Computação Orientada a Objetos II

Prof. Dr. Rodrigo Duarte Seabra

Universidade Federal de Itajubá
Bacharelado em Ciência da Computação/Sistemas de Informação

Contato

- ✓ **E-mail**: rodrigo@unifei.edu.br

- ✓ **Material:**
  - *Slides das aulas*
  - *Livros texto sobre UML*

✓ Conteúdo Programático

01/08: *Apresentação da disciplina e Aula 01 – Introdução à UML*
08/08: *Aula 02 – Revisão sobre Orientação a Objetos*
22/08: *Aula 03 – Diagrama de Casos de Uso*
29/08: *Aula 04 – Diagrama de Classes*
05/09: *Aula 05 – Diagrama de Objetos e Pacotes / Exercícios Extras*
12/09: *Aula 06 – Diagrama de Sequência e Aula 07 – Diagrama de Comunicação*
19/09: *Aula 08 – Diagrama de Máquina de Estados*
26/09: *Aula 09 – Diagrama de Atividades*
03/10: *Avaliação 1*
10/10: *Vista da Avaliação 1 e Projeto da disciplina*
17/10: *Execução do projeto*
24/10: *Execução do projeto*
31/10: *Execução do projeto*
07/11: *Execução do projeto*
14/11: *Entrega do projeto e Avaliação 2*
21/11: *Aplicação da SUB*

✓ Critérios de Avaliação

- **N1**: composta por 1 avaliação e 7 trabalhos: (Aval*0,7)+(Exer*0,3)
- **N2**: composta por 1 avaliação e 1 trabalho: (Aval*0,6)+(Exer*0,4)

Nota Final = (N1 + N2)/2

Regras

✓ **Presença**: a presença em sala de aula será verificada por meio de chamada.

✓ **Avaliações**: individuais e sem consulta.

✓ **Avaliação de Reposição**: aplicada conforme regras da Norma de Graduação da UNIFEI.

✓ **Trabalhos Práticos/Exercícios**: realizados em equipes de até 5 alunos.

✓ **Regras de Conduta**: conduzir as atividades e os trabalhos acadêmicos com honestidade e integridade. Falhas de conduta como cópia de trabalhos e exercícios de colegas, cola etc., serão punidas com **dedução total da nota do trabalho e/ou avaliação**.

✓ **Cada encontro semanal gera 4 presenças ou 4 faltas!!!**

Introdução à UML
Aula 01

Prof. Dr. Rodrigo Duarte Seabra

Universidade Federal de Itajubá
Bacharelado em Ciência da Computação/Sistemas de Informação

- ✓ **UML – Unified Modeling Language (Linguagem de Modelagem Unificada)**
  - Linguagem visual utilizada para modelar softwares baseados no paradigma da orientação a objetos
  - Aplicada a todos os domínios de aplicação
  - Linguagem padrão de modelagem adotada internacionalmente pela indústria de engenharia de software
  - Auxilia os engenheiros de software a definirem as características do sistema (requisitos, comportamento, estrutura lógica etc.) antes de seu desenvolvimento

Breve Histórico da UML

- ✓ **Surgiu da união de três métodos de modelagem (meados da década de 90):**
  - Método de Booch
  - Método OMT (Object Modeling Technique) de Jacobson
  - Método OOSE (Object-Oriented Software Engineering) de Rumbaugh
- ✓ **Amplo apoio da Rational Software**
- ✓ **"Os Três Amigos":** surgimento da primeira versão da UML (1996)
- ✓ **Várias empresas atuantes na área de modelagem e desenvolvimento de software passaram a contribuir para o projeto**
- ✓ **1997: UML é adotada pela OMG (Object Management Group)**
- ✓ **2005: versão 2.0 da linguagem (www.uml.org) → versão atual: 2.5**

Por que Modelar Software?

✓ **Qual a real necessidade de se modelar um software?**

✓ O projeto envolve **fatores extremamente complexos**, como **levantamento e análise de requisitos, prototipação, tamanho do projeto, complexidade, prazos, custos, documentação, manutenção e reusabilidade**, entre outros.

✓ Ex: construção de uma pequena casa e de um prédio de vários andares.

✓ **Todo e qualquer** sistema **deve ser modelado antes de iniciar sua implementação**.

✓ Sistemas de informação são **dinâmicos**, pois normalmente estão em constante mudança

- Os clientes desejam constantemente modificações ou melhoras no sistema
- O mercado está sempre mudando, o que força a adoção de novas estratégias por parte das empresas e, consequentemente, de seus sistemas
- O governo seguidamente promulga novas leis e cria novos impostos e alíquotas, o que acarreta a manutenção no software

Modelo de Software – Uma Definição

✓ **Captura uma visão de um sistema físico**, é uma abstração do sistema com um certo propósito, como descrever aspectos estruturais ou comportamentais do software

✓ Descreve aspectos do sistema físico que são relevantes ao propósito do modelo, no nível apropriado de detalhe

Levantamento e Análise de Requisitos

- ✓ **Levantamento de requisitos**: fase inicial do desenvolvimento
- ✓ Domínio do problema e determinar "**o que**" o software deve fazer e se é realmente possível desenvolver o software solicitado
- ✓ Engenheiro de software busca compreender as **necessidades do usuário** e o que ele deseja que o sistema a ser desenvolvido realize
- ✓ Processo usualmente utilizado: **entrevista**
- ✓ Problema enfrentado: **comunicação**
- ✓ Identificar dois tipos de requisitos: **funcionais** e **não-funcionais**

✓ **Requisitos funcionais:** *funcionalidades* do software (aquilo que o cliente deseja)

✓ **Requisitos não-funcionais:** *restrições*, *condições*, *consistências*, *validações* que devem ser levadas a efeito sobre os requisitos funcionais

✓ Ex: sistema bancário

- requisito funcional: **opção de abrir novas contas corrente**
- requisito não-funcional: **somente pessoas maiores de idade podem abrir contas corrente**

✓ Outros exemplos de requisitos não-funcionais: usabilidade, desempenho, confiabilidade, segurança ou interface.

- ✓ Alguns requisitos não-funcionais identificam **regras de negócio**

- ✓ *Políticas, normas e condições estabelecidas pela empresa que devem ser seguidas na execução de uma funcionalidade*

- ✓ Ex1: após abrir uma conta é necessário depositar um valor mínimo inicial (sistema bancário)

- ✓ Ex2: realizar uma nova locação para um sócio após ele ter devolvido as cópias locadas anteriormente (sistema de videolocadora)

- ✓ **Análise de requisitos**: necessidades apresentadas pelo cliente são analisadas.

Levantamento e Análise de Requisitos

✓ Durante a análise de requisitos, uma **linguagem de modelagem auxilia a levantar questões** que não foram concebidas durante as entrevistas iniciais

✓ Sanar os problemas o quanto antes, para que o projeto do software **não tenha que sofrer modificações** quando seu desenvolvimento já estiver em andamento

✓ **Problema 1**: os usuários não tem realmente certeza do que querem e não conseguem enxergar as reais potencialidades de um sistema de informação

✓ **Problema 2**: engenheiros encontram resistências a qualquer mudança na forma como a empresa manipula suas informações

Prototipação

✓ **Desenvolver rapidamente um "rascunho"** do que seria o sistema quando ele estivesse finalizado

✓ **Apresentar exemplos fictícios** de quais seriam os resultados apresentados pelo software, principalmente em forma de relatórios

✓ Identificar problemas quanto às necessidades do cliente devido a **falhas de comunicação** durante as entrevistas iniciais

✓ Ferramentas RAD (Rapid Application Development)

- NetBeans, Delphi, Visual Basic ou C++ Builder, entre outras
- Proporcionam facilidade de produzir e modificar as interfaces

Prototipação

✓ Por meio dos protótipos, a maioria das dúvidas e erros de especificação pode ser sanada pois eles **facilitam a compreensão do cliente**

✓ **Ressalva**: o protótipo pode induzir o cliente a acreditar que o software encontra-se em um estágio bastante avançado de desenvolvimento

✓ Deixar claro ao usuário que o software apresentado é apenas um rascunho e que não foi realmente iniciado!

Prazos e Custos

- ✓ **Como determinar o prazo real de entrega de um software?**
- ✓ **Quantos profissionais deverão trabalhar no projeto?**
- ✓ **Qual será o custo total de desenvolvimento?**
- ✓ **Qual deverá ser o valor estipulado para produzir o sistema?**
- ✓ **Como determinar a real complexidade de desenvolvimento do software?**

✓ Por melhor modelado que um sistema tenha sido, ainda assim fica difícil determinar com exatidão os prazos finais de entrega do software

✓ Uma boa modelagem auxilia a estimar a complexidade de desenvolvimento de um sistema, além do prazo final de entrega

Prazos e Custos

✓ Experiência documentada de desenvolvimento de outros softwares

✓ Acrescentar alguns meses à data de entrega, o que serve como margem de segurança para possíveis erros de estimativa

✓ Estimativas de prazos erradas:

- pagamentos de salários aos profissionais envolvidos
- desgaste dos equipamentos utilizados
- manter profissionais ocupados
- insatisfação dos clientes
- propaganda negativa

Projeto

✓ Domínio da solução procurando estabelecer "**como**" o sistema fará o que foi determinado na fase de análise

✓ Produzida a arquitetura do sistema (maior parte da modelagem do software)

✓ Como as funcionalidades deverão realizar o que foi solicitado

✓ Seleção de artefatos:

- Linguagem de programação, SGBD, interface final do sistema, distribuição física do software na empresa, especificação de hardware.

Manutenção

- ✓ Representa 40 a 60% do custo total do projeto
- ✓ Se a modelagem estiver correta o sistema não apresentará erros, e não sofrerá manutenções
- ✓ Algumas modificações são inevitáveis, decorrentes da dinamicidade e exigências de mercado
- ✓ Facilitar a compreensão do sistema para quem tiver que mantê-lo
  - “código alienígena” ou “software legado”

Manutenção

✓ Manutenções devem ser modeladas e documentadas para evitar desatualizar a documentação do sistema

✓ “Praga das saúvas e formigas lava-pés”

Documentação Histórica

✓ Documentação histórica dos projetos anteriores já concluídos pela empresa

✓ **A empresa está evoluindo?**

✓ **O processo de desenvolvimento tornou-se mais rápido?**

✓ **As metodologias hoje adotadas são superiores às práticas aplicadas anteriormente?**

✓ **A qualidade do software produzido está melhorando?**

Documentação Histórica

- ✓ Registro detalhado para determinar:

- ✓ **A média de manutenções que um sistema sofre normalmente dentro de um determinado período de tempo**
- ✓ **Qual a média de custo de modelagem?**
- ✓ **Qual a média de custo de desenvolvimento?**
- ✓ **Qual a média de tempo despendido até a finalização do projeto?**
- ✓ **Quantos profissionais são necessários envolver normalmente em um projeto?**

Documentação Histórica

- ✓ Reusabilidade

- ✓ **Onde as rotinas se encontram?**
- ✓ **Para que foram utilizadas?**
- ✓ **Em que projetos estão documentadas?**
- ✓ **Elas são adequadas ao software atualmente em desenvolvimento?**
- ✓ **Qual o nível necessário de adaptação destas rotinas para que possam ser utilizadas na construção do sistema atual?**

UML: Por que tantos diagramas?

✓ Oferecer múltiplas visões do sistema a ser modelado, analisando-o e modelando-o sob diversos aspectos

✓ Cada diagrama da UML analisa o sistema, ou parte dele, sob determinada ótica

✓ O uso de vários diagramas permite que falhas sejam descobertas, diminuindo a possibilidade da ocorrência de erros futuros

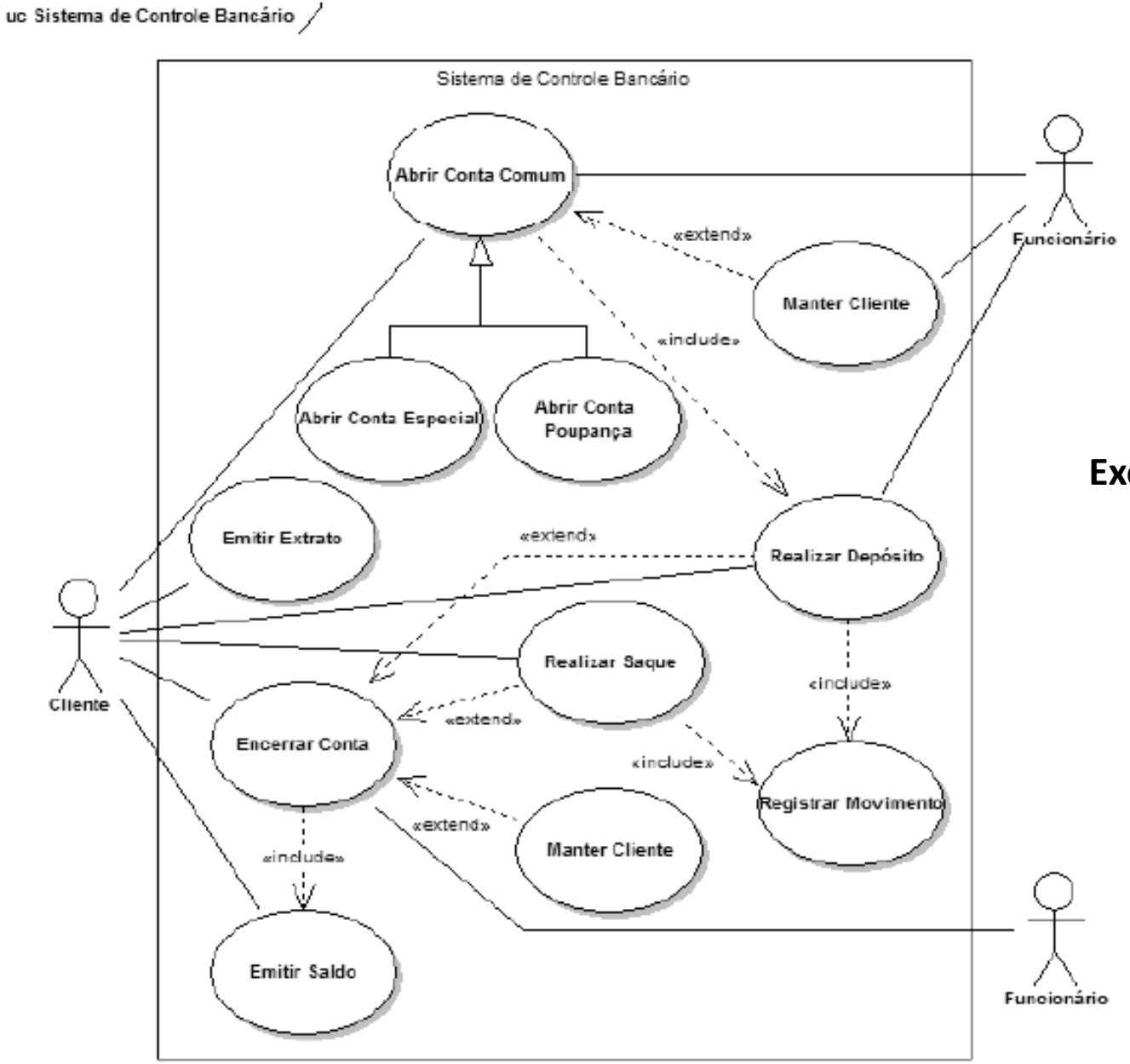
Rápido Resumo dos Diagramas da UML

- ✓ **Diagrama de Casos de Uso**
- ✓ **Diagrama de Classes**
- ✓ **Diagrama de Objetos**
- ✓ **Diagrama de Pacotes**
- ✓ **Diagrama de Sequência**
- ✓ **Diagrama de Comunicação**
- ✓ **Diagrama de Máquina de Estados**
- ✓ **Diagrama de Atividade**
- ✓ **Diagrama de Visão Geral de Interação**
- ✓ **Diagrama de Componentes**
- ✓ **Diagrama de Implantação**
- ✓ **Diagrama de Estrutura Composta**
- ✓ **Diagrama de Tempo ou de Temporização**

Diagrama de Casos de Uso

- ✓ Mais geral e informal da UML
- ✓ Utilizado normalmente nas fases de levantamento e análise de requisitos do sistema
- ✓ Usuários podem ter uma ideia geral de como o sistema irá se comportar
- ✓ Identificar os atores (usuários, outros sistemas ou até mesmo algum hardware especial) que utilizarão de alguma forma o software

Diagrama de Casos de Uso

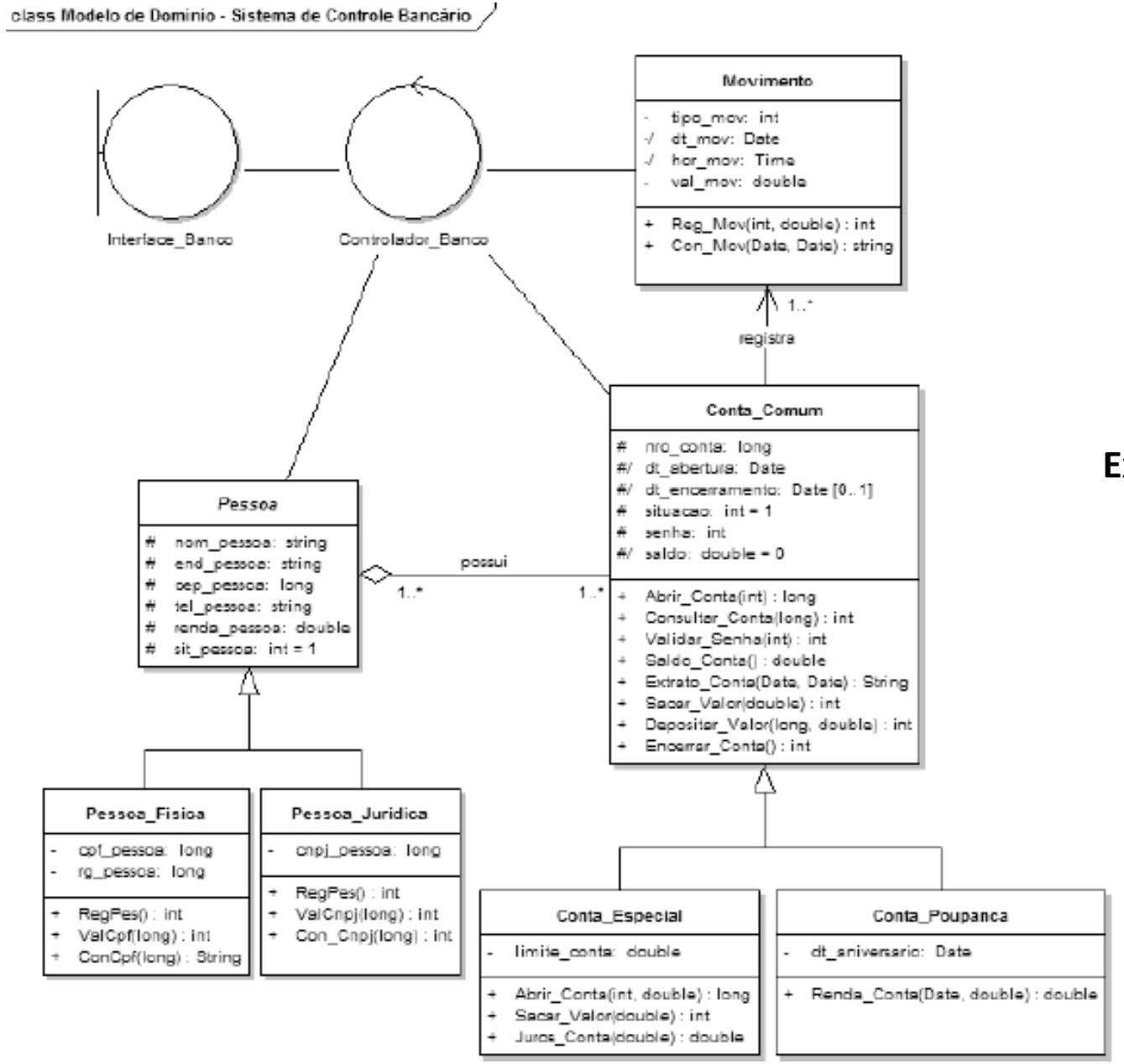


Exemplo de Diagrama de Casos de Uso

Diagrama de Classes

- ✓ Um dos mais importantes e provavelmente o mais utilizado
- ✓ Define a estrutura das classes utilizadas pelo sistema, determinando os atributos e métodos que cada classe possui
- ✓ Estabelece como as classes se relacionam e trocam informações entre si

Diagrama de Classes



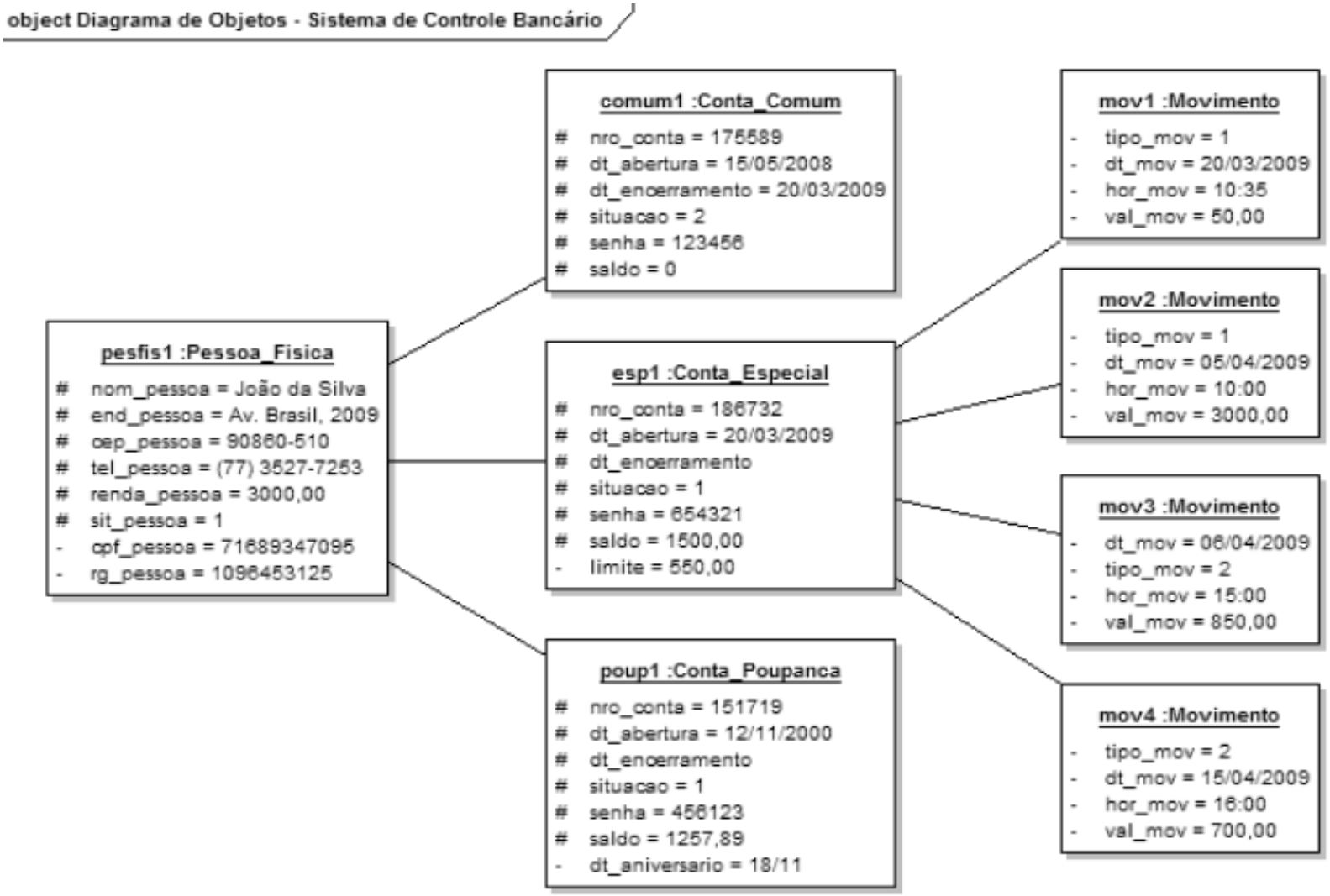
Exemplo de Diagrama de Classes

Diagrama de Objetos

- ✓ Amplamente associado ao diagrama de classes (complemento)

- ✓ Fornece uma visão dos valores armazenados pelos objetos de um diagrama de classes em um determinado momento da execução de um processo do software

Diagrama de Objetos

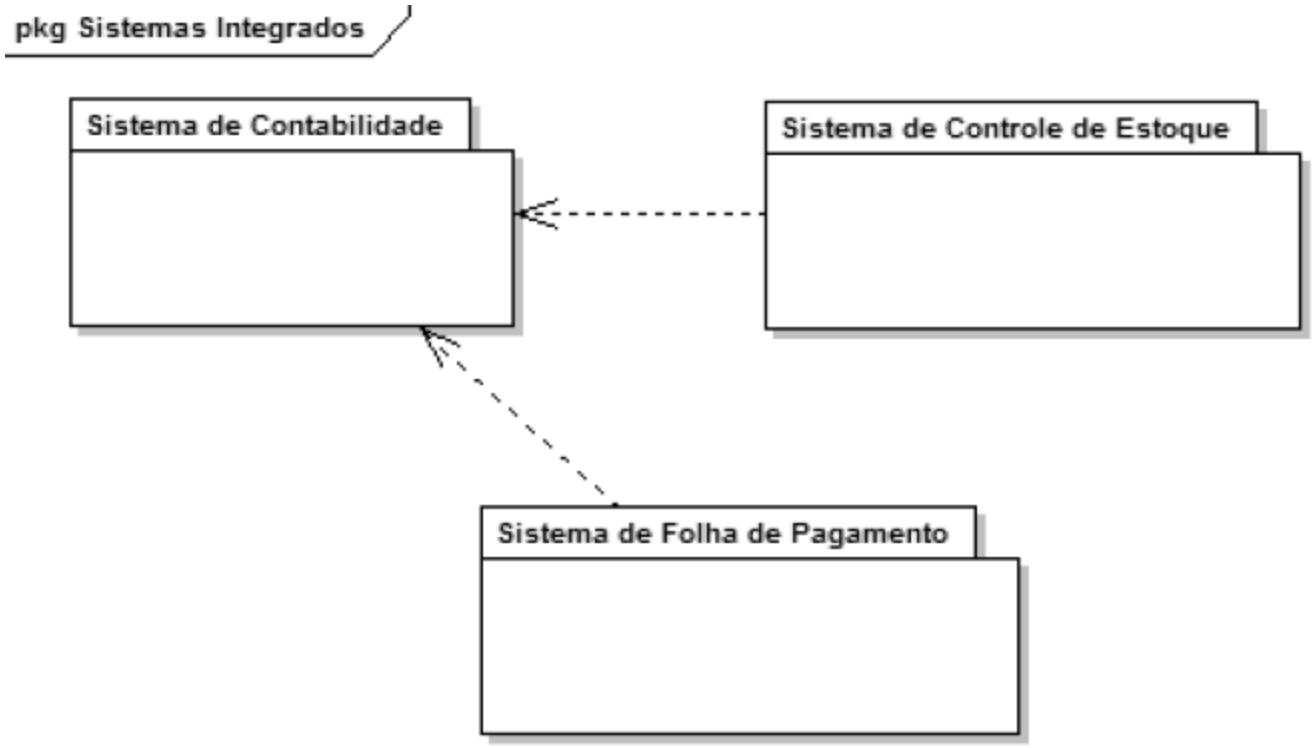


Exemplo de Diagrama de Objetos

Diagrama de Pacotes

- ✓ Representa os subsistemas ou submódulos englobados por um sistema
- ✓ Determina as partes que compõem o sistema
- ✓ Usado de maneira independente ou associado com outros diagramas
- ✓ Pode ser usado para definir as camadas de um software ou de um processo de desenvolvimento

Diagrama de Pacotes



Exemplo de Diagrama de Pacotes

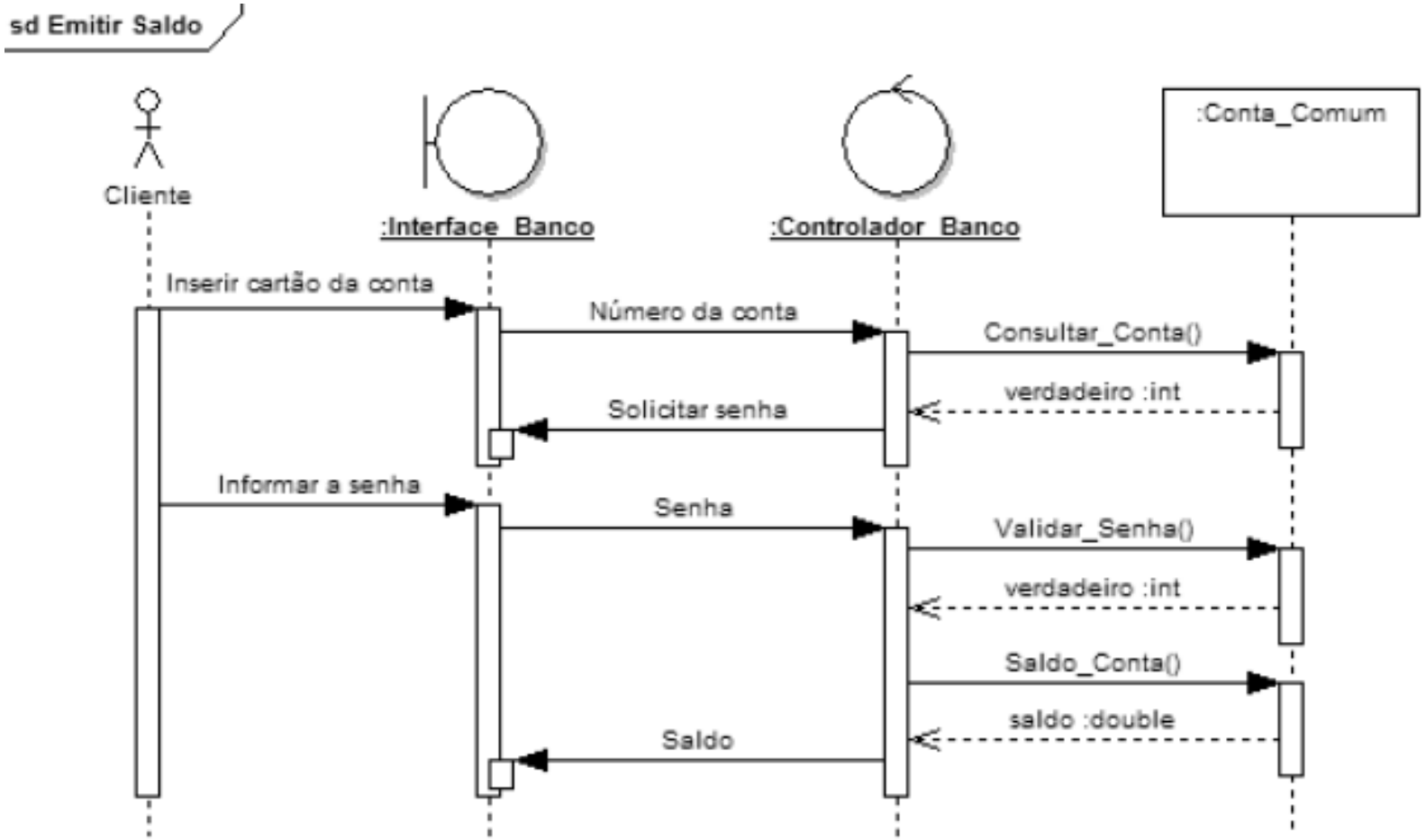
Diagrama de Sequência

✓ Diagrama comportamental que modela as mensagens trocadas entre os objetos envolvidos em um determinado processo

✓ Em geral, baseia-se em um caso de uso definido pelo diagrama de mesmo nome

✓ Mapeia como o processo deve se desenrolar e ser concluído por meio da chamada de métodos disparados por mensagens enviadas entre os objetos

Diagrama de Sequência



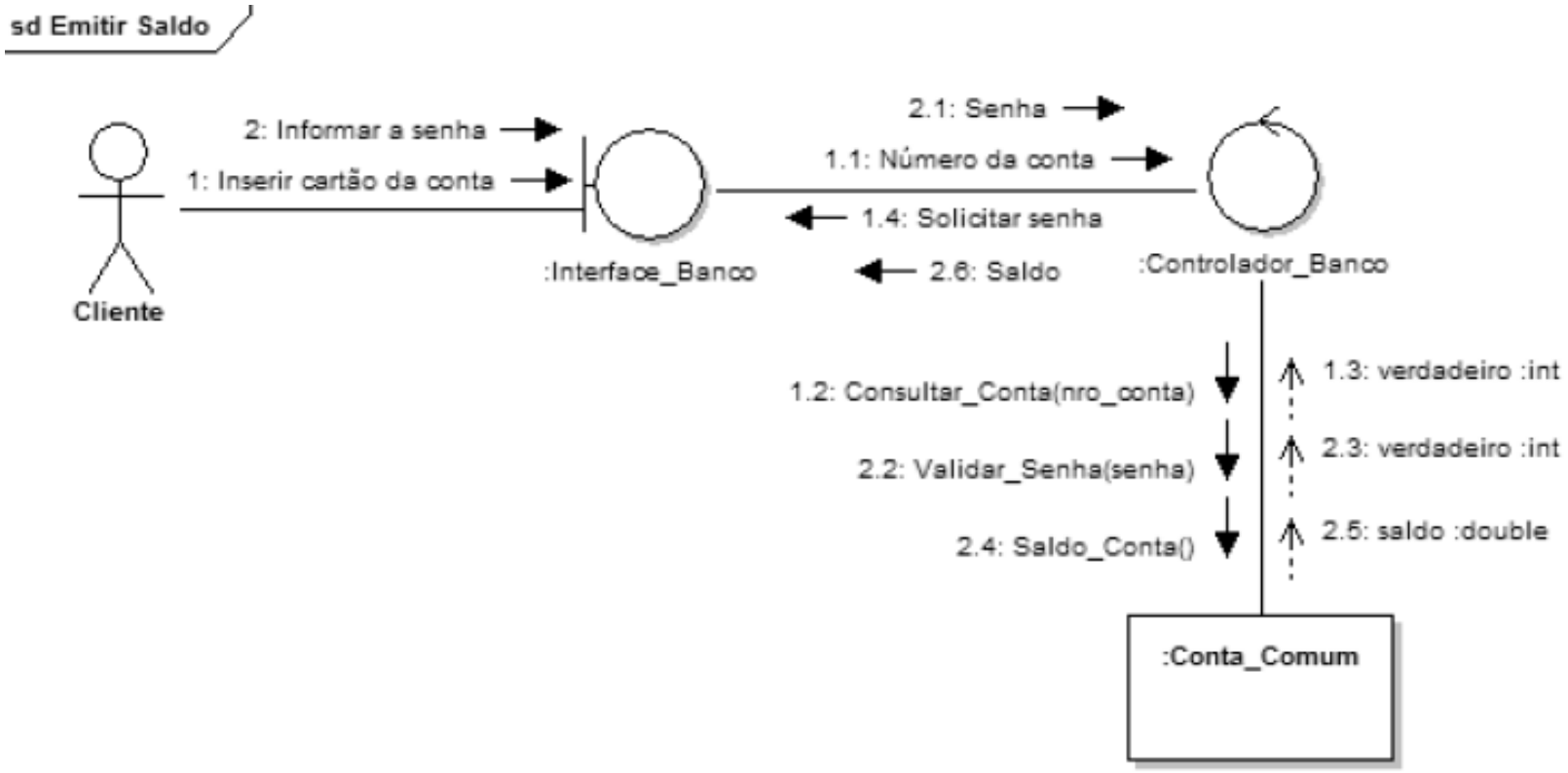
Exemplo de Diagrama de Sequência

Diagrama de Comunicação

- ✓ Associado ao diagrama de sequência, apresentando enfoque distinto deste

- ✓ Não se preocupa com a temporalidade do processo, mas sim, em como os elementos estão vinculados e quais mensagens trocam entre si

Diagrama de Comunicação



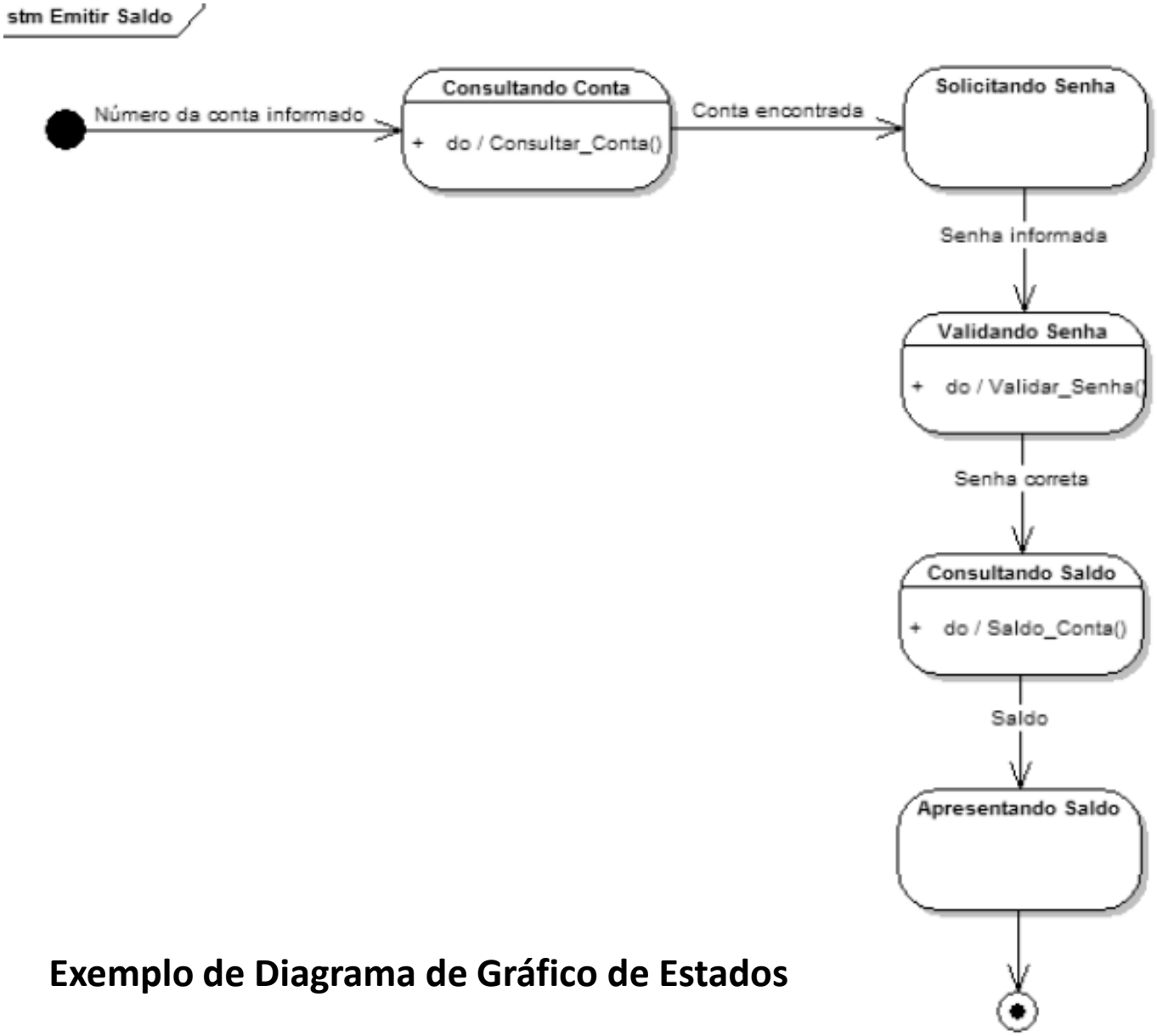
Exemplo de Diagrama de Comunicação

Diagrama de Máquina de Estados

✓ Demonstra o comportamento de um elemento por meio de um conjunto finito de transições de estado

✓ Pode basear-se em um caso de uso, mas também pode ser utilizado para acompanhar os estados de outros elementos, por exemplo, uma instância de uma classe

Diagrama de Máquina de Estados



Exemplo de Diagrama de Gráfico de Estados

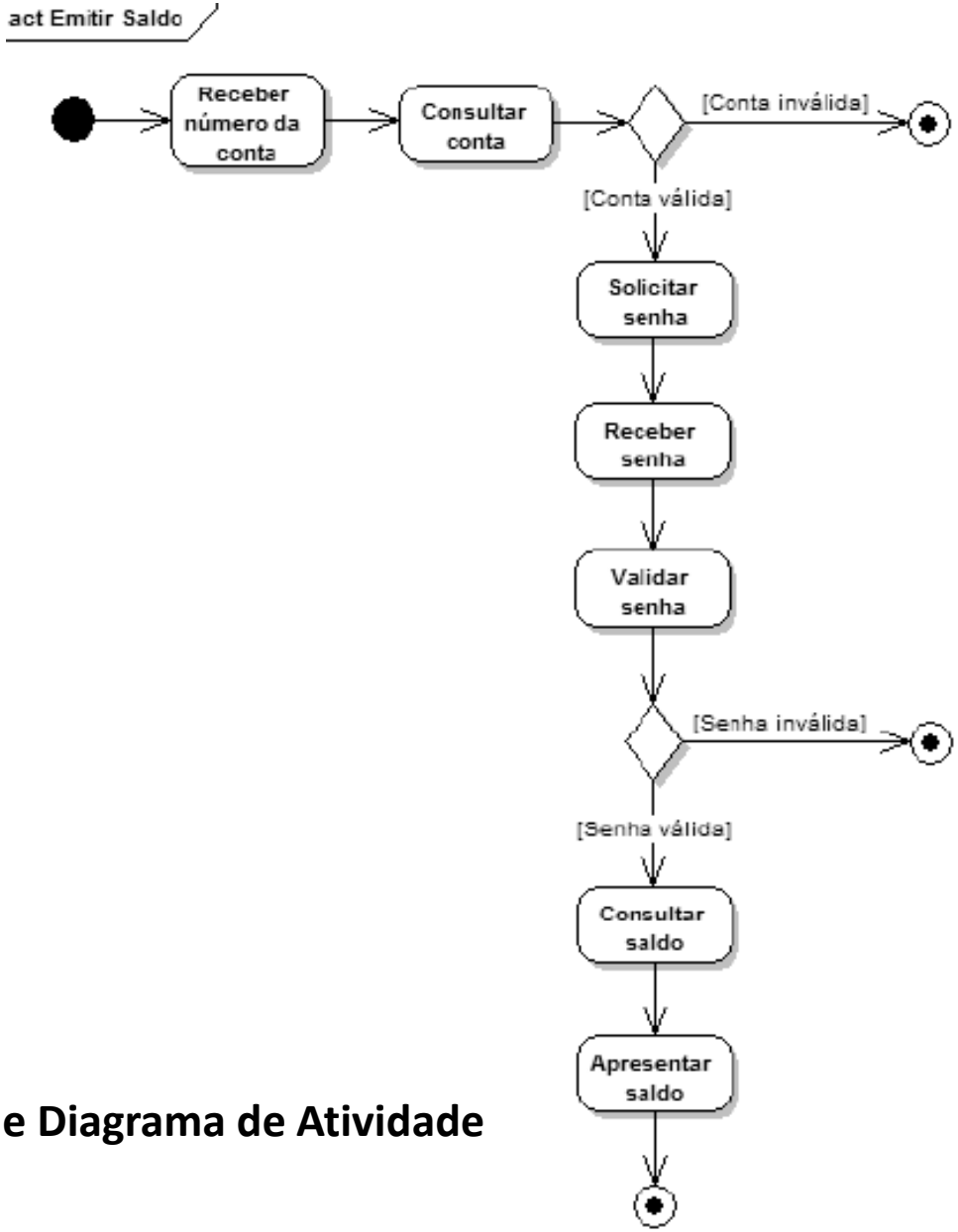
Diagrama de Atividade

✓ Descreve os passos a serem percorridos para a conclusão de uma atividade específica

✓ A atividade pode ser representada por um método com certo grau de complexidade, um algoritmo, ou mesmo um processo completo

✓ Concentra-se na representação do fluxo de controle de uma atividade

Diagrama de Atividade

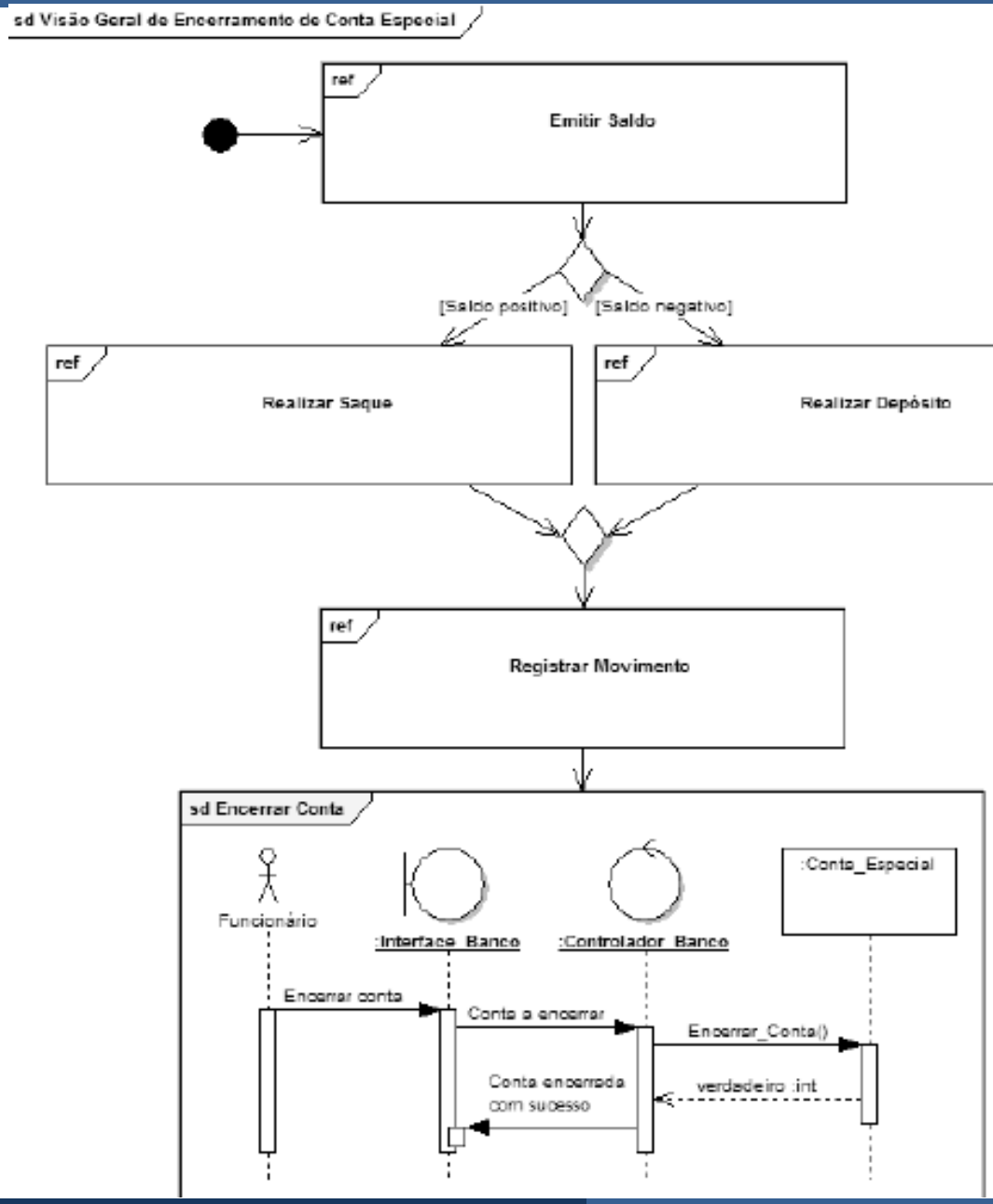


Exemplo de Diagrama de Atividade

Diagrama de Visão Geral de Interação

- ✓ Variação do diagrama de atividade
- ✓ Fornece uma visão geral dentro de um sistema ou processo de negócio

Diagrama de Visão Geral de Interação



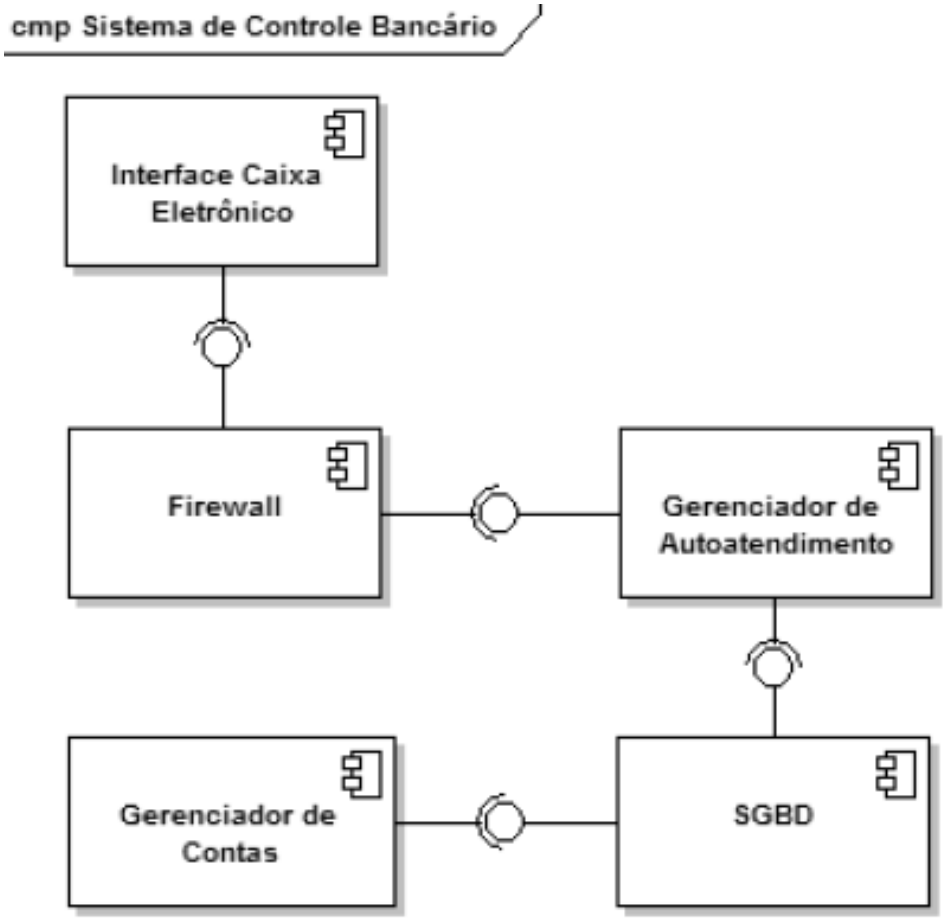
Exemplo de Diagrama de Visão Geral de Interação

Diagrama de Componentes

✓ Representa os componentes do sistema quando este for ser implementado em termos de módulos de código-fonte, bibliotecas, formulários, arquivos de ajuda, módulos executáveis etc.

✓ Determina como tais componentes estarão estruturados e irão interagir para que o sistema funcione de maneira adequada

Diagrama de Componentes



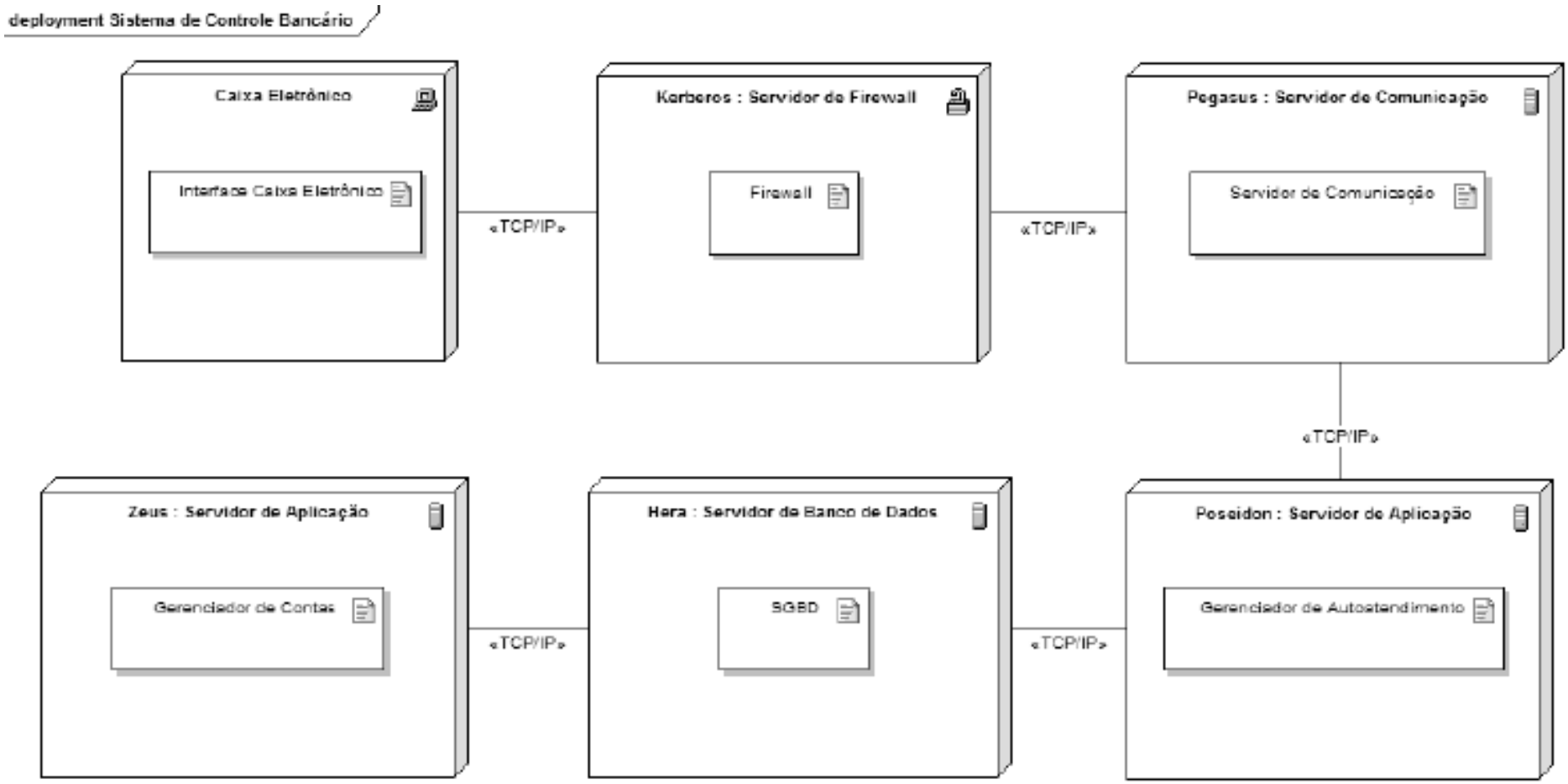
Exemplo de Diagrama de Componentes

Diagrama de Implantação

✓ Determinar as necessidades de hardware do sistema, isto é, todo o aparato físico sobre o qual o sistema deverá ser executado

✓ Servidores, estações, topologias, protocolos de comunicação etc.

Diagrama de Implantação



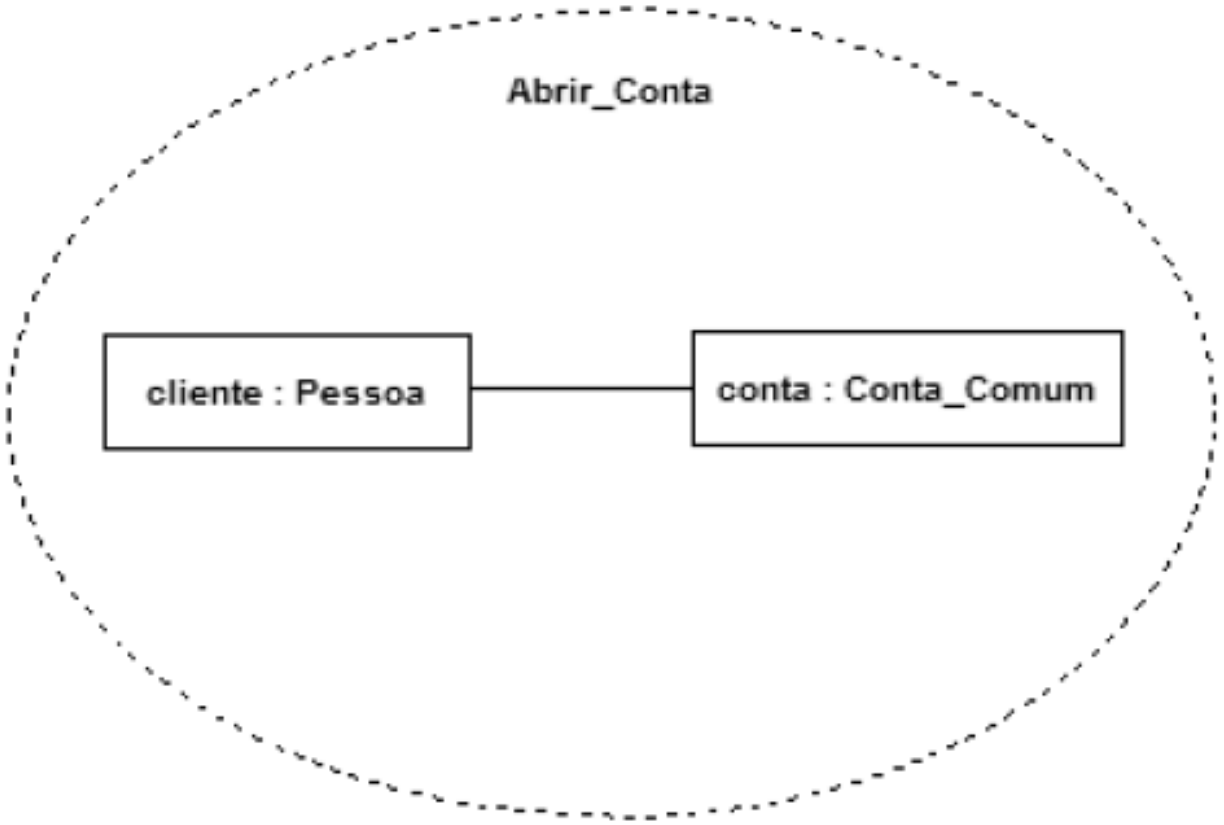
Exemplo de Diagrama de Implantação

Diagrama de Estrutura Composta

✓ Descreve a estrutura interna de um classificador, como uma classe ou componente

✓ Descreve a colaboração em que um conjunto de instâncias cooperam entre si para realizar uma tarefa

Diagrama de Estrutura Composta



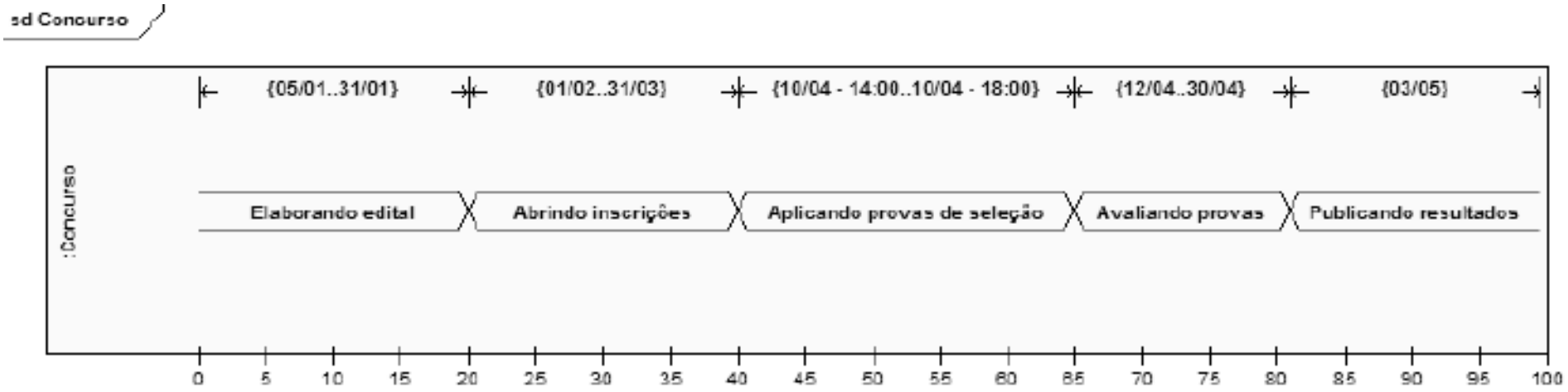
Exemplo de Diagrama de Estrutura Composta

Diagrama de Tempo ou de Temporização

✓ Descreve a mudança no estado ou condição de uma instância de uma classe ou seu papel durante um período

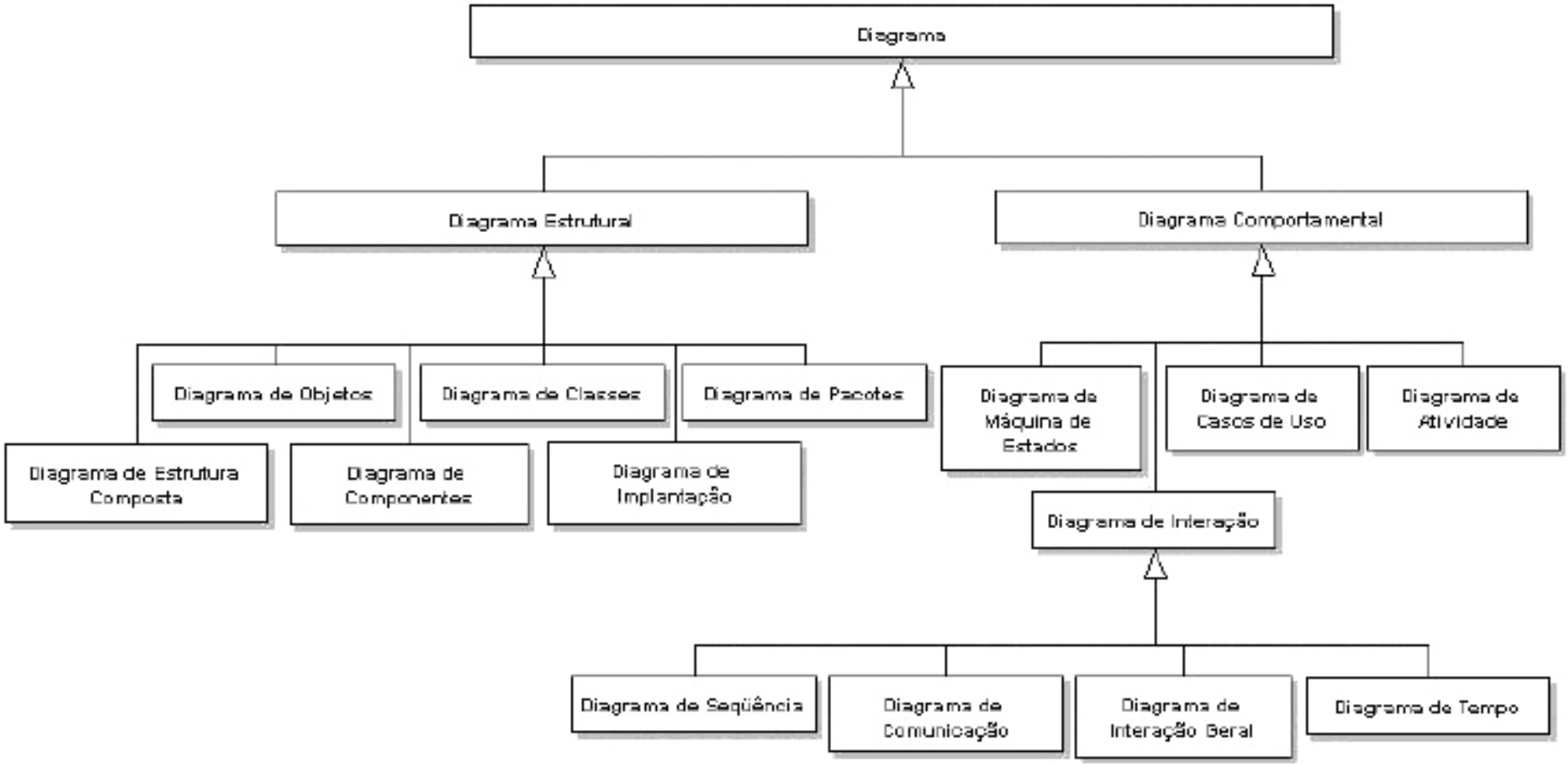
✓ Utilizado para demonstrar a mudança no estado de um objeto no tempo em resposta a eventos externos

Diagrama de Tempo ou de Temporização



Exemplo de Diagrama de Tempo

Síntese Geral dos Diagramas



Ferramentas CASE Baseadas na Linguagem UML

Computer-Aided Software Engineering

- ✓ **Enterprise Architect:** www.sparxsystems.com.au
- ✓ **Visual Paradigm for UML ou VP-UML:** www.visual-paradigm.com
- ✓ **Poseidon for UML:** www.gentleware.com
- ✓ **ArgoUML:** www.argouml.tigris.org
- ✓ **Astah:** www.astah.net