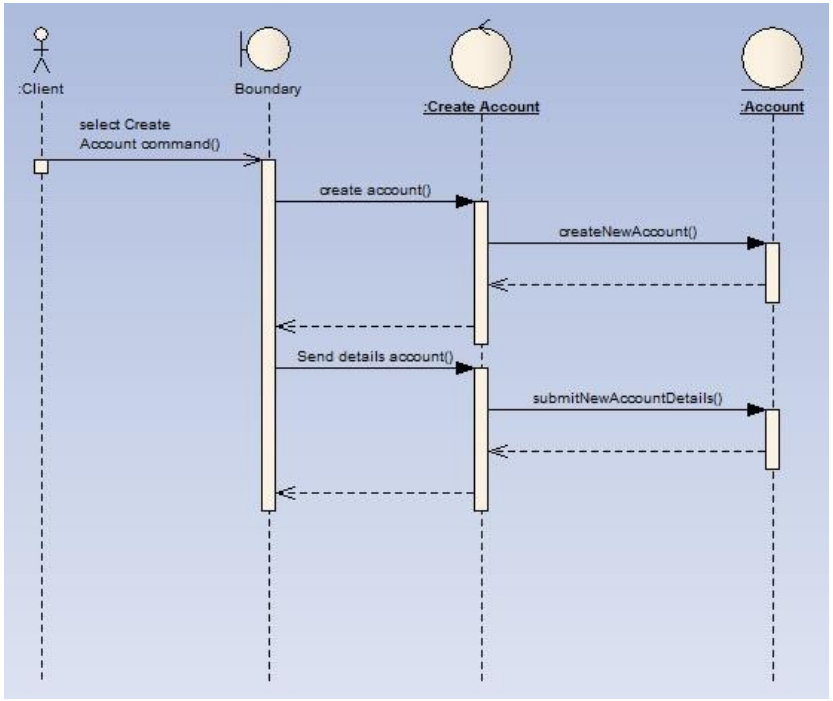
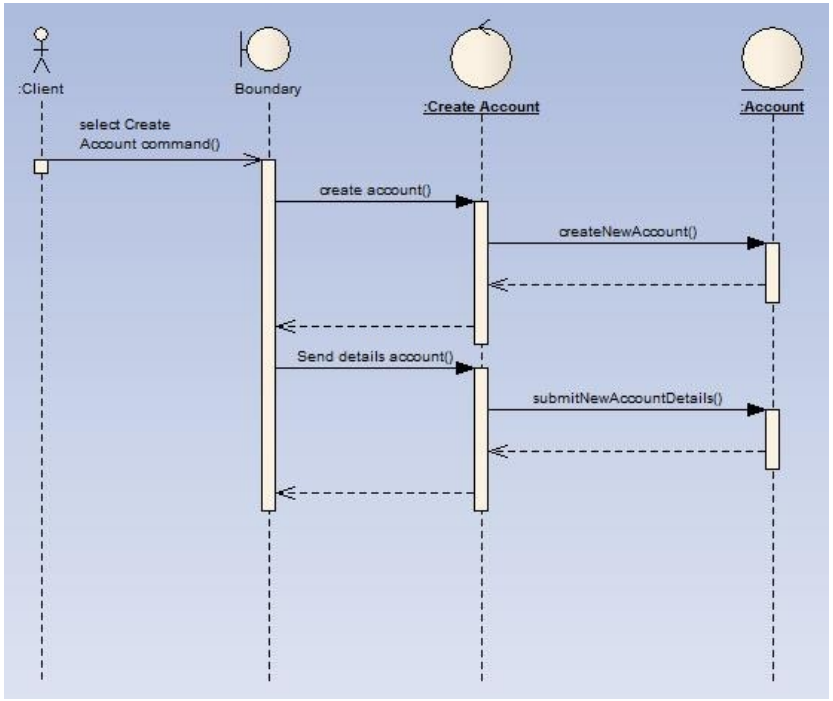


Diagrama de Sequência

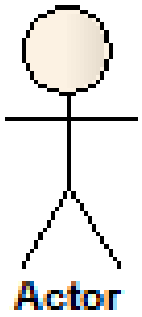
Um elemento ator pode ser usado para representar o usuário que inicia o fluxo de eventos.

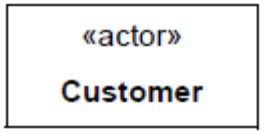
Elementos estereotipados, como <<boundary>>, <<control>> e <<entity>> podem ser usados para ilustrar as telas, os controles e os itens de banco de dados, respectivamente.

Diagramas de sequência são comumente usados como modelos explicativos dos cenários das Use Cases.



Elementos do diagrama de Sequência





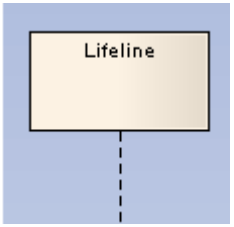
Ator

Representado por um boneco, um retângulo (com o estereótipo <<actor>> ou um ícone.

Um ator é um usuário do sistema: o usuário pode ser uma pessoa, uma máquina, ou mesmo outro sistema.

Qualquer coisa que interaja com o sistema a partir de sua fronteira é chamado de ator.

Elementos do diagrama de Sequência

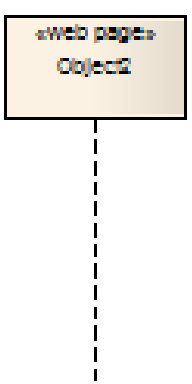


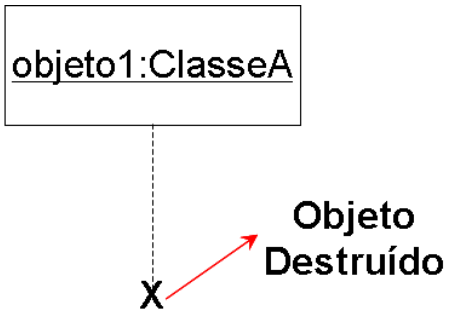
Linha de vida

Representa um participante individual na interação.

Podem ser elementos estereotipados (boundary, control, entity).

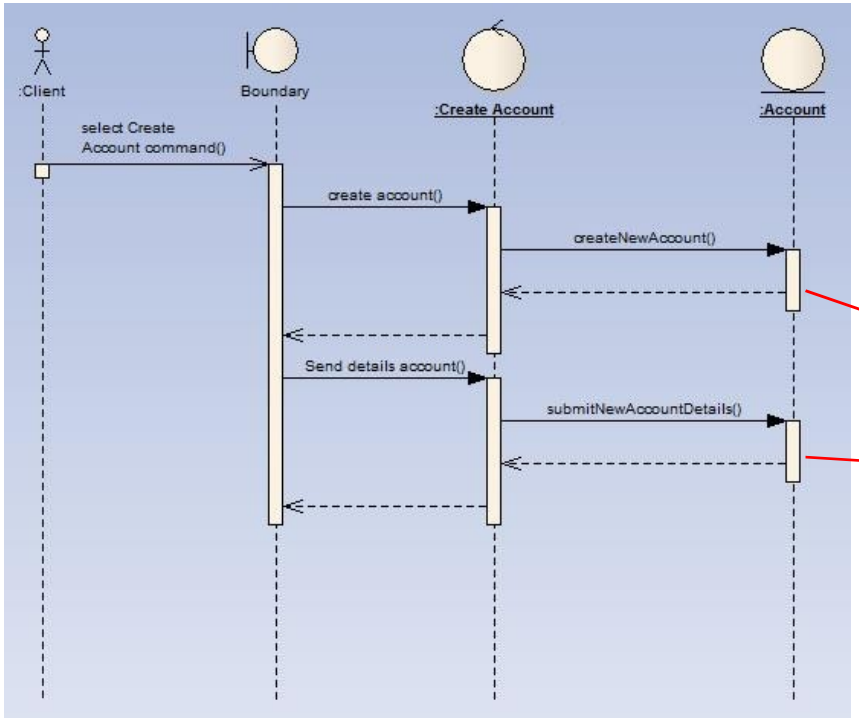
Objetos destruídos têm sua linha de vida interrompida por um “X”.





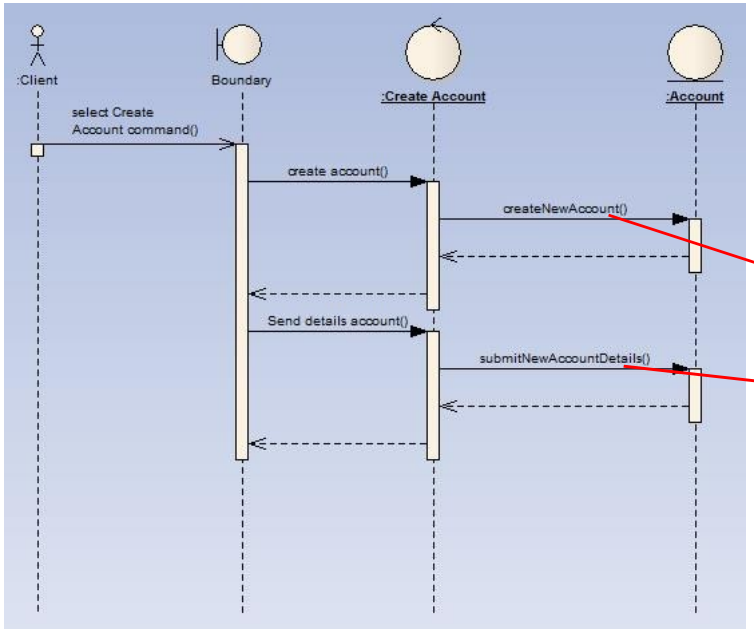
Elementos do diagrama de Sequência

- Linha de vida: **Foco de Controle (Ativação)**
  - Indica os períodos em que um determinado objeto está participando ativamente do processo (executando algum método).



Elementos do diagrama de Sequência

- **Mensagem**
  - Define a comunicação entre os elementos em uma interação;
  - A comunicação pode ser, de exemplo, invocar uma operação, criar ou destruir uma Instância.



Mensagens

Diagrama de Sequência

- **Mensagens**

  – Retorno:
    - Respondem a uma mensagem de estímulo recebida.
    - Podem retornar informações específicas do método chamado ou valores.

  – Assíncronas:
    - Não precisam esperar resposta do objeto chamado para continuar o processamento.
    - O objeto que disparou continua com o Foco de Controle.

Diagrama de Sequência

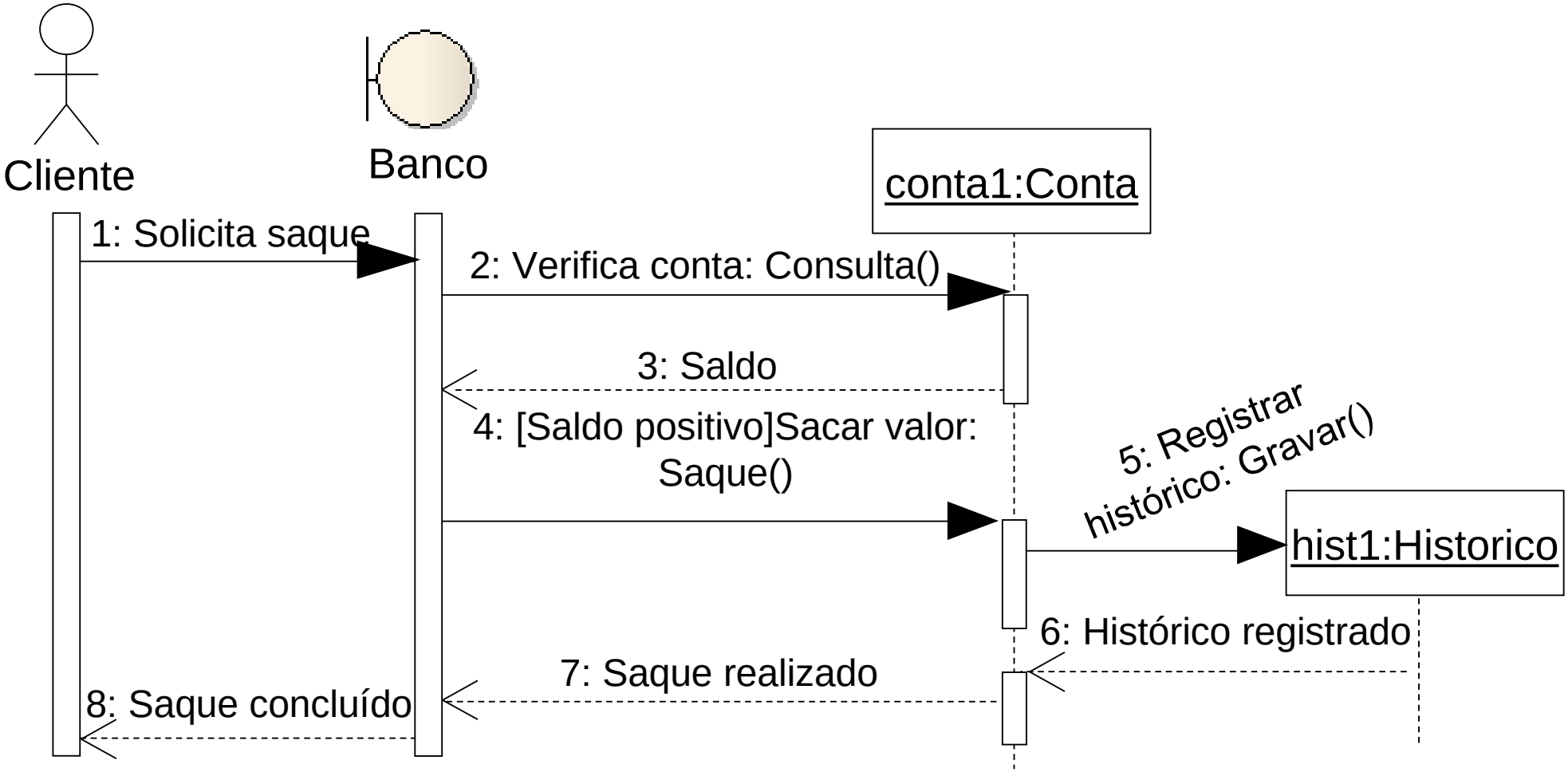


Diagrama de Sequência

Mensagens

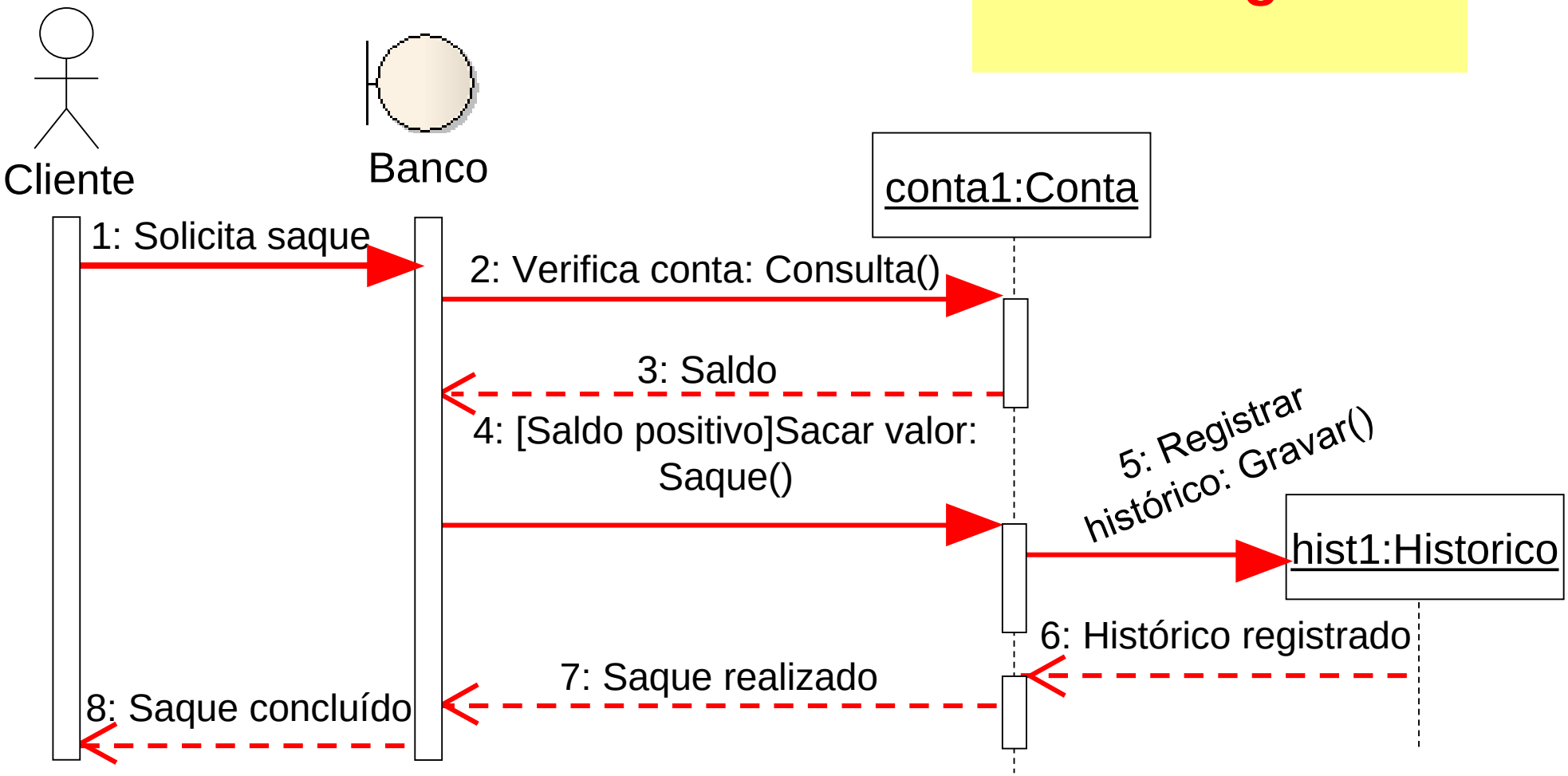


Diagrama de Sequência

Mensagens de Retorno

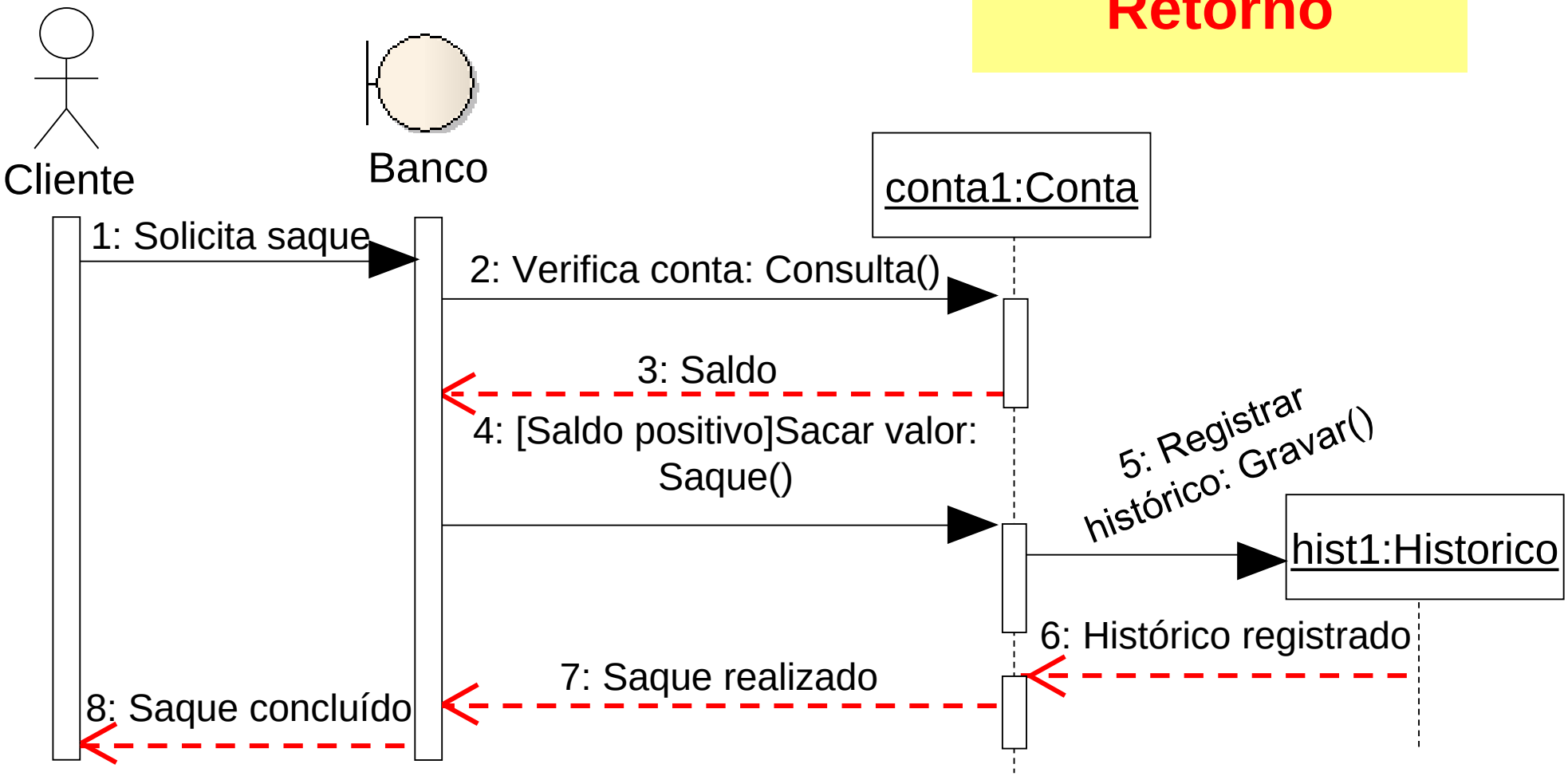


Diagrama de Sequência

Objeto criado durante o processo

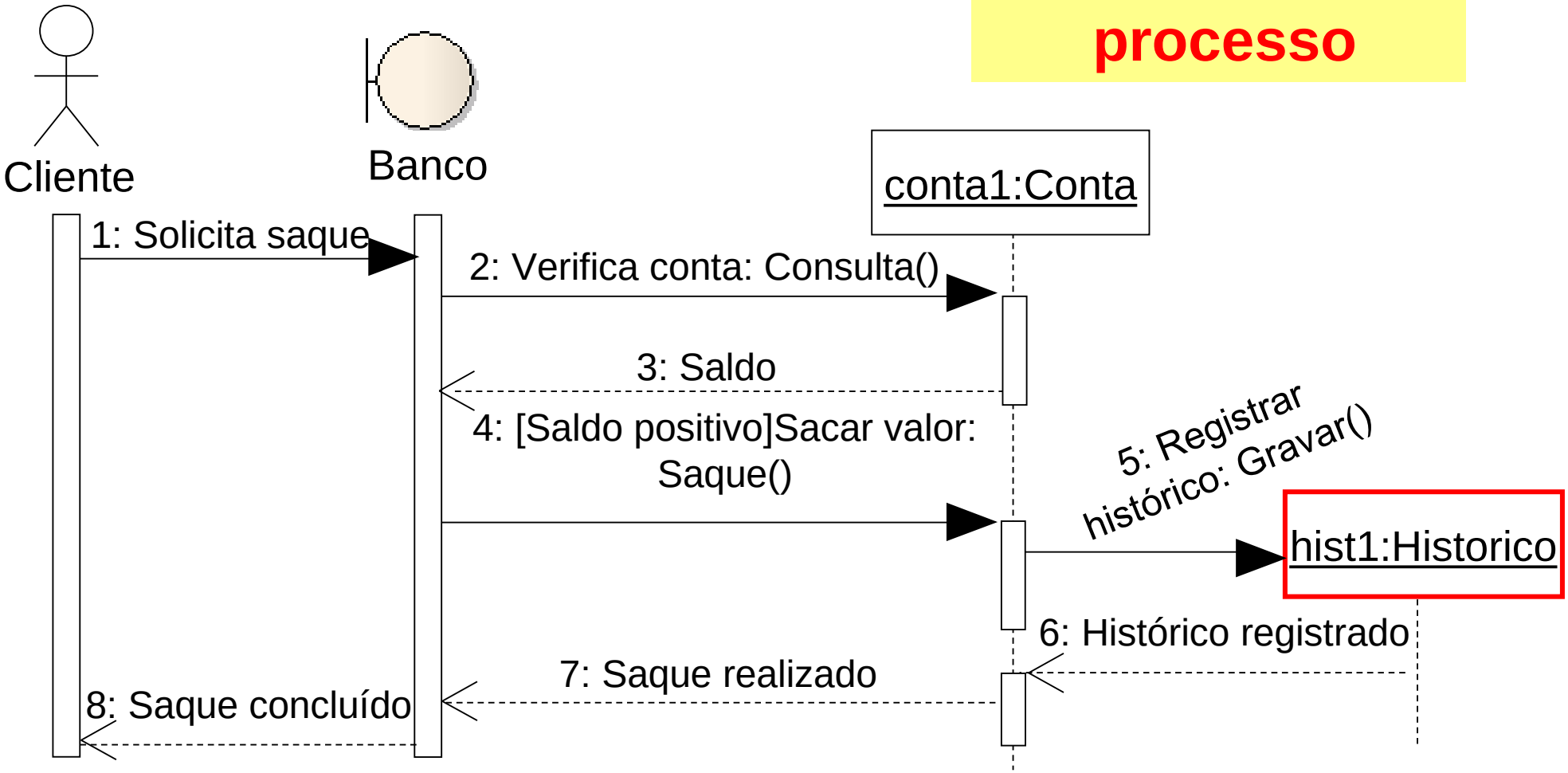


Diagrama de Sequência

- Auto-mensagem
  - Mensagens que um objeto envia a si mesmo.

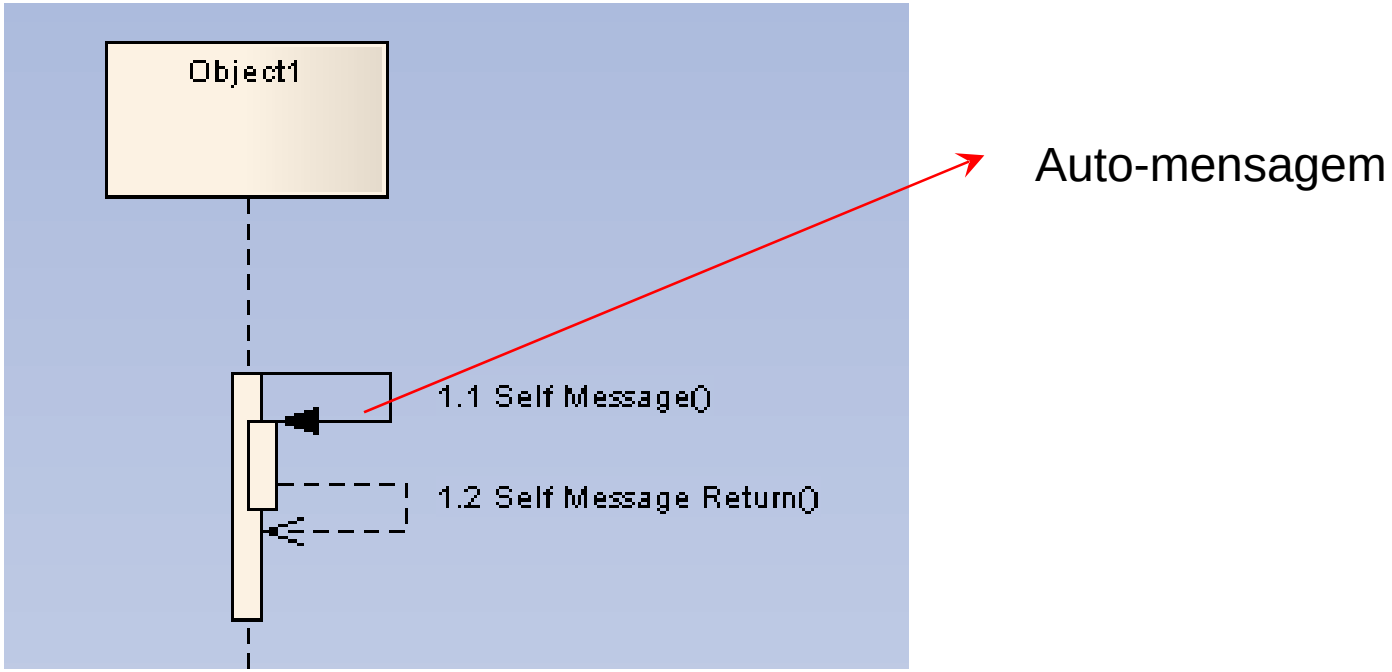


Diagrama de Sequência

- Condição de Guarda
  - Indica que uma dada mensagem só poderá ser enviada se uma determinada condição for verdadeira.

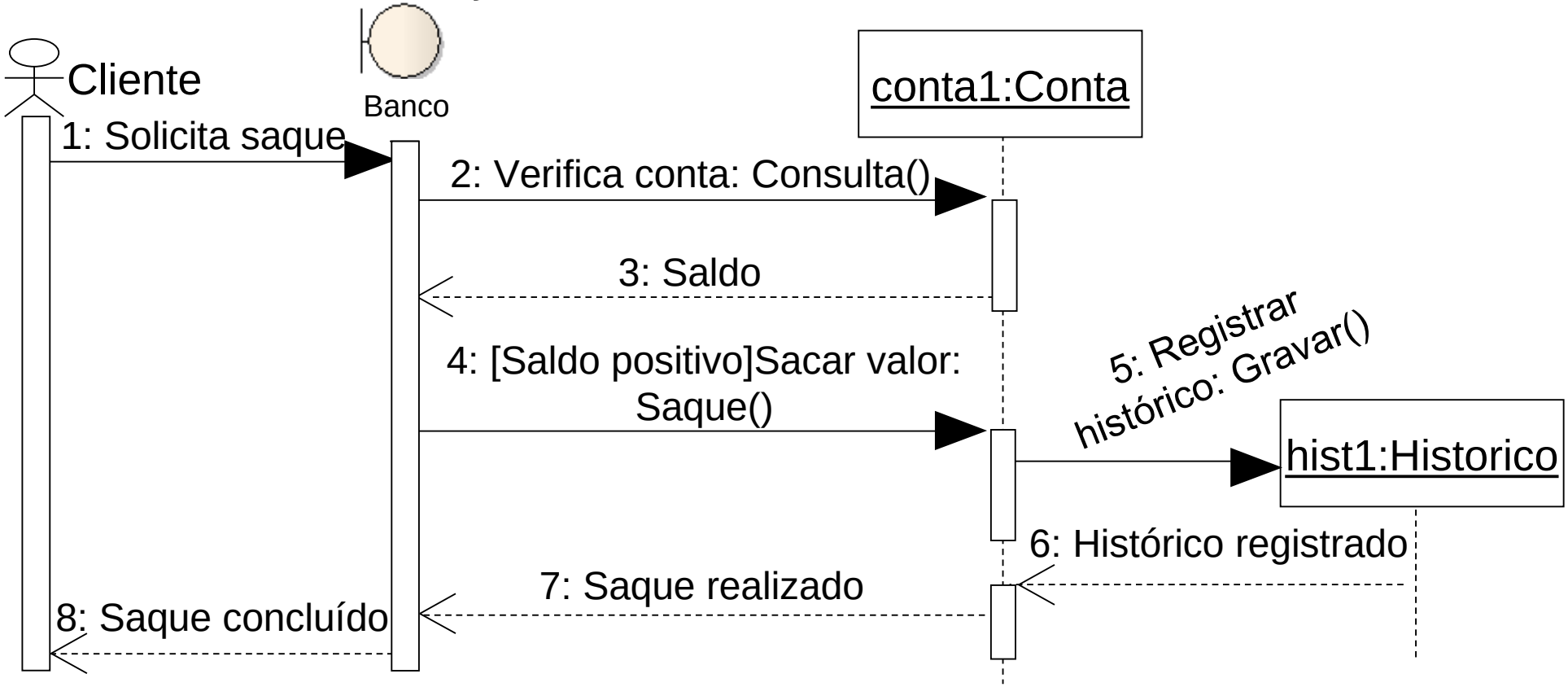


Diagrama de Sequência

Condição de Guarda

- Condição de Guarda
  - Indica que uma dada mensagem só poderá ser enviada se uma determinada condição for verdadeira.

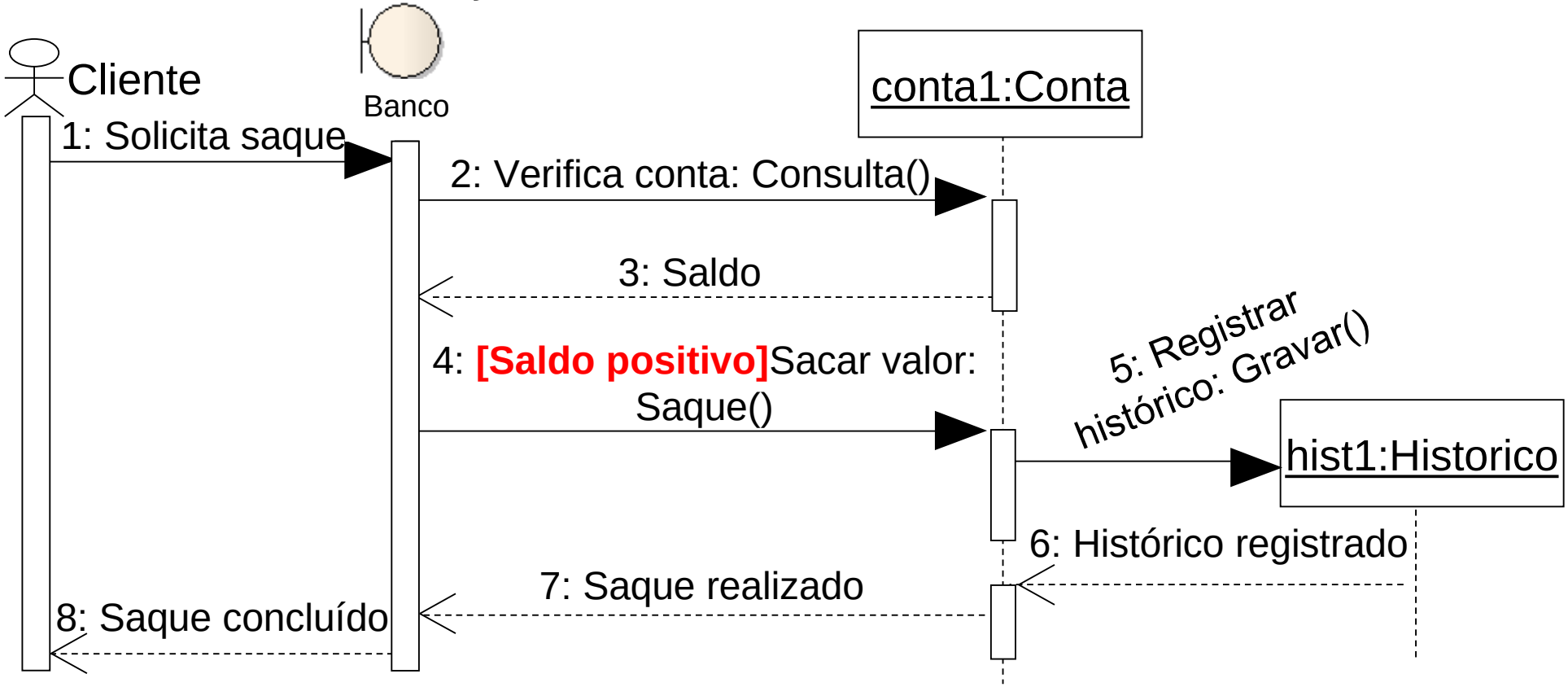
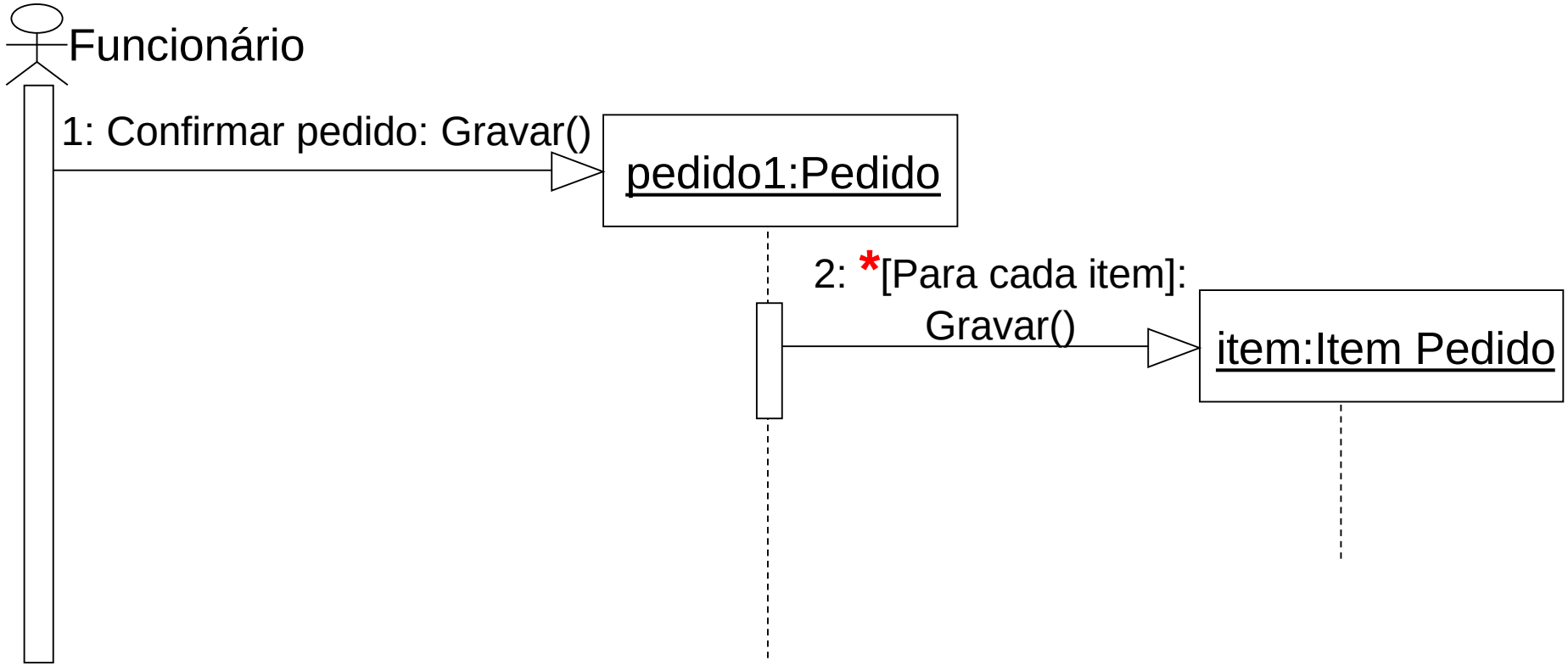
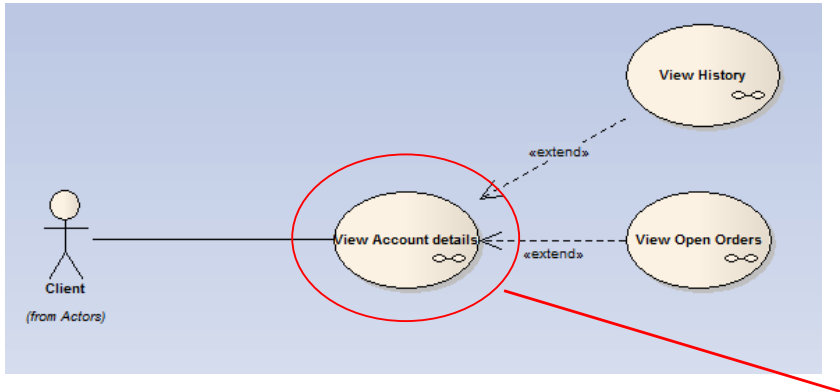


Diagrama de Sequência

- Condição de Guarda
  - Múltiplas ações: representa o disparo de uma mensagem a vários objetos.



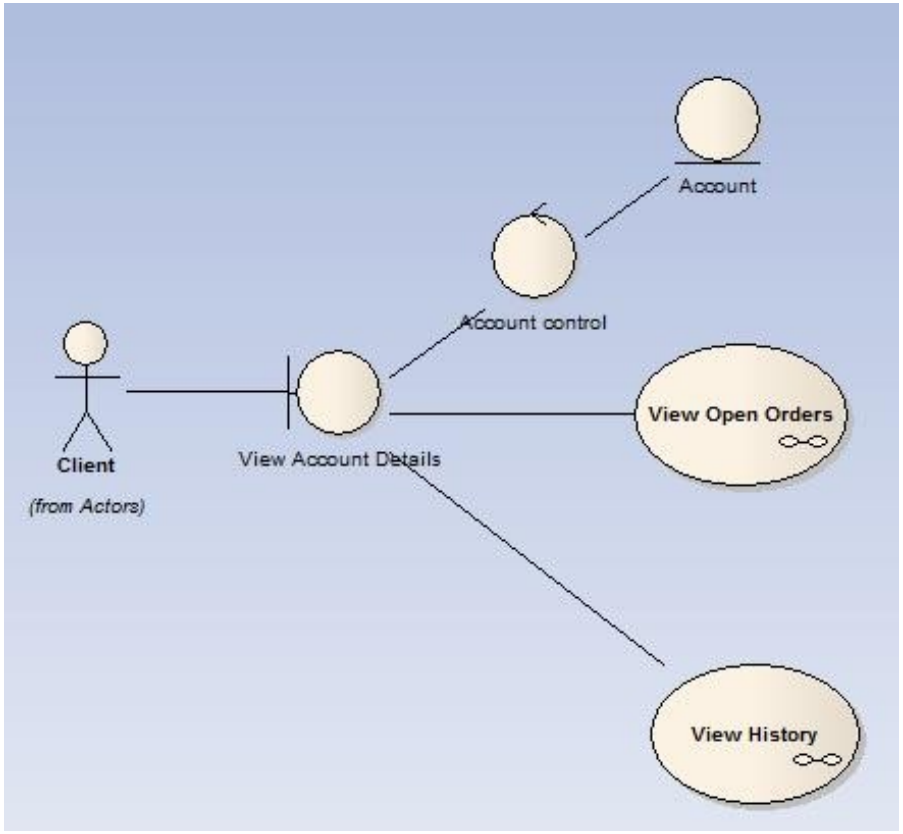
Exemplo 1 – View Account Details



Basic Path
This use case begins when a Client requests a display of their account details. General account information is displayed (i.e. ID, username, billing address, delivery address etc). Command buttons are displayed to allow the user to view Open Orders or History. The use case terminates when the user selects the Exit or Back button.

View Open Orders
If the View Open Orders command is selected, the View Open Orders use case is executed.

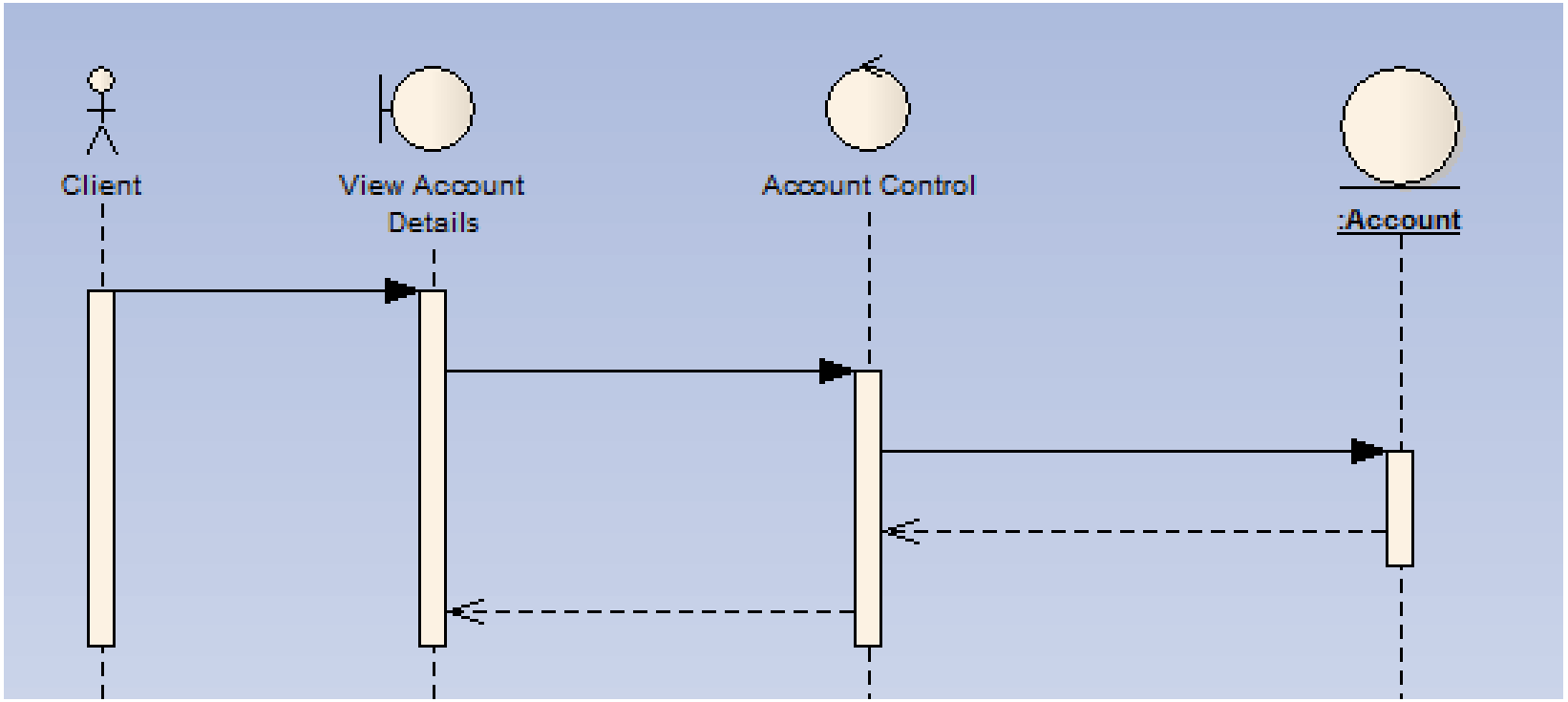
View History
If the user selects the View History command, the View History use case is executed.



Fonte: Enterprise Architect

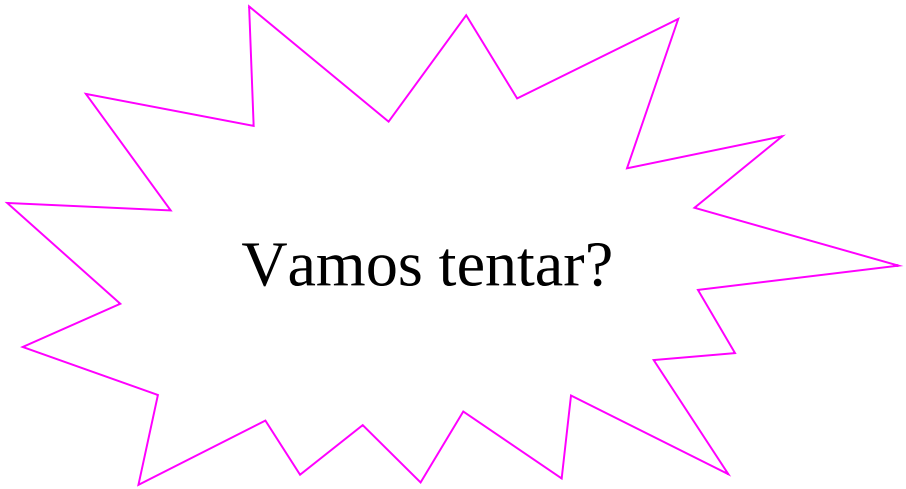
Exemplo 1 – Fluxo Básico

Basic Path: This use case begins when a Client requests a display of their account details. General account information is displayed (i.e. ID, username, billing address, delivery address etc).



Exemplo 1 – View Open Orders

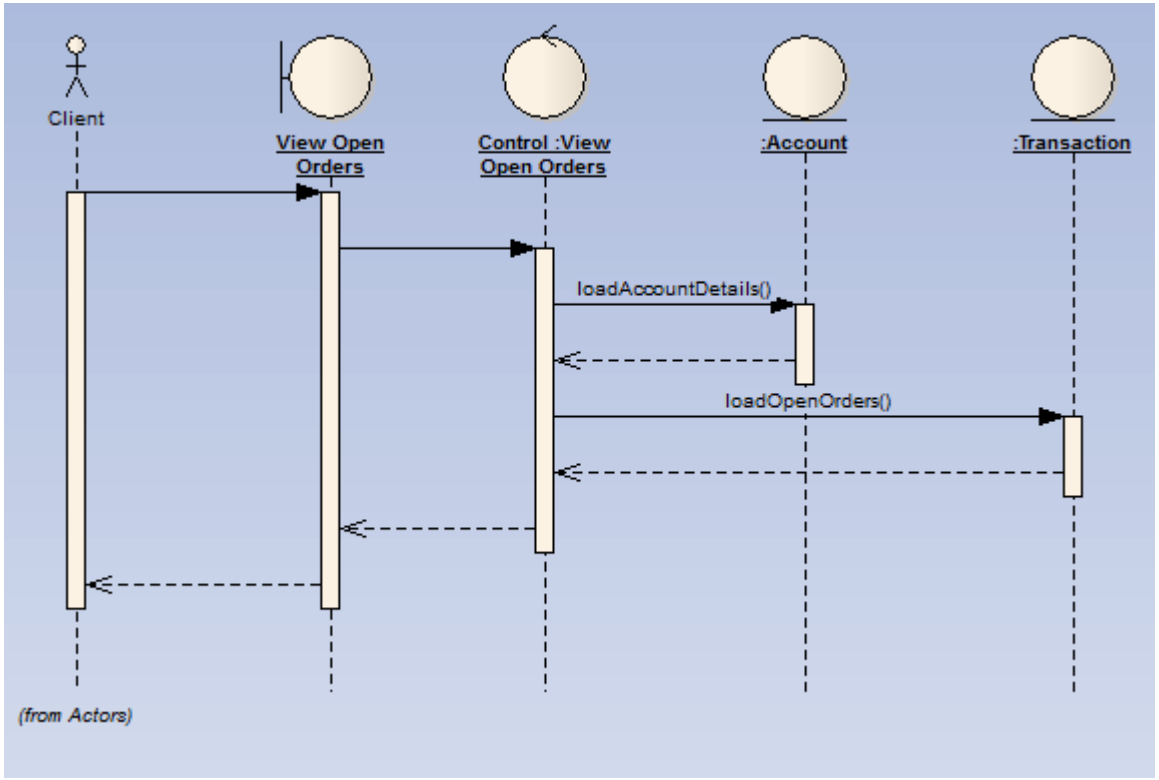
Basic Path: This use case begins when the user request to view a list of current transactions against their account. The account ID is used as a key to lookup the appropriate records in the database. The results are then displayed sorted in date order.



Fonte: Enterprise Architect

Exemplo 1 – View Open Orders

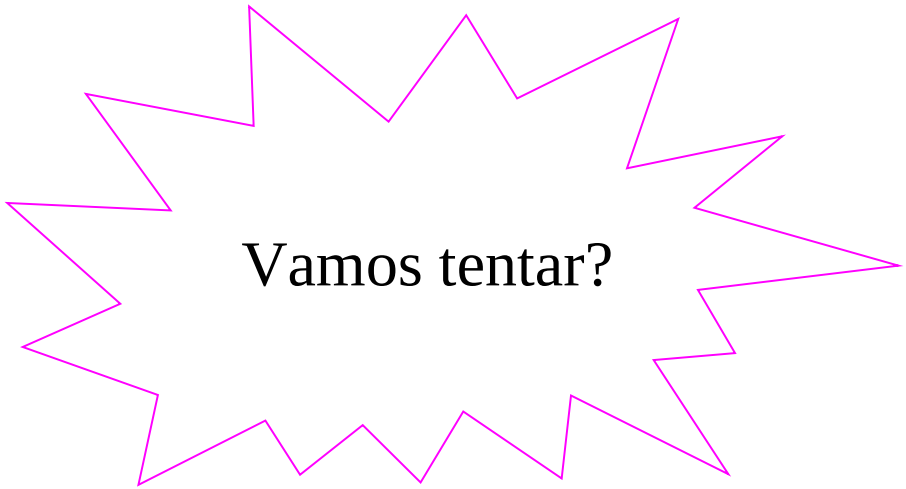
Basic Path: This use case begins when the user request to view a list of current transactions against their account. The account ID is used as a key to lookup the appropriate records in the database. The results are then displayed sorted in date order.



Fonte: Enterprise Architect

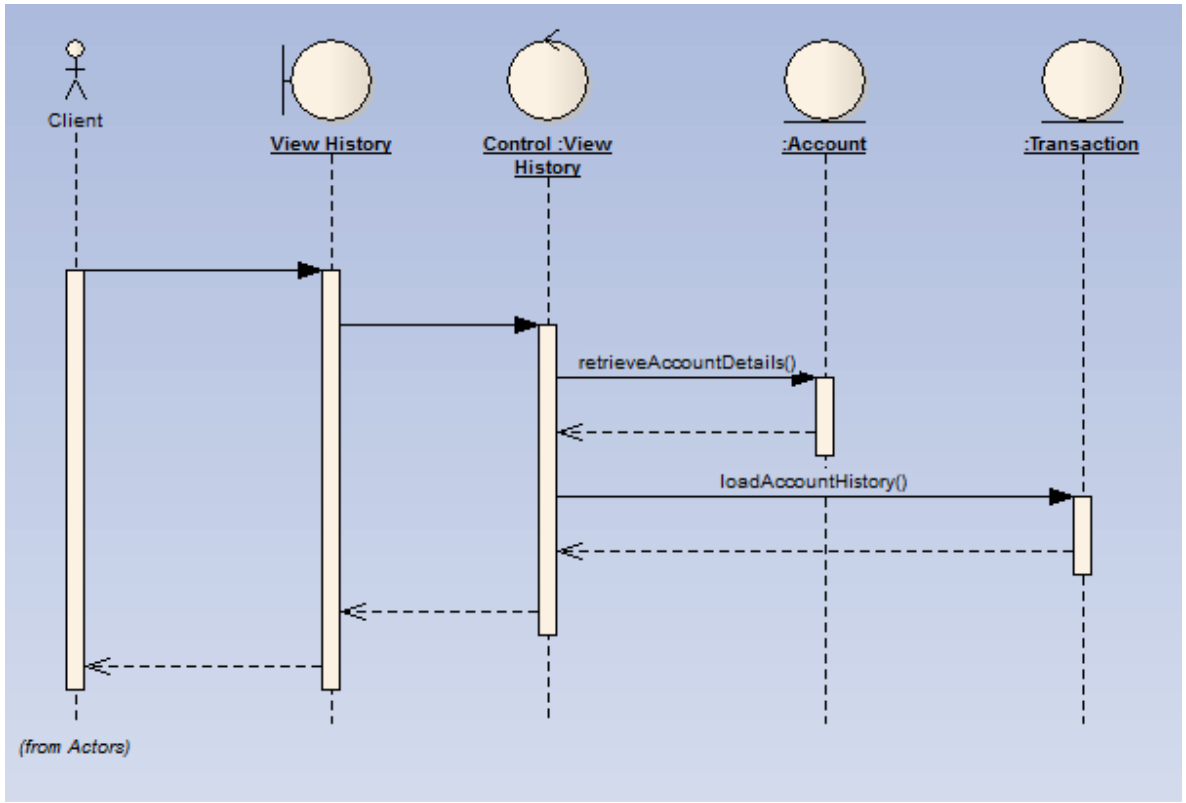
Exemplo 1 – View History

Basic Path: This use case begins when the user requests to view a history of transactions against their account. The account ID is used as a key to lookup the appropriate records in the database. The results are then displayed sorted in date order.



Exemplo 1 – View History

Basic Path: This use case begins when the user requests to view a history of transactions against their account. The account ID is used as a key to lookup the appropriate records in the database. The results are then displayed sorted in date order.

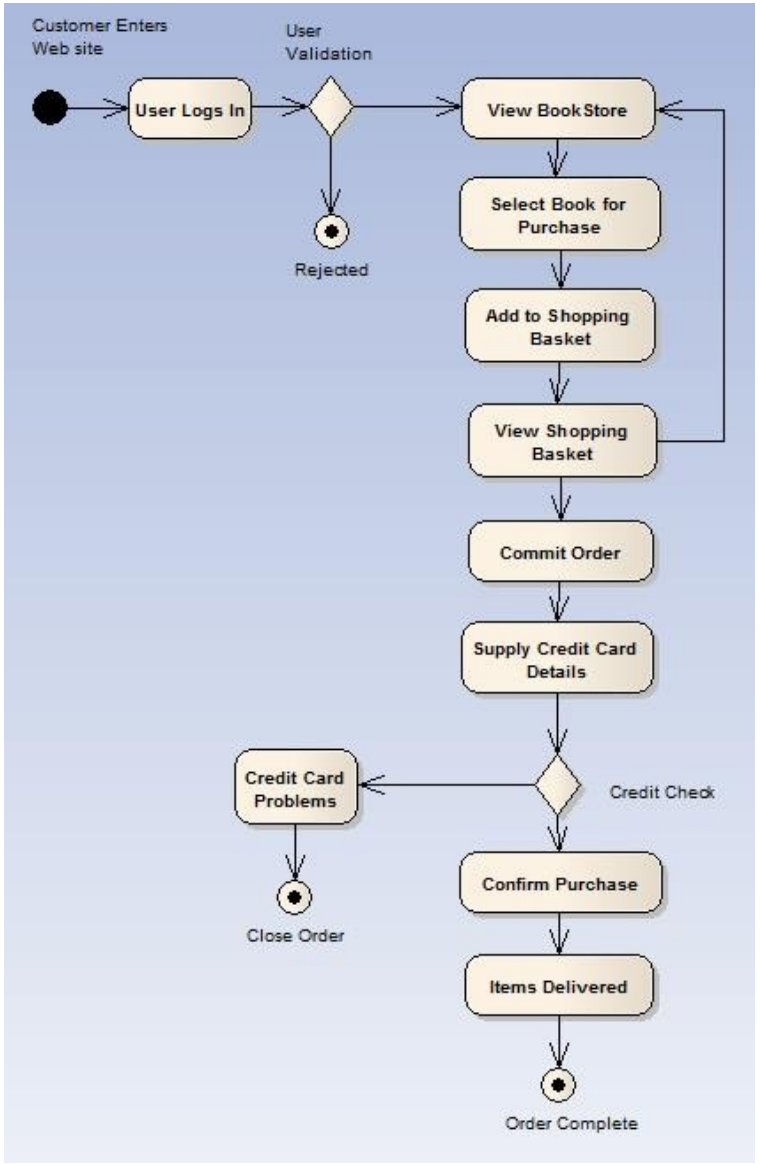


Fonte: Enterprise Architect


UNIFIED
MODELING
LANGUAGE
TM

Diagrama de Atividades

- Usado para descrever uma seqüência de atividades com suporte para o comportamento condicional e paralelo de processos de workflow (processos de fluxo de negócios)
- Permitw uma radiografia de uma lógica de código, mostrando essa lógica como um fluxograma, mas com recursos mais poderosos.
- Concentra-se no fluxo de controle da atividade.



Elementos do Diagrama de Atividades

- **Formado por:**
  - **Estados Inicial e Final**
  - **Transições**
  - **Barras de Sincronização**
  - **Estado de Ação**
  - **Ponto de Decisão**

Diagrama de Atividades

- **Estados Inicial e Final**
  - **São estados abstratos que determinam o início e o fim de um Diagrama de Atividades.**

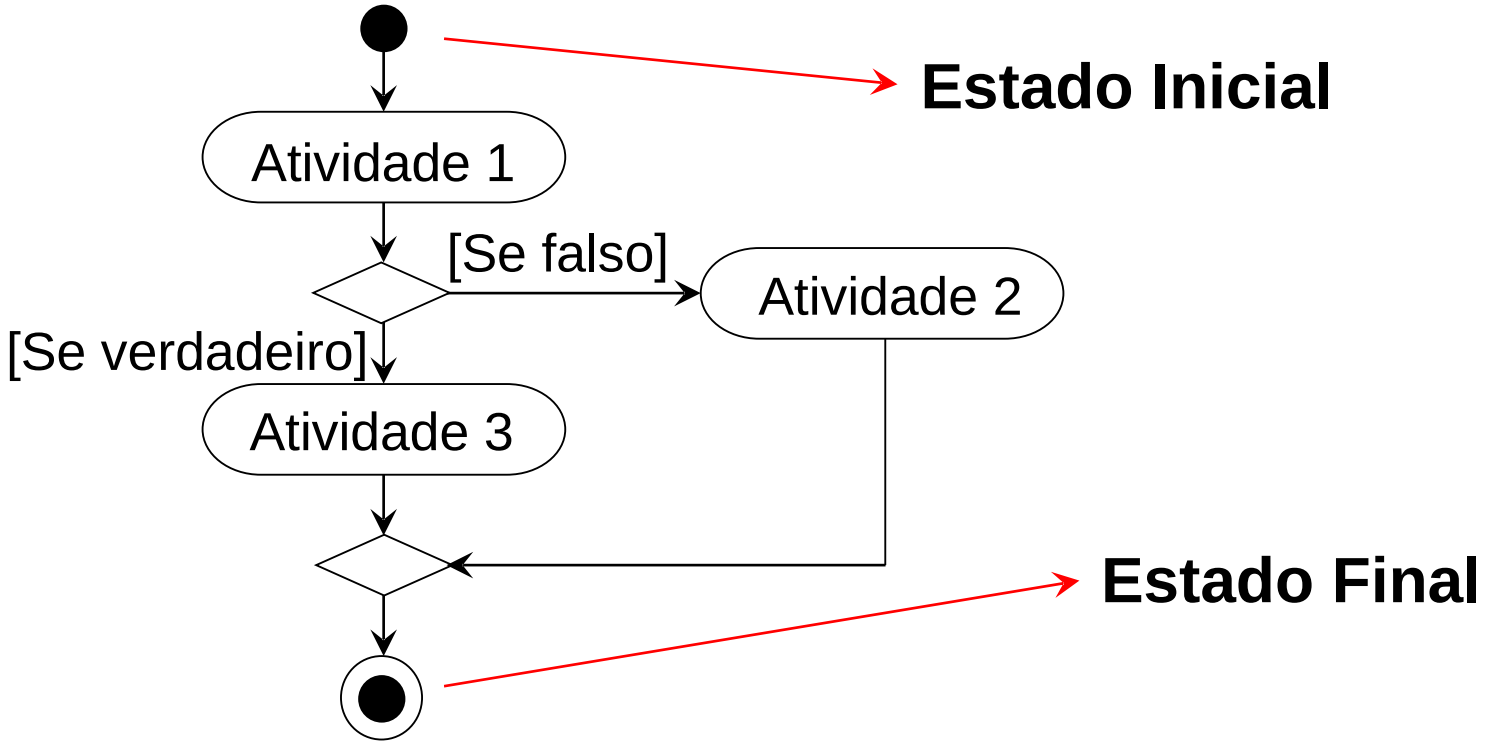


Diagrama de Atividades

- **Transição**
  - **Interliga os estados de um diagrama.**

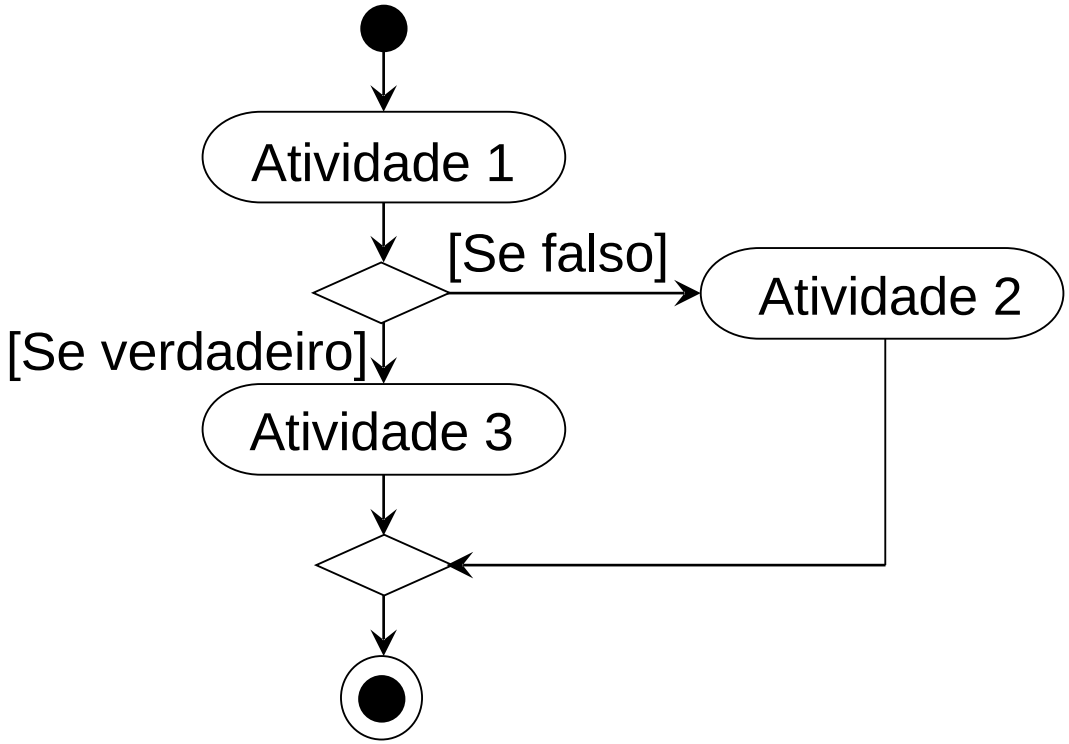
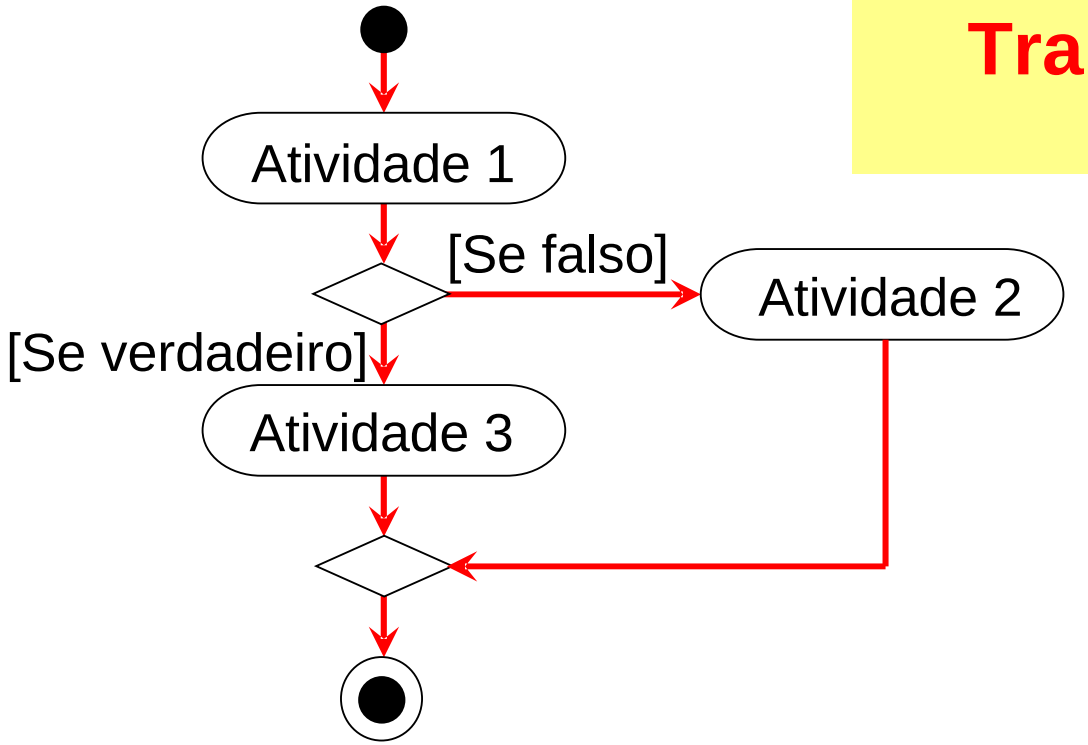


Diagrama de Atividades

- **Transição**
  - **Interliga os estados de um diagrama.**



Transições

Diagrama de Atividades

- **Estado de Ação**
  - **Representa a realização de uma ação dentro de um fluxo de controle.**

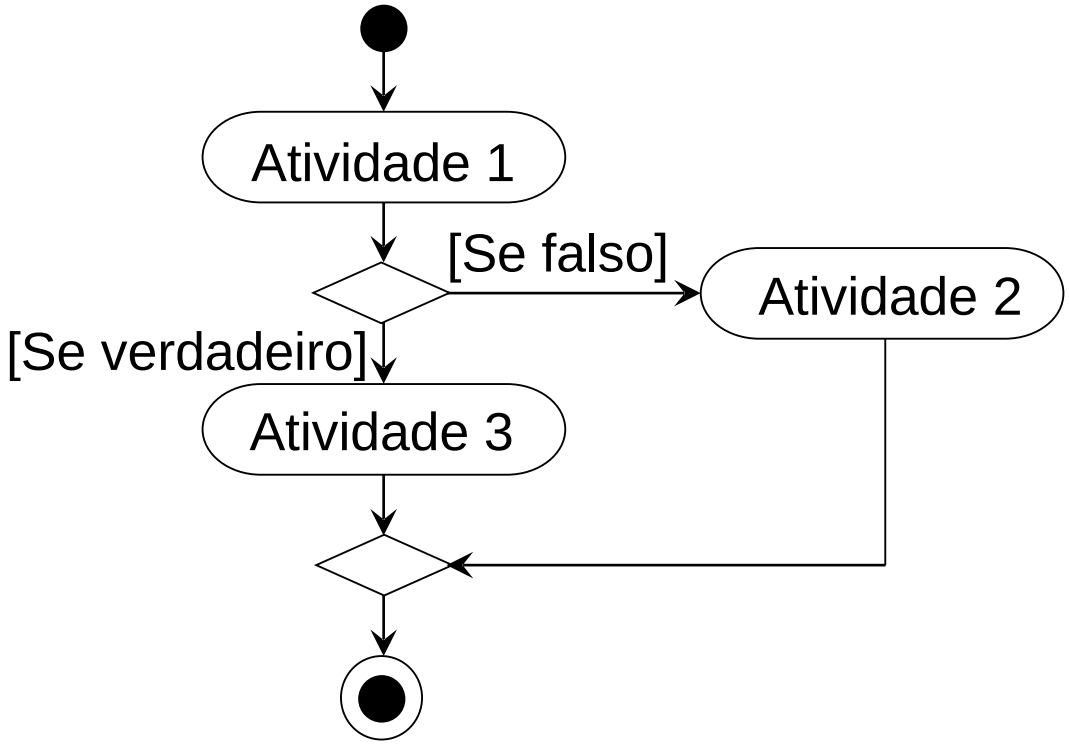


Diagrama de Atividades

- **Estado de Ação**
  - **Representa a realização de uma ação dentro de um fluxo de controle.**

Estados de Ação

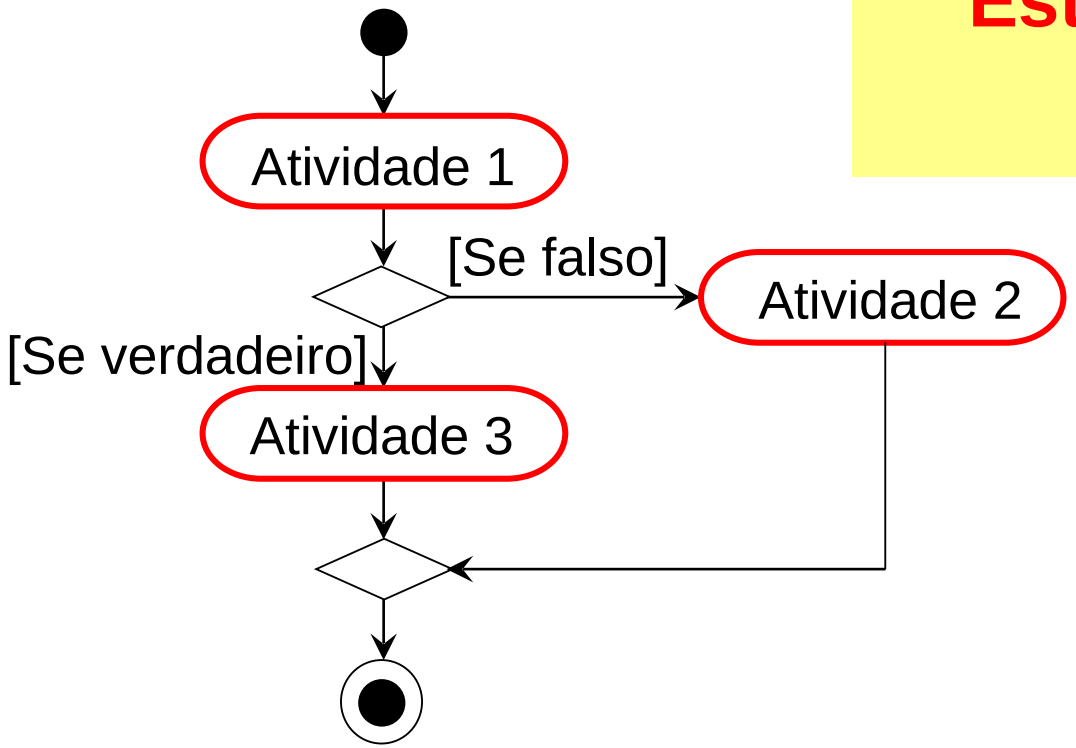


Diagrama de Atividades

- **Ponto de Decisão**
  - **Representa um ponto do fluxo de controle onde deve ser realizado um teste, uma tomada de decisão.**

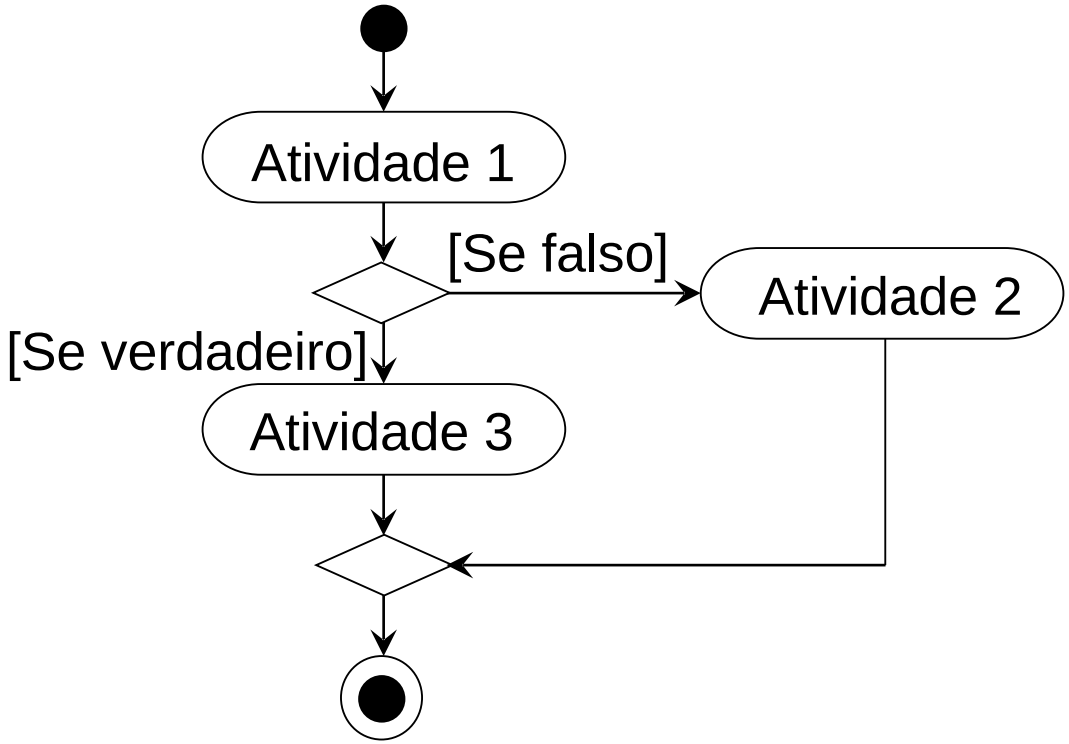
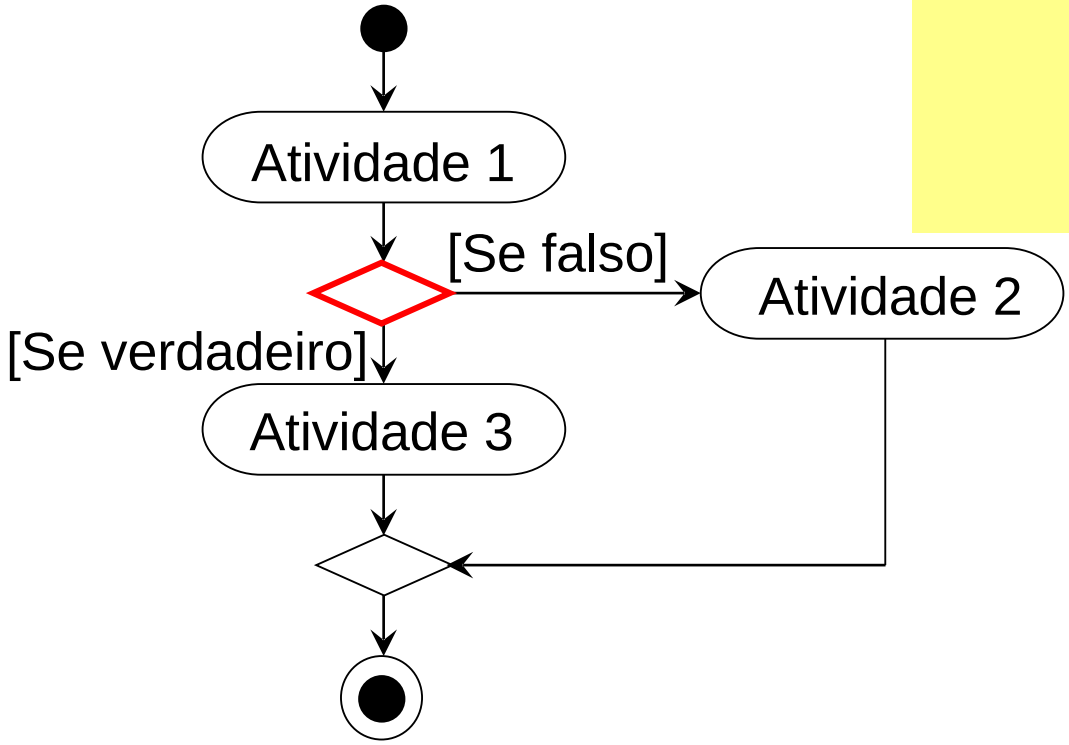


Diagrama de Atividades

- **Ponto de Decisão**
  - **Representa um ponto do fluxo de controle onde deve ser realizado um teste, uma tomada de decisão.**



Ponto de Decisão

Diagrama de Atividades

- **Condição de guarda**
  - **Trata-se de uma expressão lógica (pode ser verdadeira ou r determinar a transição para uma ou outra atividade.**

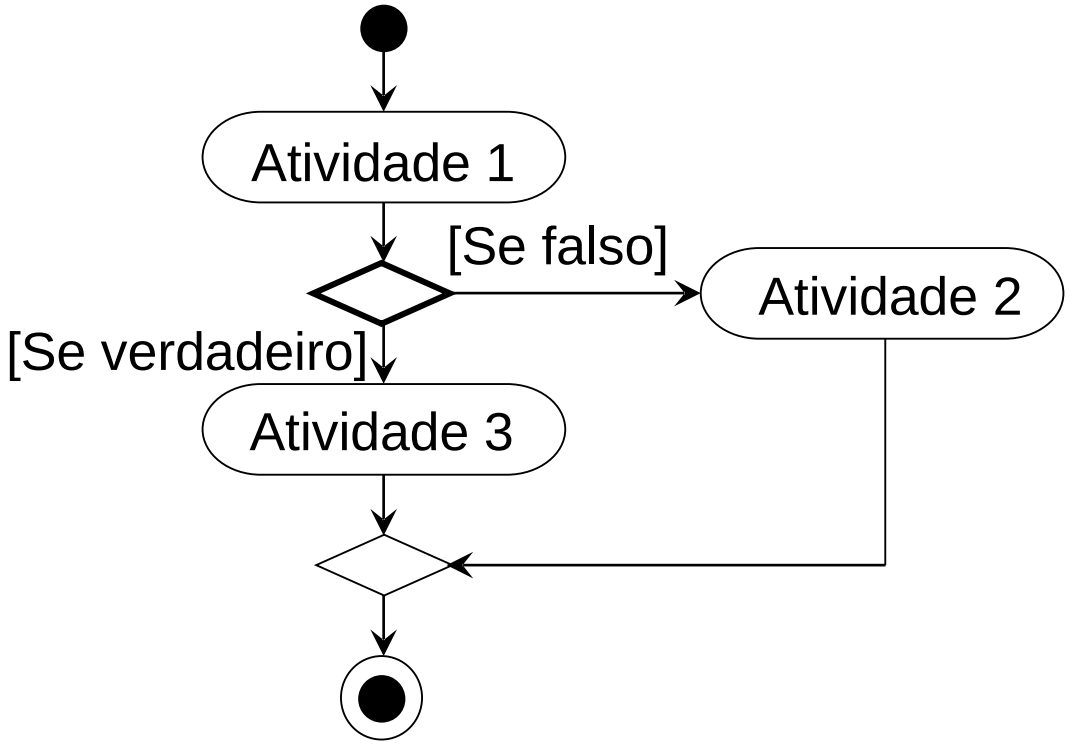
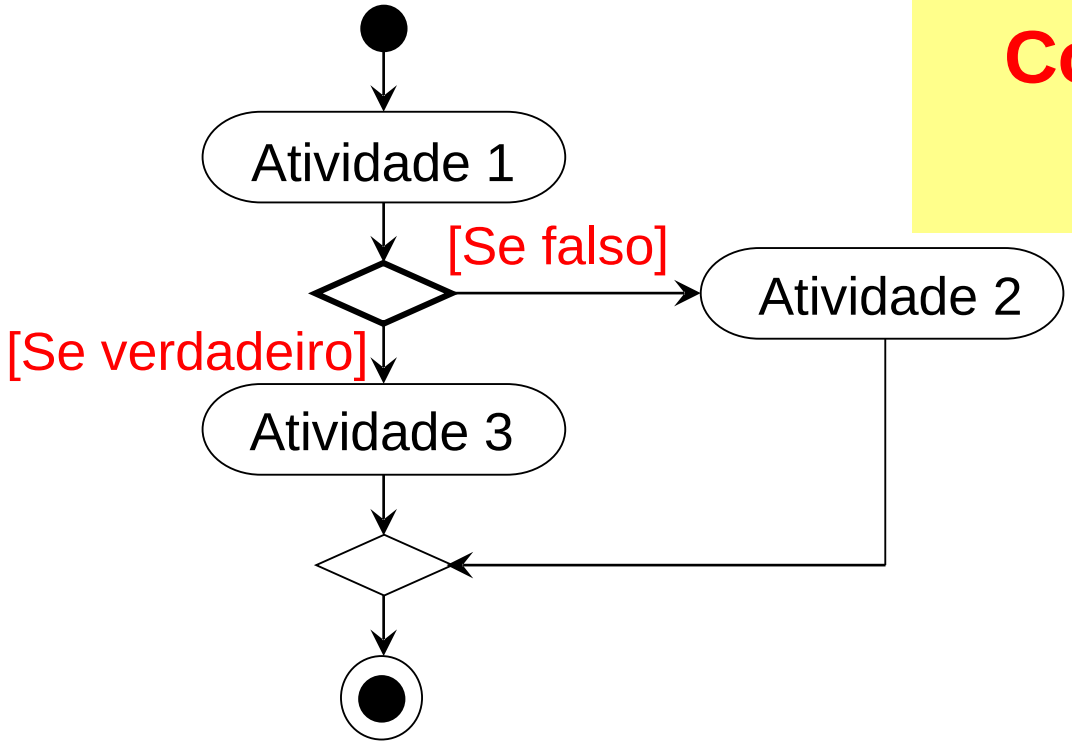


Diagrama de Atividades

- **Condição de guarda**
  - **Trata-se de uma expressão lógica (pode ser verdadeira ou f determinar a transição para uma ou outra atividade.**



Condição de Guarda

Diagrama de Atividades

- **Barra de Sincronização**
  - **Identifica o ponto a partir do qual o processo passou a ser executado em paralelo, ou processos em paralelo se unem.**

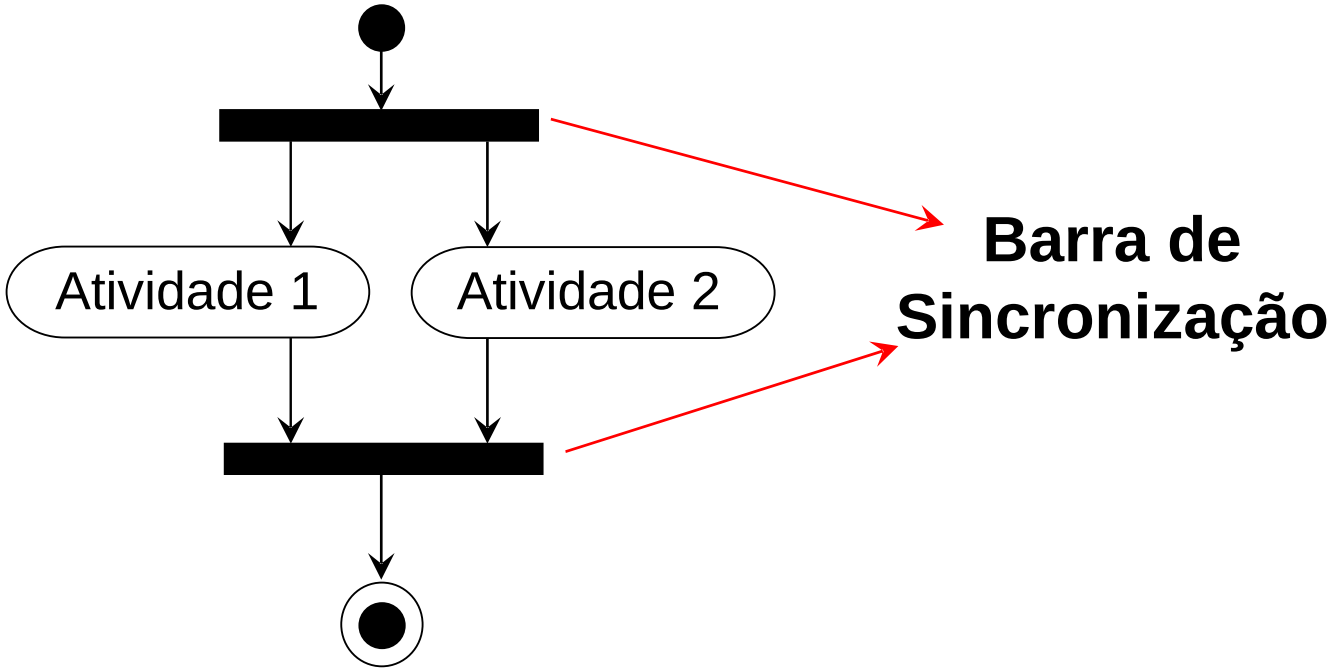


Diagrama de Atividades

- **Bifurcação (Fork)**
  - **Identifica o ponto a partir do qual o processo passou a ser executado em paralelo.**

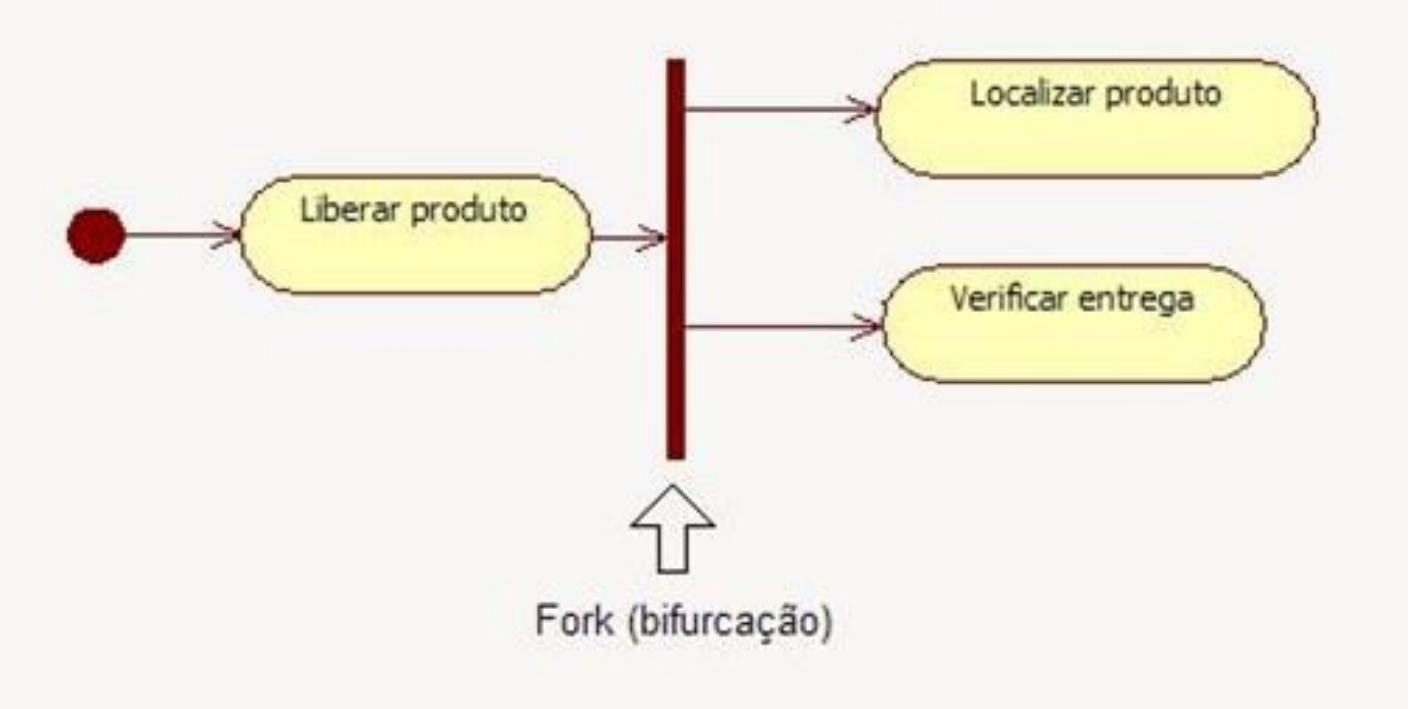


Diagrama de Atividades

- **Junção (Join)**
  - **Identifica o ponto a partir do qual o processo paralelo foi sincronizado.**

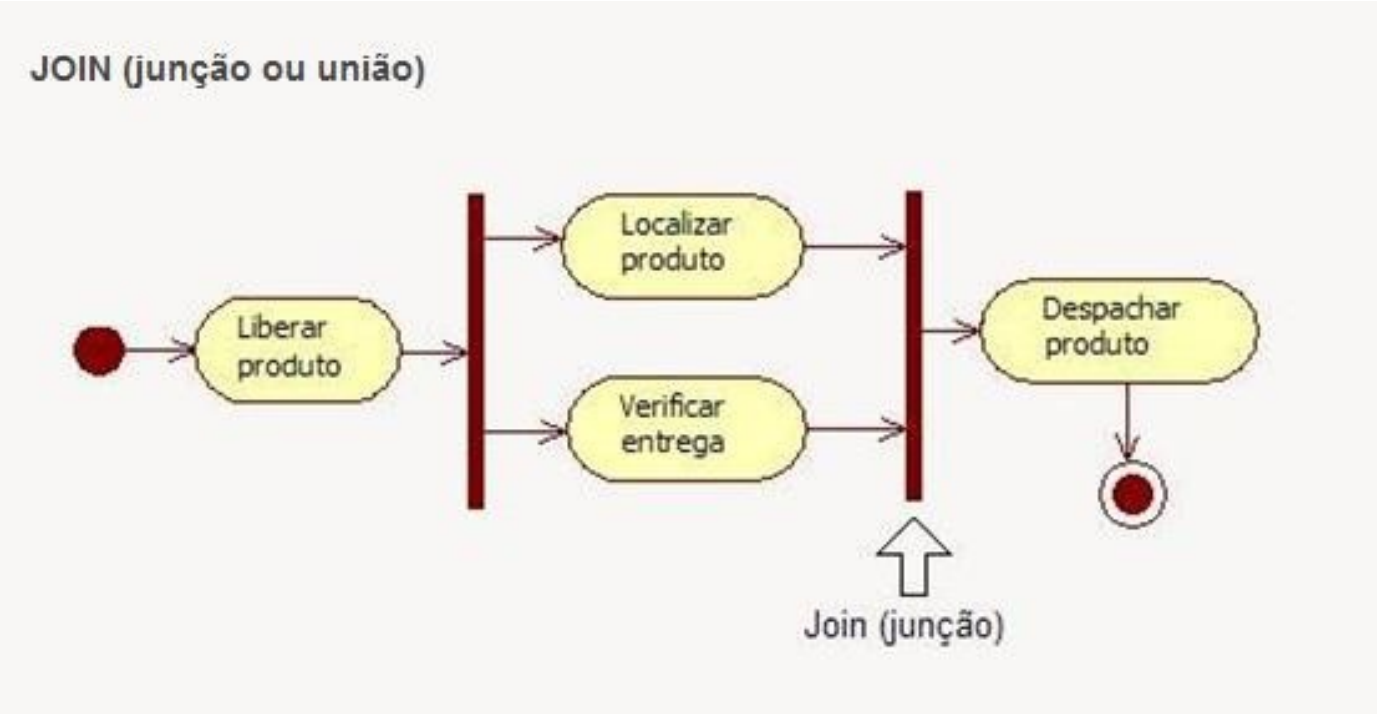
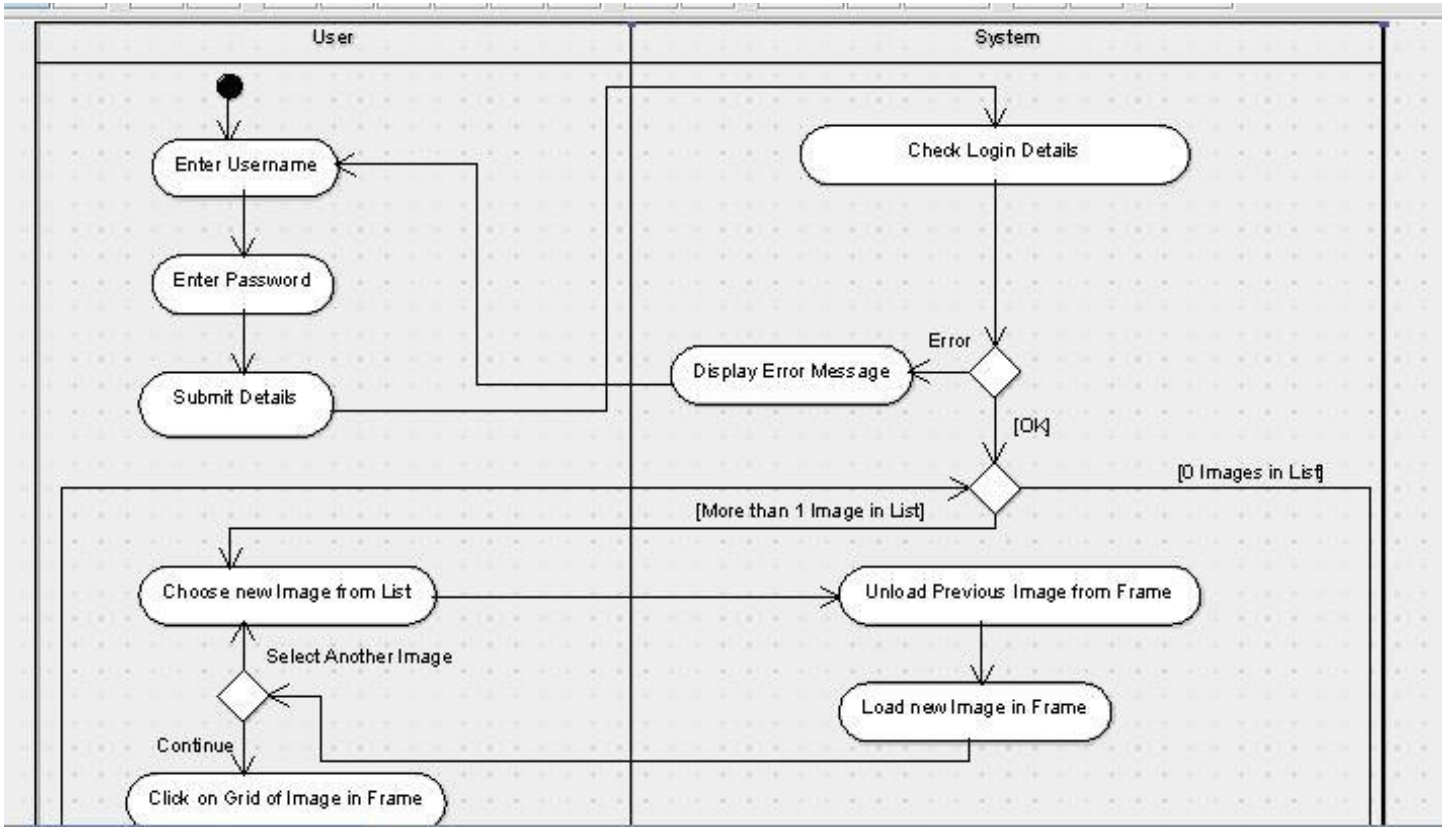
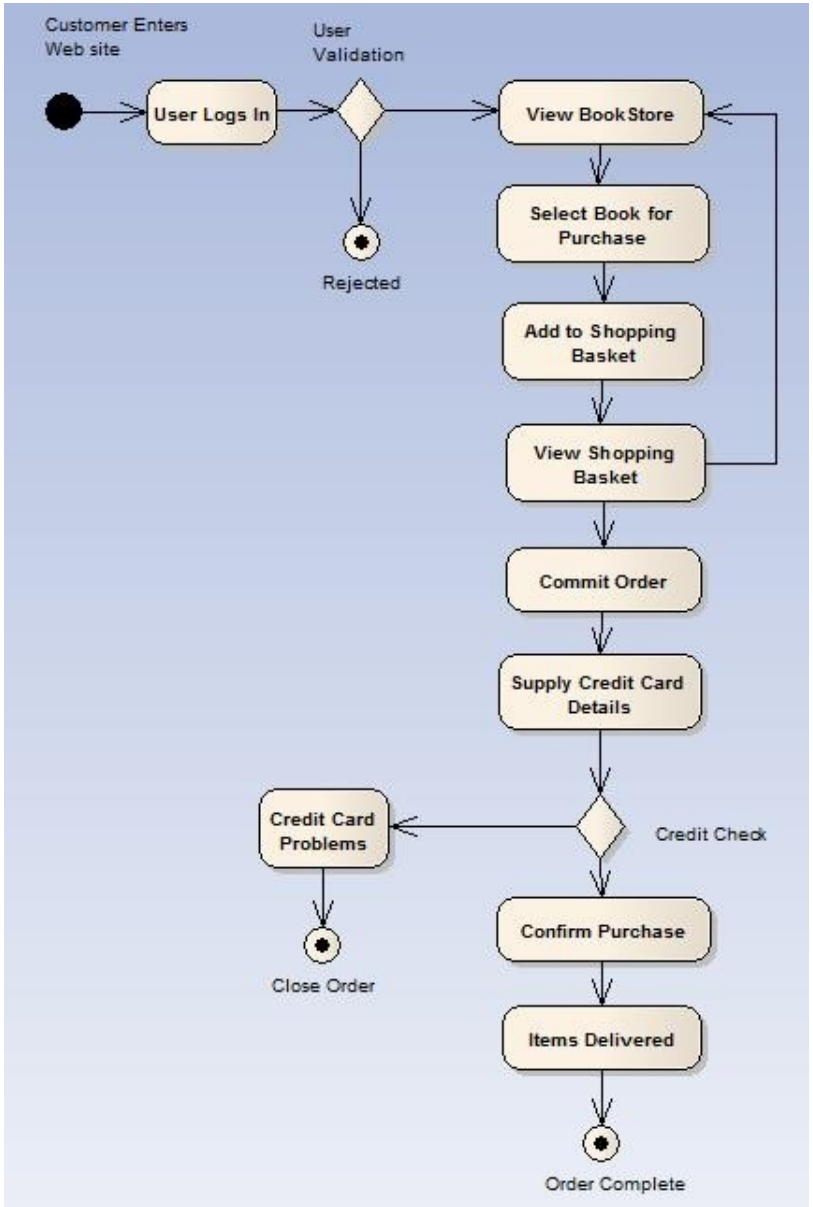


Diagrama de Atividades

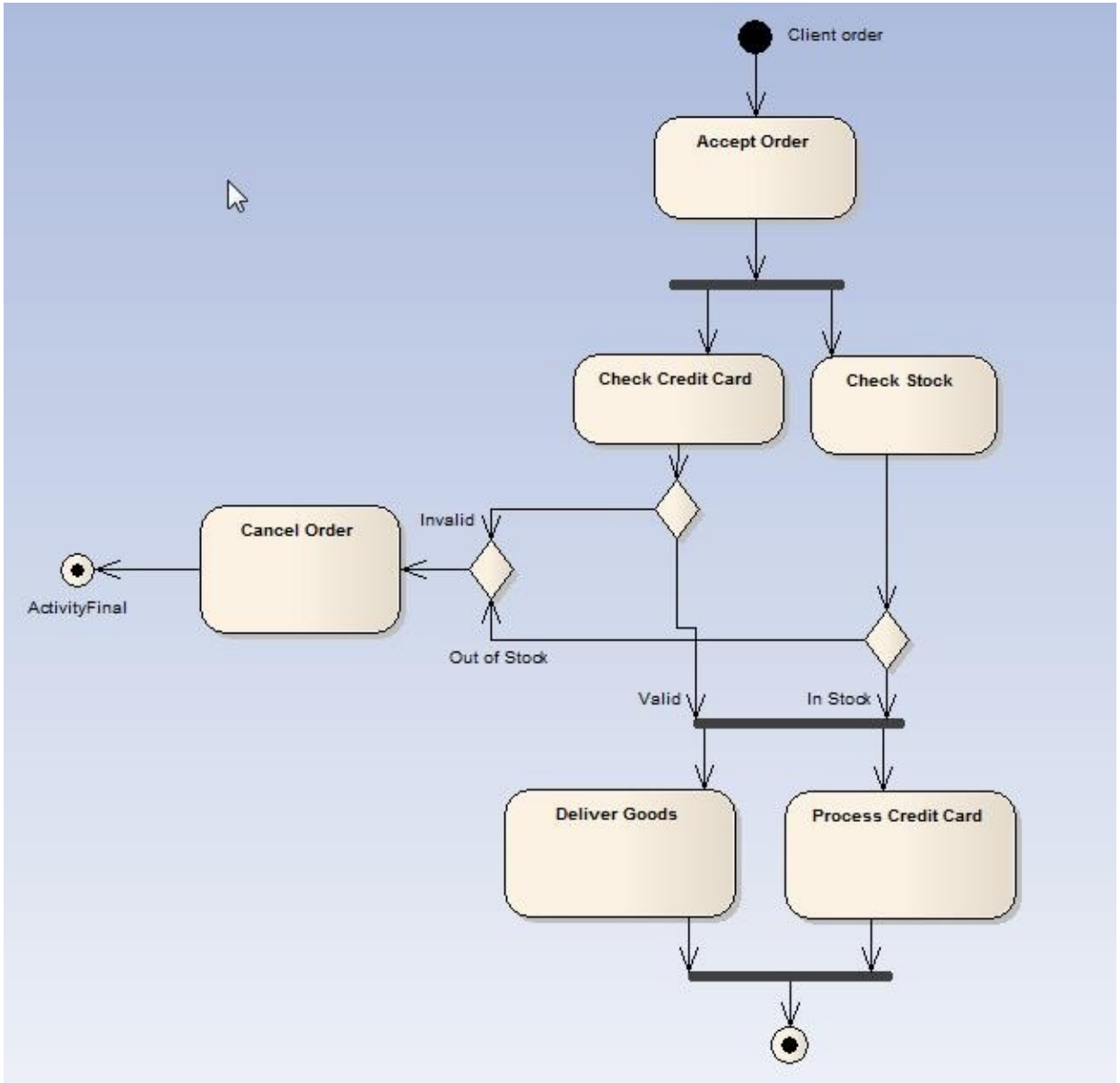
- **Raia (*swimlane*)**
  - **visualmente distingue o compartilhamento de trabalho e as responsabilidades**



Exemplos - 1


Customer Enters Web site
User Validation
User Logs In
View Book Store
Select Book for Purchase
Rejected
Add to Shopping Basket
View Shopping Basket
Commit Order
Supply Credit Card Details
Credit Card Problems
Credit Check
Close Order
Confirm Purchase
Items Delivered
Order Complete

Exemplos - 2


Client order
Accept Order
Check Credit Card
Check Stock
Cancel Order
Invalid
ActivityFinal
Out of Stock
Valid
In Stock
Deliver Goods
Process Credit Card

Referencias

- http://blog.tcavalcante.com/2009/a-importancia-da-modelagem-de-sistema/
- http://www.omg.org/index.htm
- http://www.sparxsystems.com/uml-tutorial.html
- Enterprise Architect – EAExample
- http://www.fernandoamaral.com.br/Default.aspx?Artigo=40