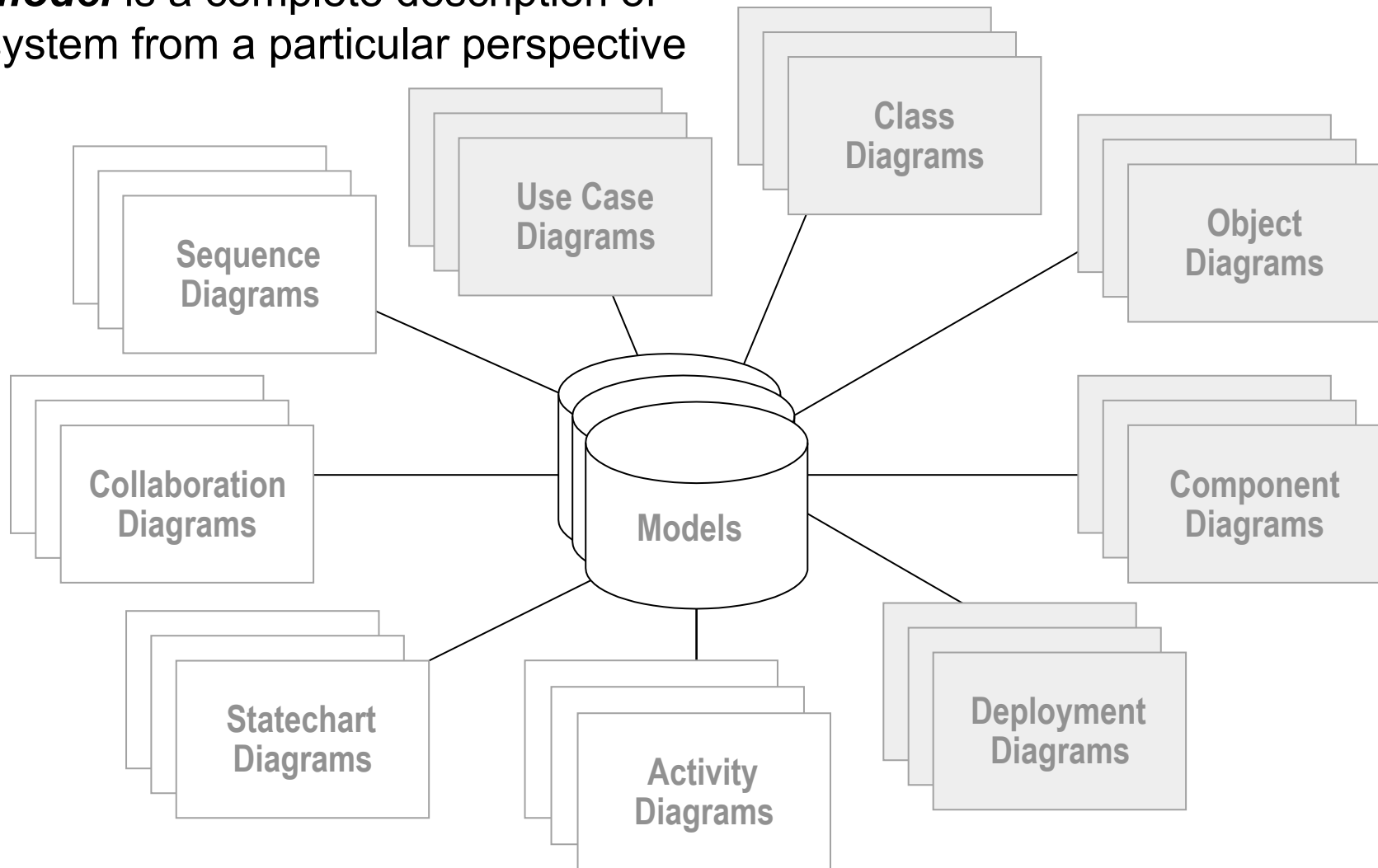


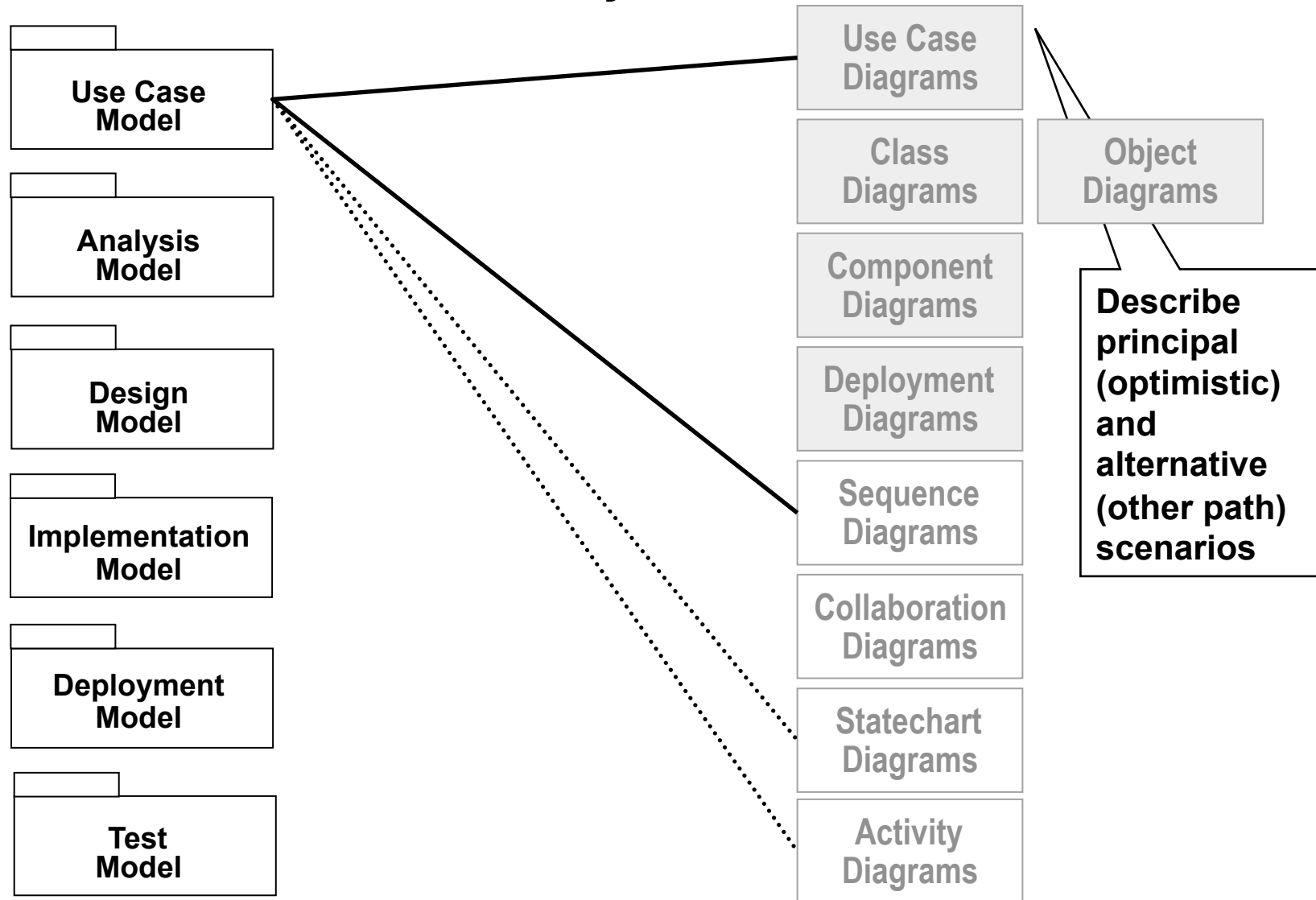
UML – Modelos e Diagramas

... Models and Diagrams

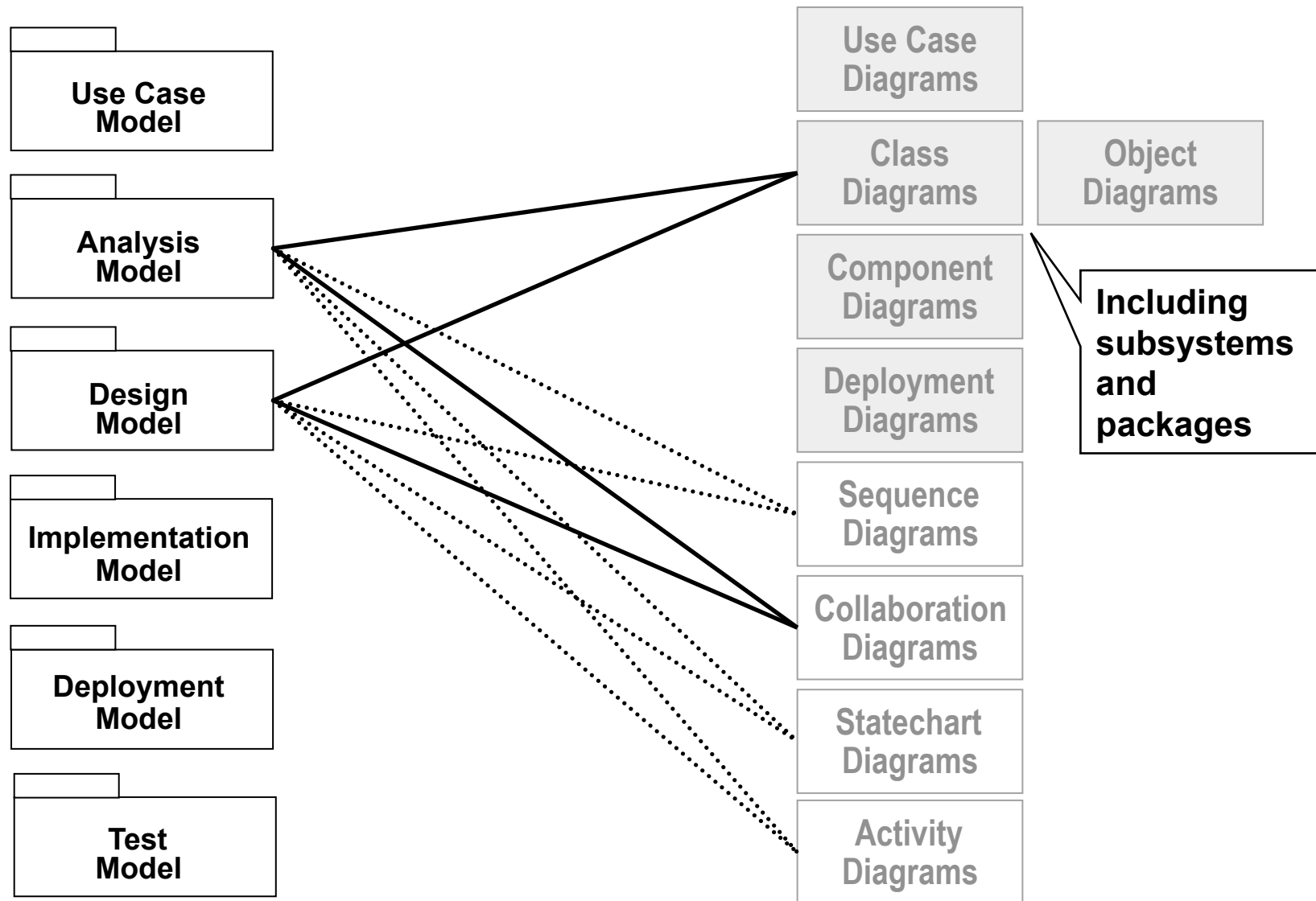
A **model** is a complete description of a system from a particular perspective



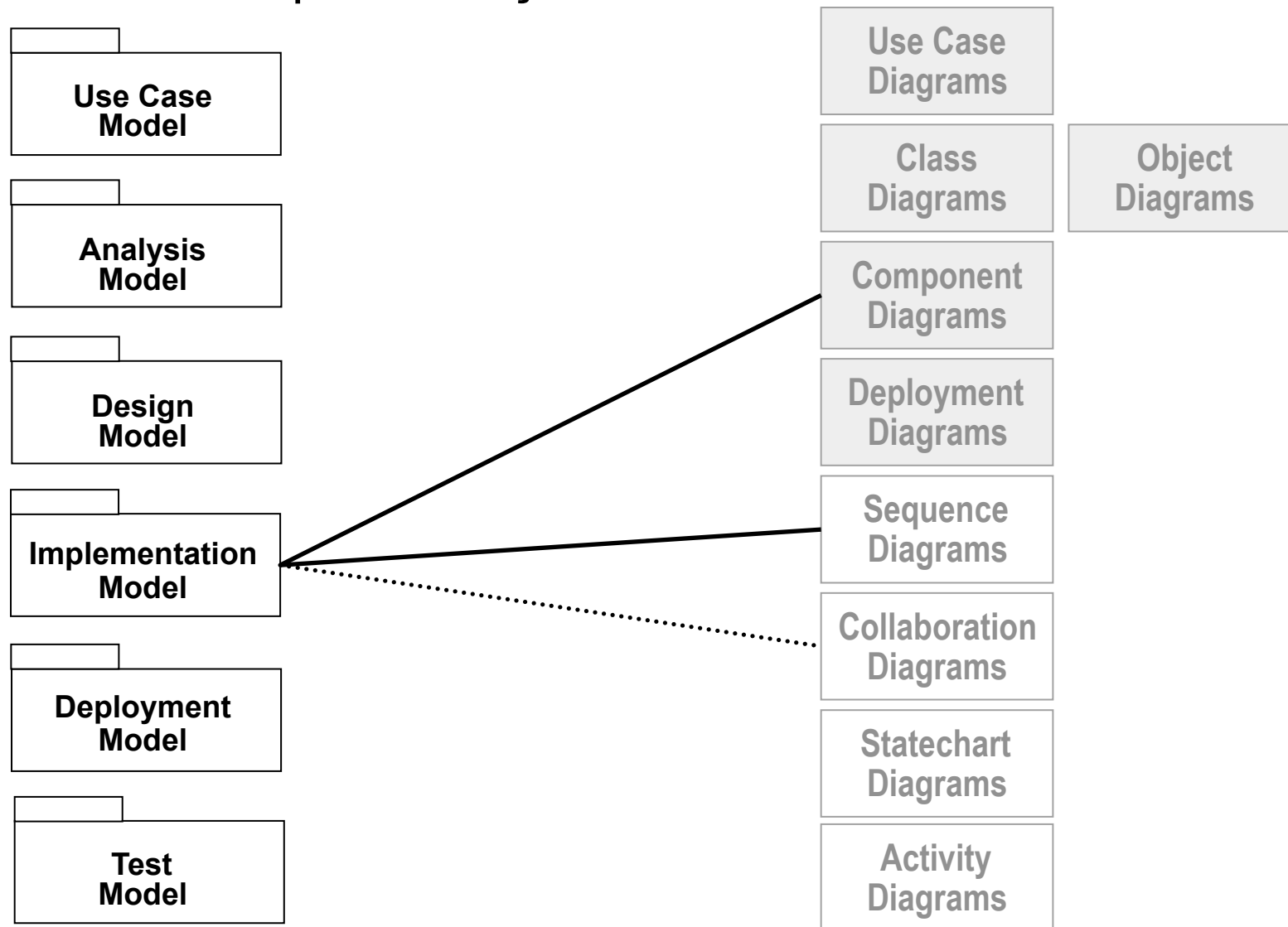
Modelo de Casos de Utilização



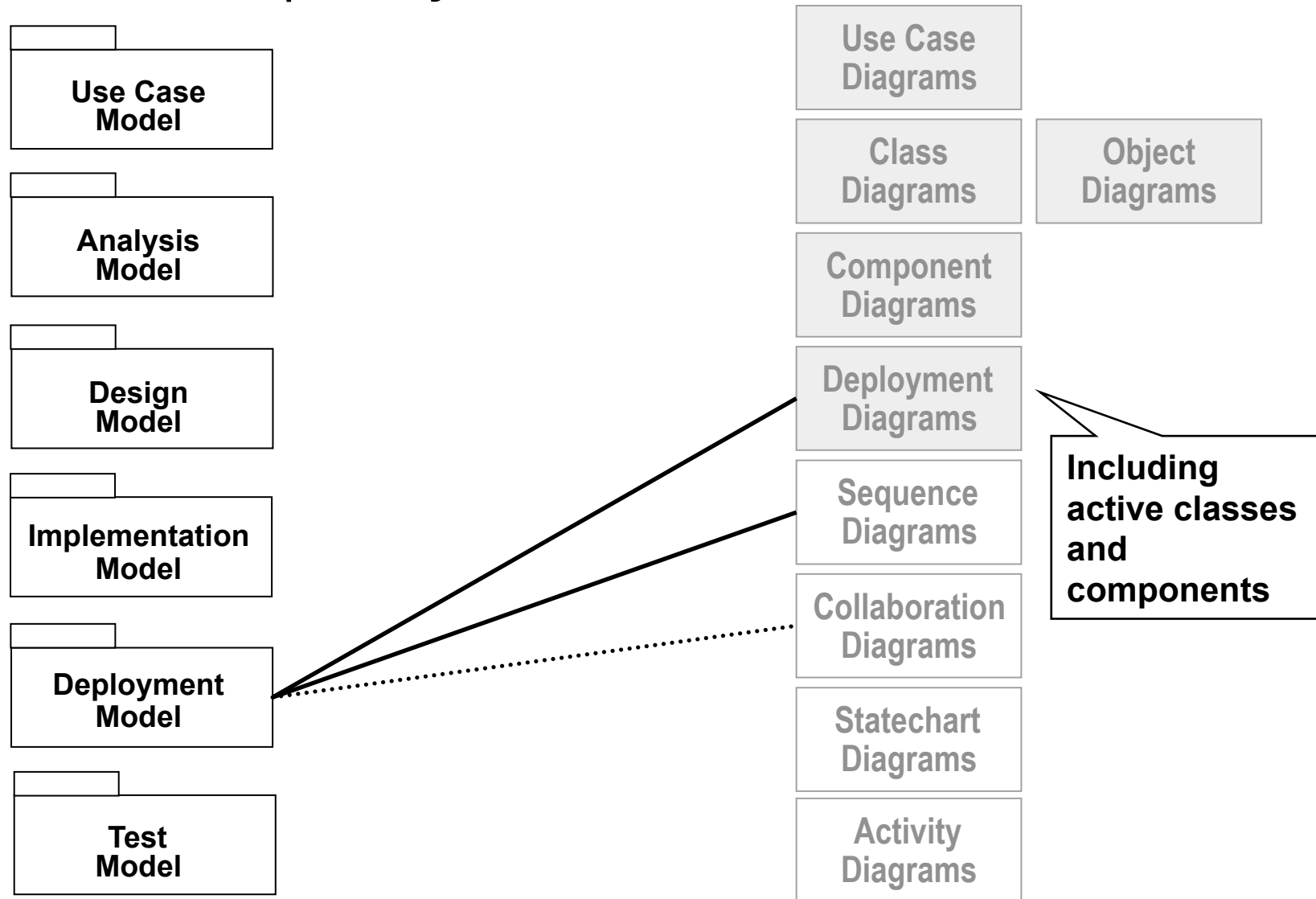
Modelo de Análise & Desenho



Modelo de Implementação



Modelo de Implantação



Modelo de Teste

Use Case Model

Analysis Model

Design Model

Implementation Model

Deployment Model

Test Model

Test model refers to all other models and uses corresponding diagrams

Use Case Diagrams

Class Diagrams

Object Diagrams

Component Diagrams

Deployment Diagrams

Sequence Diagrams

Collaboration Diagrams

Statechart Diagrams

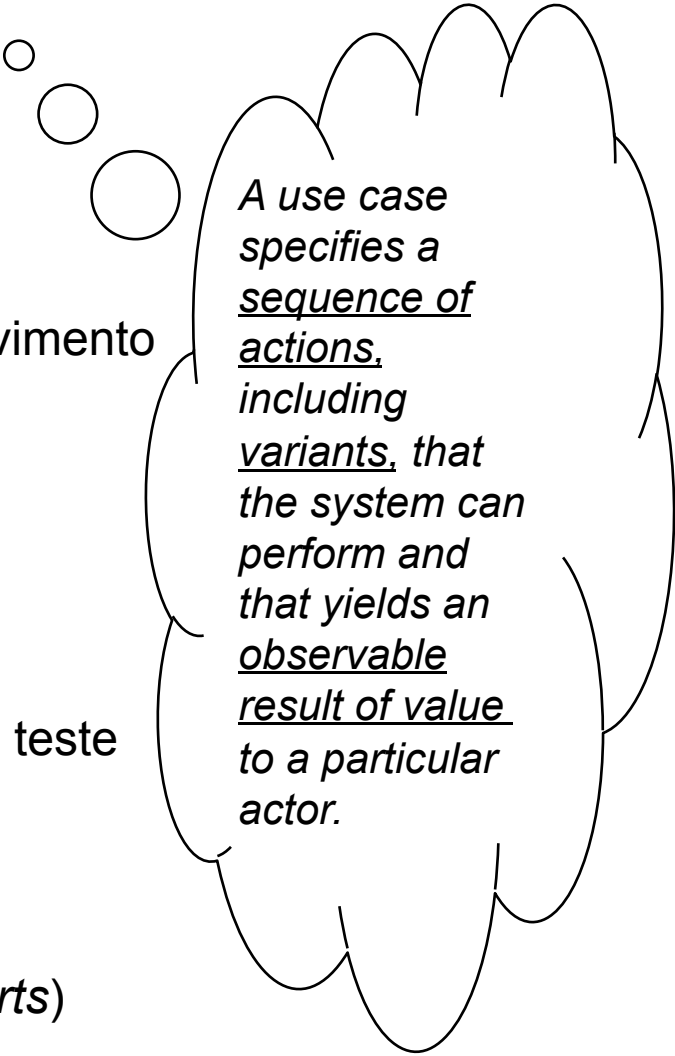
Activity Diagrams

Diagramas

- Cada Diagrama é uma Vista de um Modelo
 - apresenta a perspectiva de algum dos intervenientes no sistema
 - constitui uma representação parcial do sistema
 - é semanticamente consistente com as restantes Vistas
 - Na UML existem 9 espécies de Diagramas
 - Casos de Utilização (*Use Case*)
 - Classes (*Class*)
 - Objectos (*Object*)
 - Componentes (*Component*)
 - Implantação (*Deployment*)
 - Sequência (*Sequence*)
 - Colaboração (*Collaboration*)
 - Transição de Estados (*Statechart*)
 - Actividade (*Activity*)
- Vistas dos Modelos de Estrutura (*Static Views*)
- Vistas dos Modelos de Comportamento (*Dynamic Views*)

Diagramas de Casos de Utilização

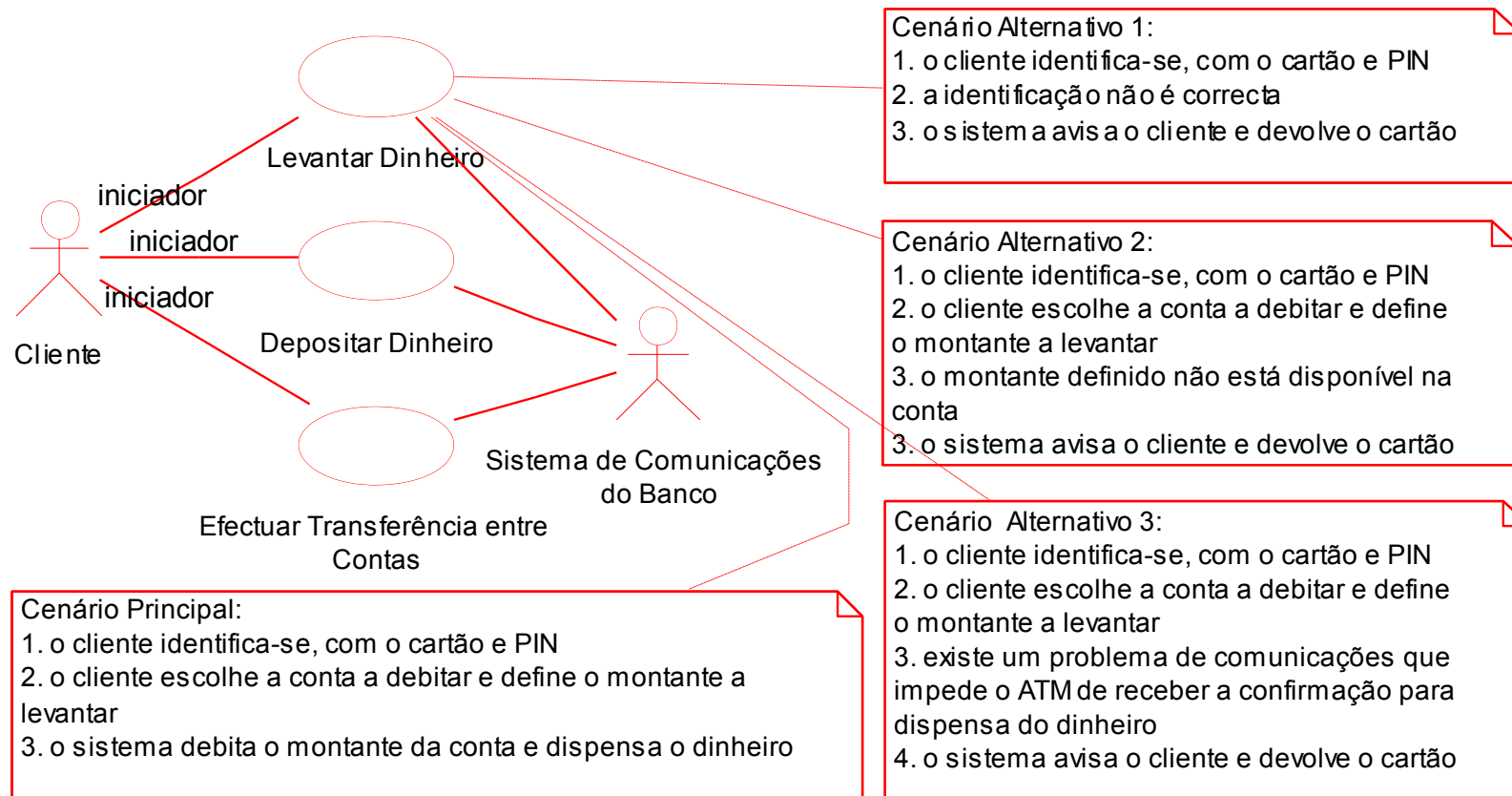
- Capturam a funcionalidade do sistema
 - na perspectiva dos seus utilizadores
- São construídos
 - nas fases iniciais do processo de desenvolvimento
- Têm como objectivo
 - especificar o contexto do sistema
 - capturar os requisitos do sistema
 - elaborar uma arquitectura para o sistema
 - orientar a implementação e gerar casos de teste
- São desenvolvidos por
 - analistas (*analysts*) e
 - peritos do domínio / negócio (*domain experts*)



A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor.

Modelo de Casos de Utilização – Diagrama

- Considere um sistema ATM (*Automatic Teller Machine*) – multibanco
 - "O cliente do banco utiliza o ATM para levantar e depositar dinheiro das suas conta e para efectuar transferências entre diferentes contas"



Diagramas de Classes

- Capturam o vocabulário do sistema
 - usando a terminologia dos seus utilizadores
 - usando as convenções e técnicas dos analistas
- São construídos e refinados
 - nas diversas fases do processo de desenvolvimento
- Têm como objectivo
 - concretizar com nomes e modelos, os conceitos no sistema
 - especificar as colaborações (partindo dos Casos de Utilização)
 - especificar esquemas lógicos de bases de dados
- São desenvolvidos por
 - analistas (*analysts*)
 - analistas / especialistas em desenho (*designers*)
 - programadores (*implementers*)

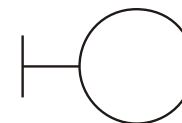
Diagramas Casos de Utilização ... Diagramas de Classes

- A construção de Diagramas de Casos de Utilização
 - é um modo intuitivo e sistemático de capturar **requisitos** funcionais
 - ... com ênfase no "valor acrescentado" aos seus utilizadores / clientes
- ... do Diagrama de Casos de Utilização, para o de Classes fazer,
 - olhar para um *use case* de cada vez
 - ler cada um dos seus cenários
 - criar uma sua realização – *use case realization*
 - eventualmente uma já anteriormente definida
 - ... esta realização faz a ligação com a fase de análise ...
 - identificar, no *use case realization*, classes de cada um de 3 estereótipos
 - *boundary* – a fronteira do sistema com os atores
 - *entity* – a representação do que é persistente no *use case*
 - *control* – a perspectiva de análise do *use case*

Casos de Utilização ... estereótipos de Classes – *boundary*

- *boundary* – "... *should be related to at least one actor and vice versa*"
 - usado para modelar a interacção entre o sistema e os seus actores
 - interacção envolve, normalmente,
 - recepção e apresentação de informação a actores
 - atender e efectuar pedidos a actores
 - não precisa ser específico de um único *use case*
 - é usual ser criado e destruído no contexto de determinado *use case*
 - podendo, ao longo da vida, participar em diversos *use case*
- Representam normalmente abstracções de,
 - janelas, formulários, terminais, sensores, interfaces de comunicação, ...
 - e APIs – *Application Programming Interface* (eventualmente não O. O.)

"... boundary classes need not describe how the interaction is physically realized, since this is considered in subsequent design and implementation activities."

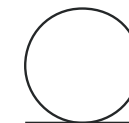


NomeClasse

Casos de Utilização ... estereótipos de Classes – *entity*

- *entity* – "... isolate changes to the information they represent"
 - usado para modelar informação persistente (com longa vida ...)
 - estado e comportamento de algum fenómeno ou conceito como,
 - um indivíduo
 - um objecto do mundo real
 - um evento do mundo real
 - é usual ter uma vida longa e participar em diversos *use case*
 - normalmente "sobrevivem" à terminação do *use case*
- Representam normalmente,
 - a estrutura lógica dos dados
 - contribui para identificar a informação da qual o sistema depende

"... an entity object need not be passive and may sometimes have complex behaviour related to the information it represents."

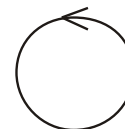


NomeClasse

Casos de Utilização ... estereótipos de Classes – *control*

- *control* – "... *dynamics of the system are modeled by control classes*"
 - usado para, estabelecer canais de comunicação entre objectos
 - p.e. construindo os *links* entre eles,
 - coordenar actividades; construir sequências de acções,
 - definir os mecanismos de transacção; controlar os outros objectos
 - usado para encapsular o controlo de um use case específico
 - é usual ser criado ao iniciar o *use case*; ser destruído quando ele acaba
 - excepções: o *control* participar em mais que um *use case*; diversos participarem num mesmo *use case*; o *use case* não precisar dele
- Normalmente usado para modelar a dinâmica do sistema,
 - coordenando as acções principais e controlando o controlo de fluxo
 - e delegando trabalho nos outros objectos (*boundary* e *entity*)

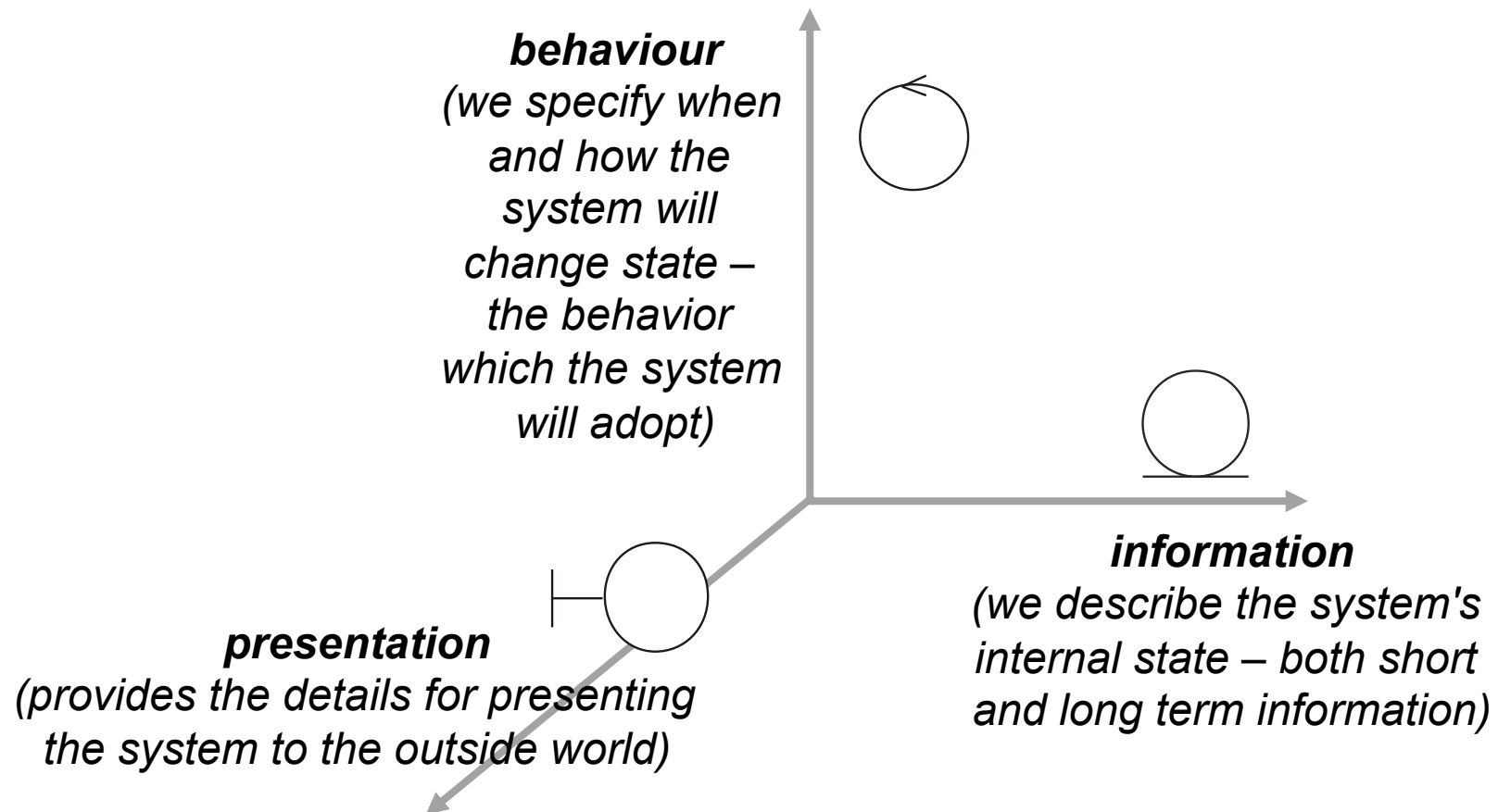
"... *control classes* are also used to represent complex derivations and calculations, such as business logic, that cannot be related to any specific, long-lived information stored by the system."



NomeClasse

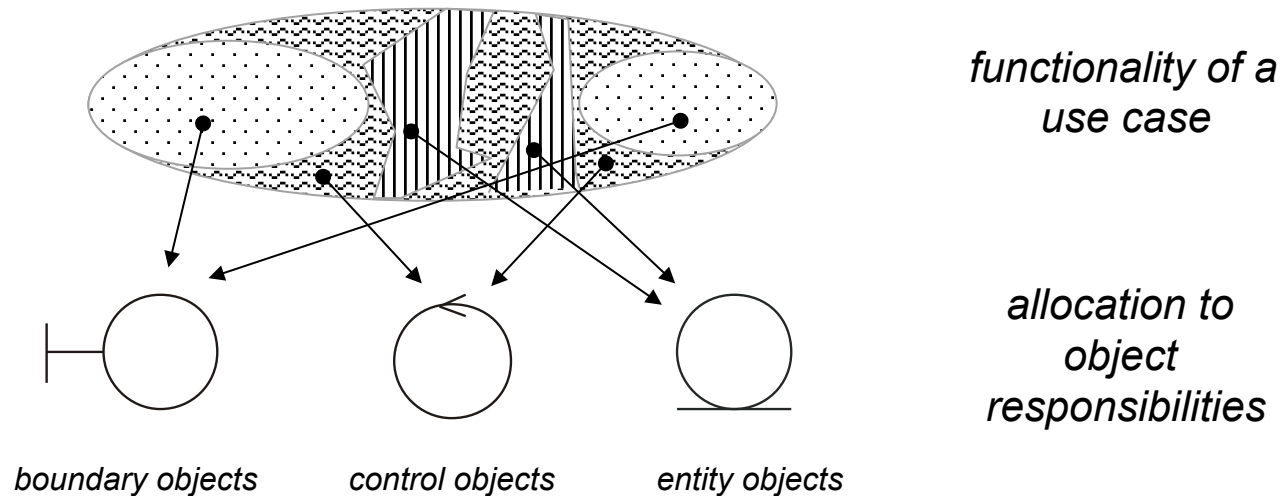
... information, behavior and presentation

- "... our aim is to capture information, behavior and presentation"
 - Ivar Jacobson, em *Object-Oriented Software Engineering – A Use Case Driven Approach*; 1992.



... *the Analysis Model is formed from Use Case Model*

- "...each use case will be entirely divided into those 3 types of objects"
 - Ivar Jacobson, em *Object-Oriented Software Engineering – A Use Case Driven Approach*; 1992.

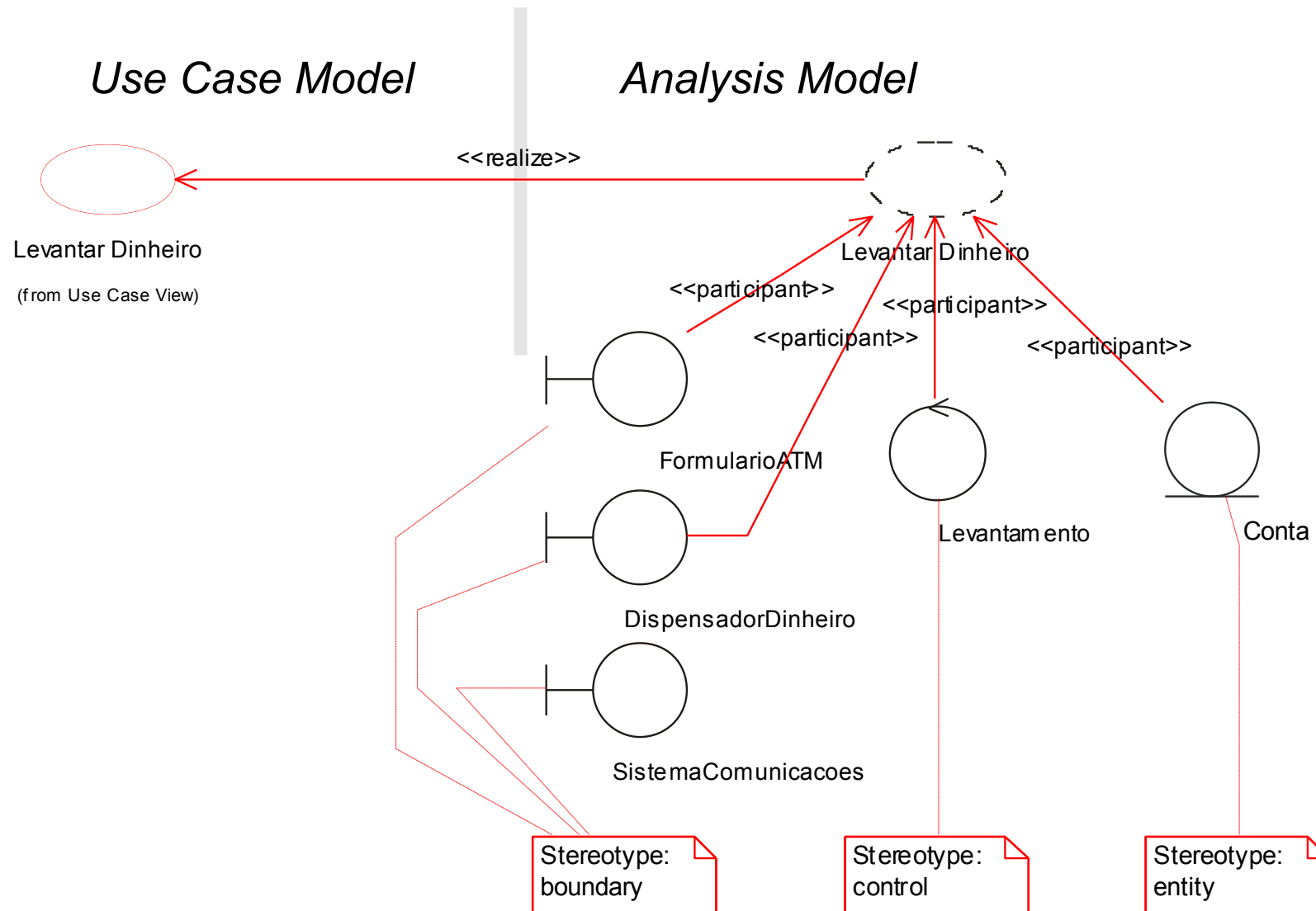


"... we model with these 3 object types to have localization in changes to the system"

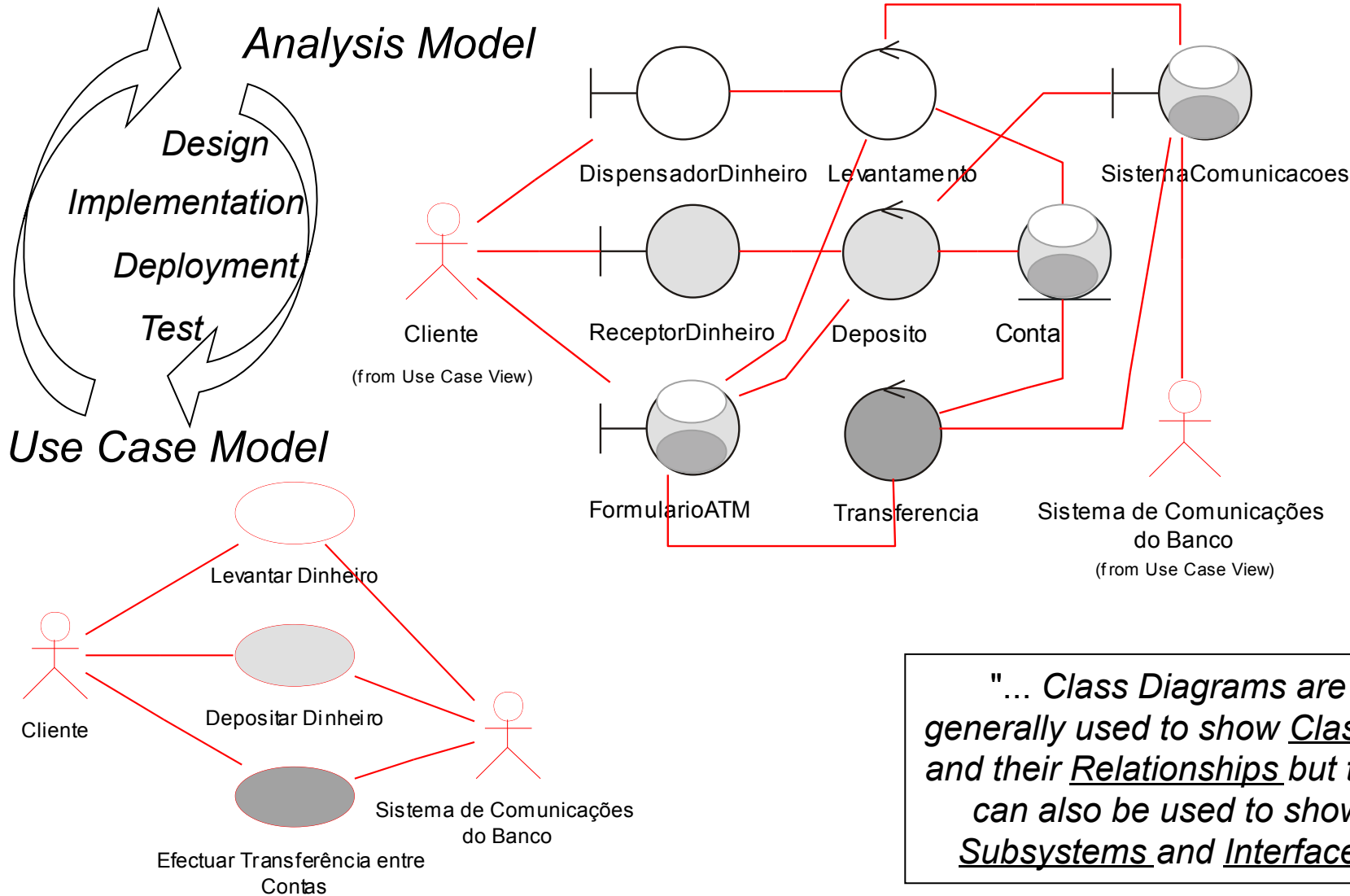
"... how do we know that these are the 'right' object types ?"

"... this is something you cannot draw any conclusions about until a system has been changed a number of times – that is from accumulated project experience"

Modelo de Análise – Diagrama de Classes



Modelo de Análise – Diagrama de Classes (cont.)

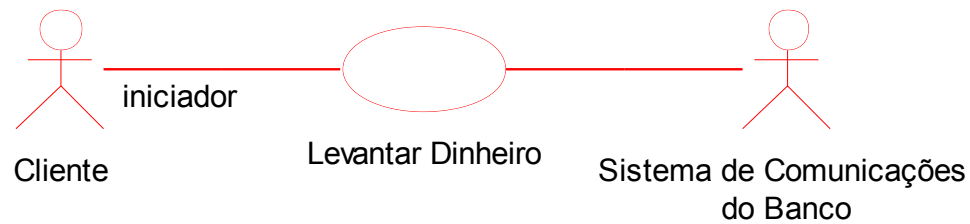


"... Class Diagrams are generally used to show Classes and their Relationships but they can also be used to show Subsystems and Interfaces"

Diagramas de Objectos

- Capturam instâncias (*instances*) e ligações (*links*)
 - instâncias – concretizações de Classes
 - ligações – concretizações de Relações
- São construídos e refinados
 - durante as fases de Análise (*Analysis*) e Desenho (*Design*)
- Têm como objectivo
 - ilustrar a estrutura de objectos (ou de dados)
 - especificar instantâneos (*snapshots*) (da interacção entre Objectos)
- São desenvolvidos por
 - analistas (*analysts*)
 - analistas / especialistas em desenho (*designers*)
 - programadores (*programmers*)

Modelo de Análise – Diagrama de Objectos



Cenário Principal:

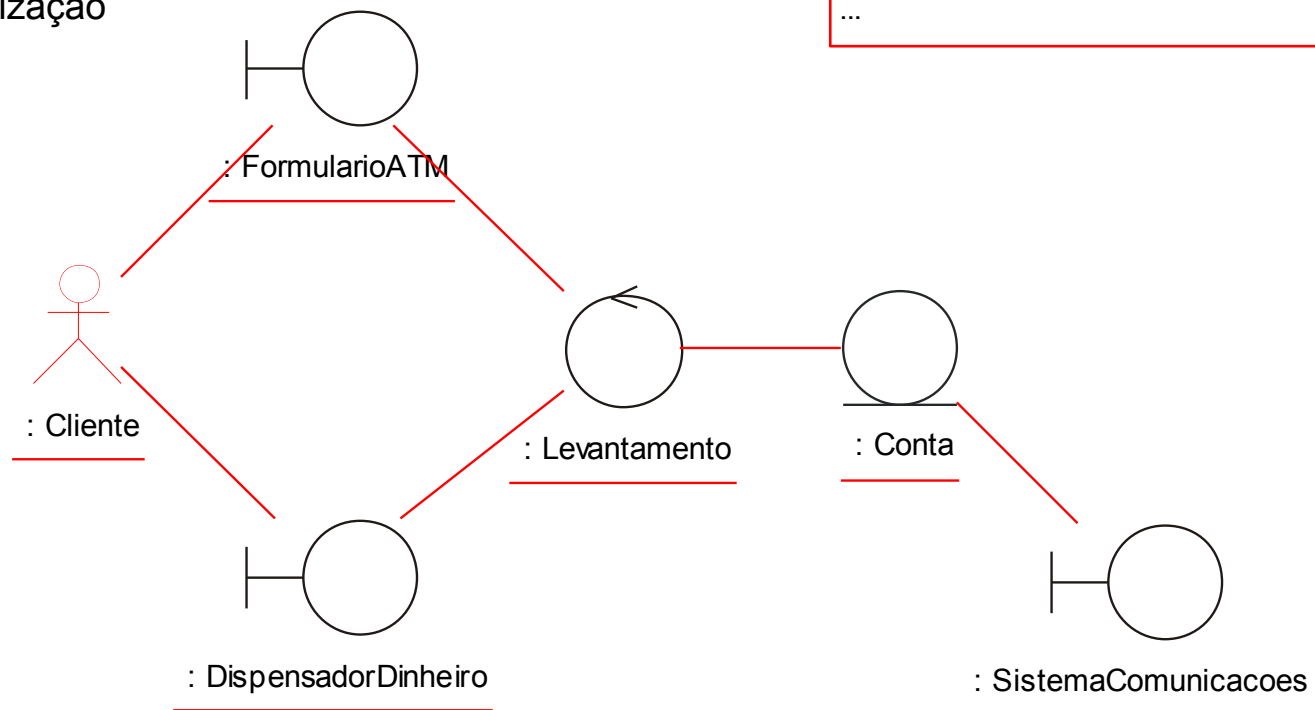
1. o cliente identifica-se, com o cartão e PIN
2. o cliente escolhe a conta a debitar e define o montante a levantar
3. o sistema debita o montante da conta e dispensa o dinheiro

Cenário Alternativo 1, 2, 3, ...

...

Diagrama de Casos de Utilização

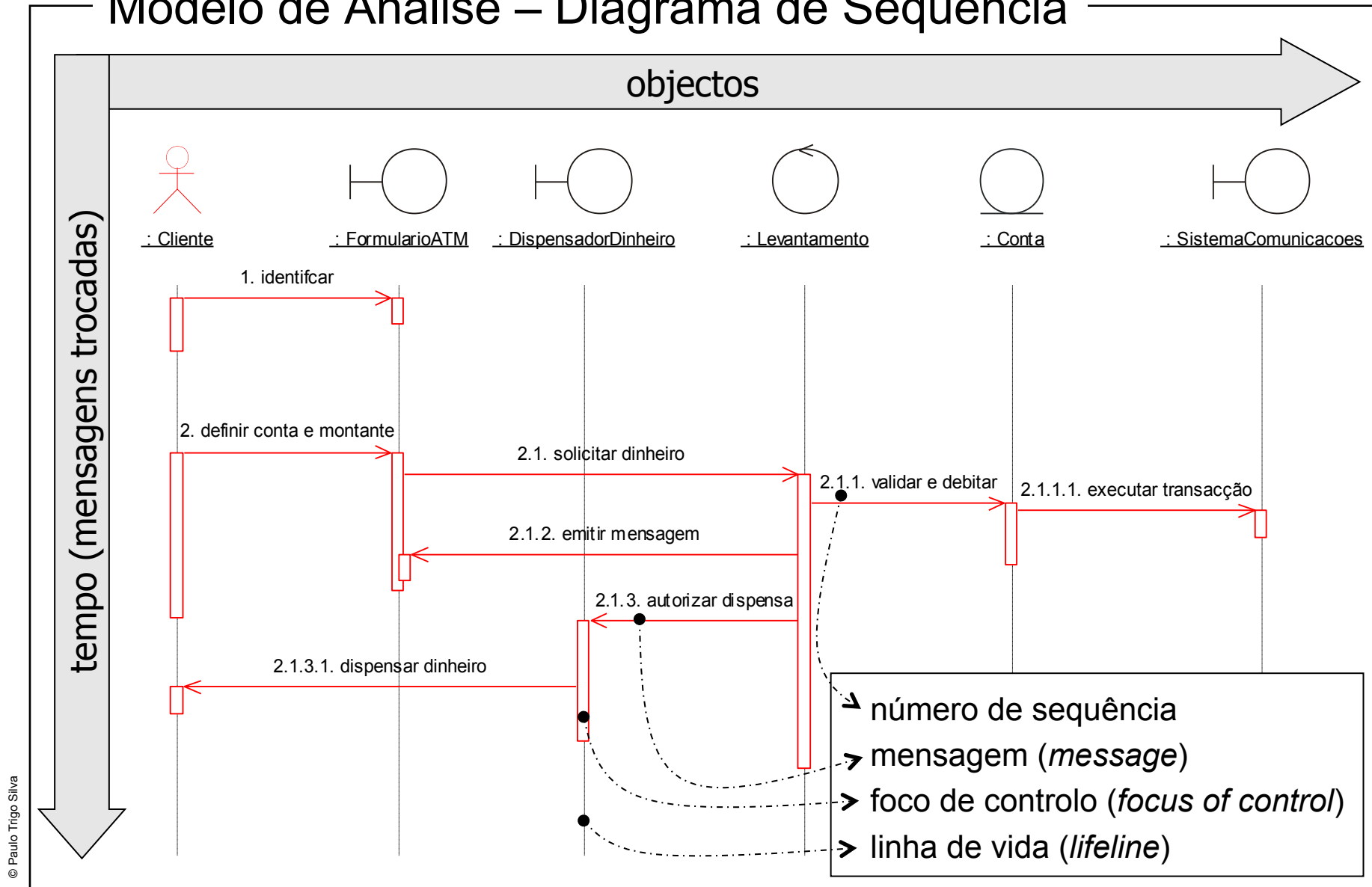
Diagrama de Objectos



Diagramas de Interação – Sequência e Colaboração

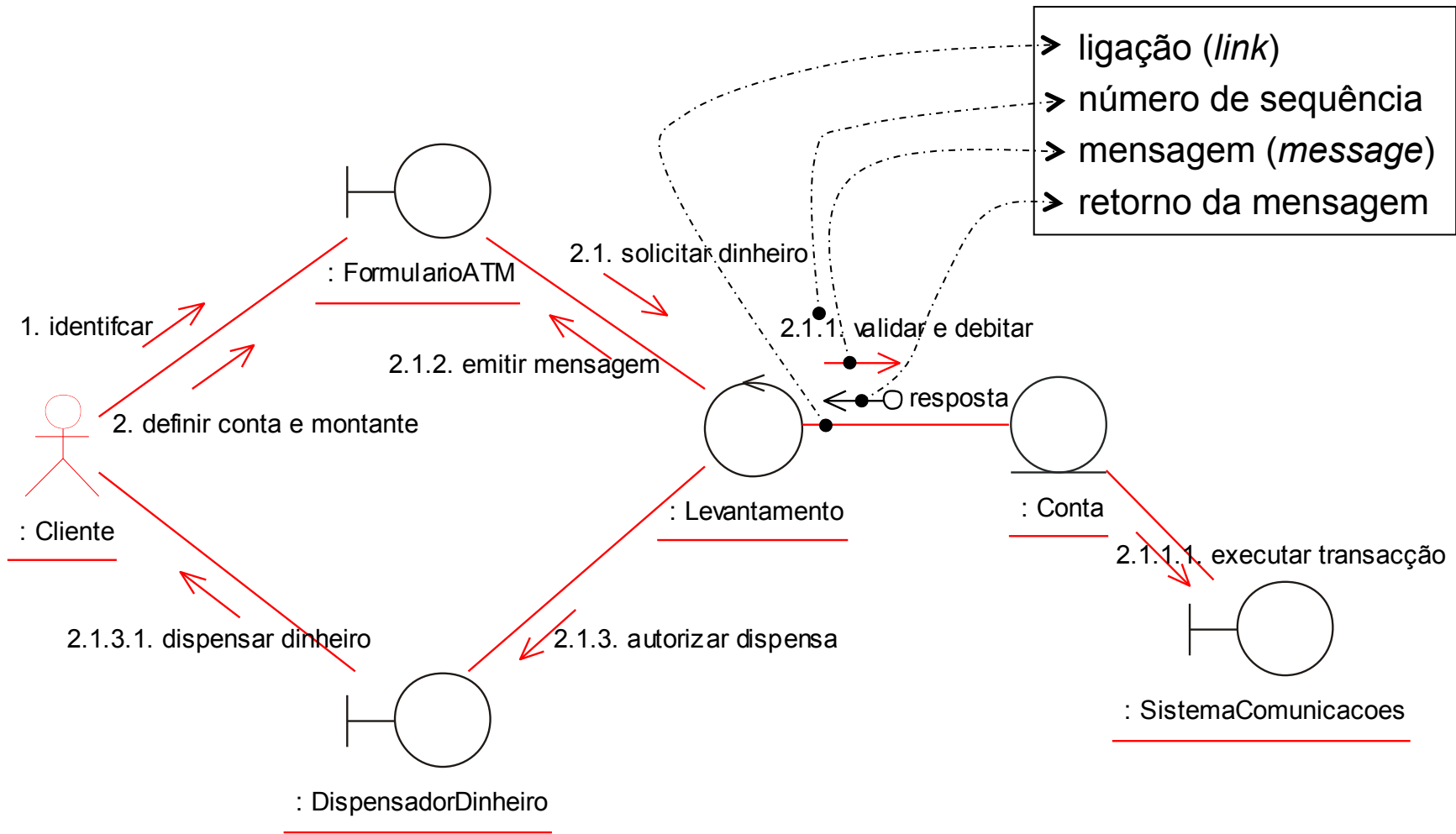
- Capturam a comportamento dinâmico do sistema
 - Sequência
 - ênfase na ordem temporal das mensagens (*time oriented*)
 - Colaboração
 - ênfase nas ligações entre objectos (*message oriented*)
- Sequência – têm como objectivo
 - modelar o fluxo de controlo
 - ilustrar os cenários (os dos Casos de Utilização) e outros típicos
- Colaboração – têm como objectivo
 - modelar o fluxo de controlo
 - ilustrar o modo como acontece a coordenação entre objectos

Modelo de Análise – Diagrama de Sequência



Modelo de Análise – Diagrama de Colaboração

Semanticamente equivalente ao Diagrama de Sequência



Modelo de *Análise versus* Modelo de Desenho

- Modelo de Análise (*Analysis Model*)
 - descreve o sistema quanto à sua função
 - *O que é oferecido a cada utilizador ?*
 - constrói uma percepção detalhada dos requisitos
- Modelo de Desenho (*Design Model*)
 - descreve o sistema quanto à sua forma
 - *Quais os blocos e qual a sua interacção ?*
 - tem como *input* o Modelo de Análise
 - visa aprofundar aspectos não funcionais e restrições
 - linguagens de programação; sistemas operativos; tecnologias de suporte à concorrência e distribuição; tecnologias de bases de dados; gestão de transacções; interfaces pessoa-máquina; ...
 - visa definir uma forma (arquitectura) que contemple,
 - os aspectos funcionais
 - os aspectos não funcionais e restrições

Modelo de Análise *versus* Modelo de Desenho (cont.)

Modelo de Análise	Modelo de Desenho
Conceptual - é uma abstracção do sistema e evita aspectos de implementação	Físico - é um plano a ser usado na implementação
Genérico - aplicável a diferentes Modelos de Desenho (<i>Design generic</i>)	Específico - não é genérico; é específico de uma implementação
Três estereótipos (conceptuais) de Classes: <<boundary>>, <<control>>, <<entity>>	Qualquer número de estereótipos (físicos) de Classes, dependendo da linguagem de implementação
Menos formal	Mais formal
Menos "caro" de construir (razão de 1:5 para o Modelo de Desenho)	Mais "caro" de construir (razão de 5:1 para o Modelo de Análise)
Construído essencialmente como resultado de anotações em reuniões (<i>leg work in workshops and the like</i>), ...	Construído essencialmente por "programação visual" em "ambiente de engenharia"; é feito em iteração (<i>round-trip engineering</i>) com o Modelo de Implementação
Pode não ser mantido ao longo de todo o ciclo de desenvolvimento	Deve ser mantido ao longo de todo o ciclo de desenvolvimento
Define uma estrutura que é o <i>input</i> essencial para se conseguir definir a forma do sistema	Contrói a forma do sistema tentando manter, tanto quanto possível, a estrutura definida pelo Modelo de Análise

Classe no Modelo de Desenho – *Design Class*

- Linguagem de especificação
 - a linguagem de programação (l.p.) usada para implementação
 - na sintaxe da l.p., definem-se, atributos, operações, parâmetros, tipos, ...
- Visibilidade de atributos e operações
 - usando e.g., *public*, *protected*, *private* – Java, C++, Python, ...
- Relações em que a Classe participa
 - generalização – linguagens de programação O.O. implementam conceito
 - associações – comum serem concretizadas através de variáveis (atributos) cujos valores serão referências para os objectos relacionados
 - agregações – concretização idêntica à da associação, no entanto, nas agregações de semântica forte, o objecto que agrega dá contexto aos objectos agregados – "estes só existem no contexto de quem agrega"

Classe no Modelo de Desenho – *Design Class* (cont.)

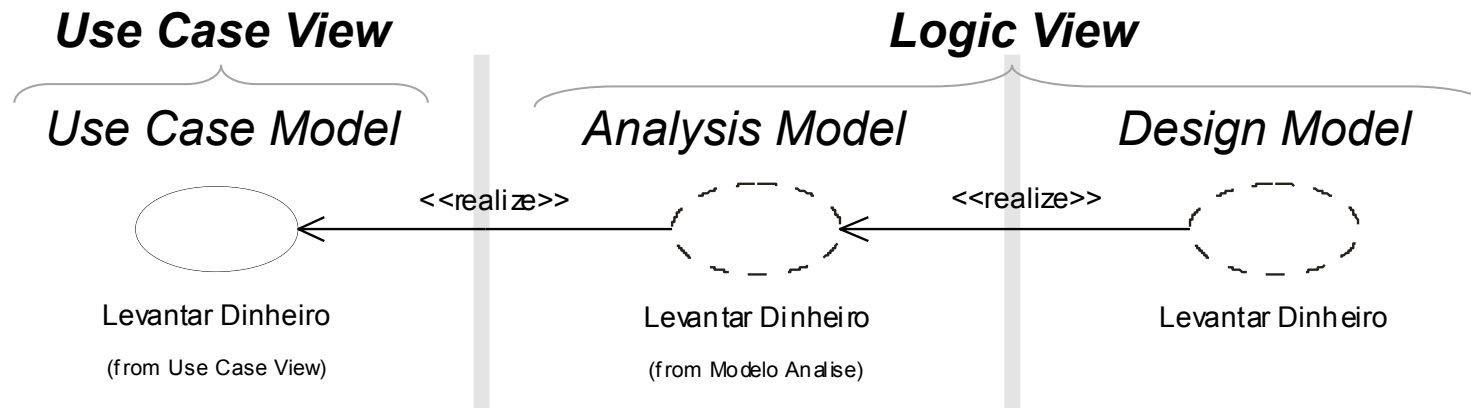
- Operações
 - correspondência nos métodos das linguagens de programação O.O.
 - especificação de métodos – no Modelo de Desenho, pode ser feita,
 - em língua natural ou pseudo-código
 - e usada como anotação às implementações do método
 - especificação de métodos
 - principal abstracção entre Modelos de Desenho e Implementação
 - no entanto, raramente é necessária, pois se recomenda que o mesmo técnico desenhe e implemente uma classe
- Estereótipo
 - pode ter um qualquer, específico das opções de implementação
 - *Java* – p.e. <<servlet>>
 - *Visual Basic* – p.e. <<class module>>, <<form>>, <<user control>>
 - *SGBD* – p.e. <<table>>

Classe no Modelo de Desenho – *Design Class* (cont. 1)

- Interface
 - uma *Design Class* pode implementar (fornecer) interfaces, se fizer sentido fazer isso na linguagem de programação adoptada
 - e.g., em *Java* uma *Design Class* pode fornecer uma interface
- Classe Activa – cada objecto executa em concorrência com outros
 - normalmente uma *Design Class*, não é activa,
 - ou seja, os seus objectos executam sob controlo de outro objecto
 - devido às diferenças semânticas entre Classes Activas e não Activas
 - ... as Activas podem "residir" no Modelo de Processamento (*Process Model*), em vez de estar no Modelo de Desenho (*Design Model*)
 - ... essa abordagem é útil quando existem várias Classes Activas, cujos objectos têm interacções complexas – e.g., sistemas de tempo real

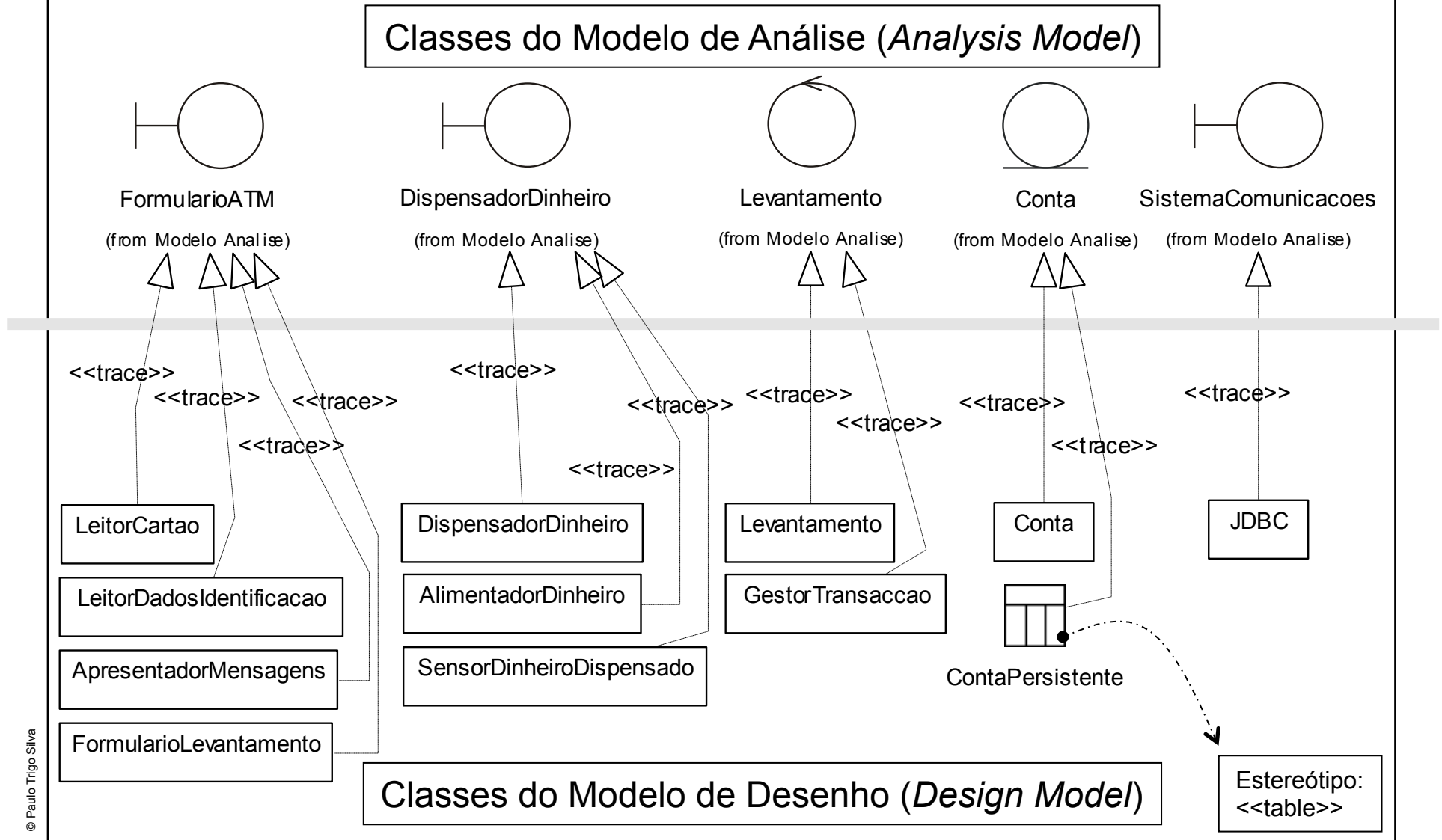
Vistas, Modelos e realização de Casos de Utilização

- A realização de Casos de Utilização – *use case realization*
 - em diferentes modelos serve diferentes objectivos

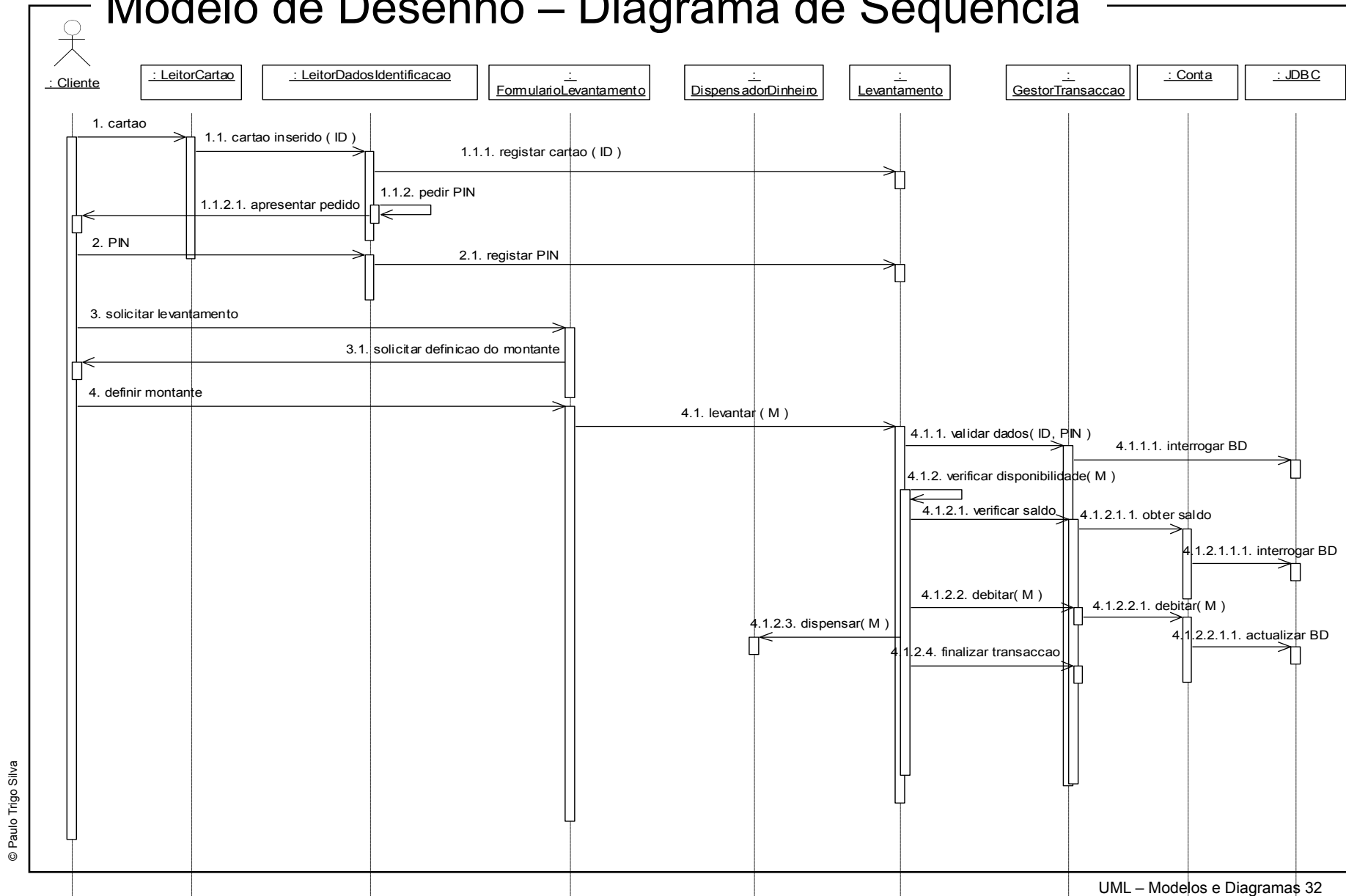


- Exemplo do ATM
 - Classes do Modelo de Análise
 - `FormularioATM`, `DispensadorDinheiro`, `Levantamento`, `Conta`, `SistemaComunicacoes`
 - com base nessas Classes, no Modelo de Desenho são construídas,
 - Classes mais refinadas e adaptadas ao ambiente de implementação

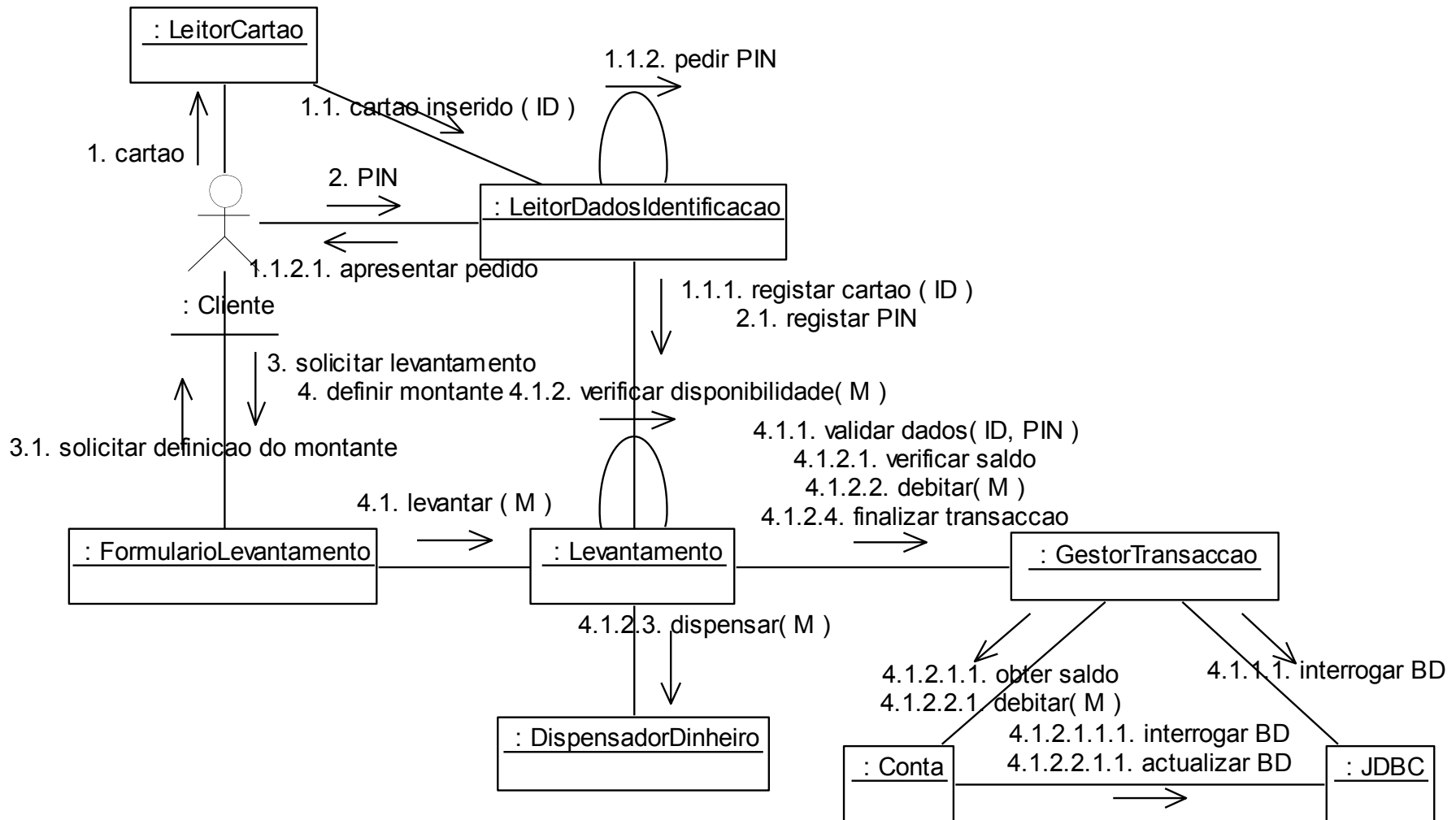
Das Classes do Modelo de Análise às de Desenho



Modelo de Desenho – Diagrama de Sequência



Modelo de Desenho – Diagrama de Colaboração



... do estereótipo *boundary* às Classes de Desenho

- Como tudo no Modelo de Desenho esta passagem
 - depende da tecnologia adoptada
- Adoptando por exemplo,
 - *Java*, na interacção pessoa-máquina,
 - podem ser <<applet>> em conjunto com outras classes que representam *controls* (possivelmente *AWT* ou *Swing controls*)
 - *Visual Basic*, na interacção pessoa-máquina,
 - podem ser <<form>>, em conjunto com outras classes que representam *controls* (possivelmente *ActiveX controls*)
- As ferramentas visuais de construção de interfaces pessoa-máquina
 - tornam implícita esta criação das Classes de Desenho
- Qualquer protótipo de interface pessoa-máquina que já exista
 - é um valioso *input* a esta actividade

... do estereótipo *entity* às Classes de Desenho

- Esta passagem implica normalmente a adopção de
 - uma tecnologia específica de base de dados
- Adoptando por exemplo,
 - *modelo relacional* – *SGBD R*, na suporte à persistência,
 - criar classes de desenho que representam tabelas,
 - especial ênfase – estratégias de conversão do modelo da orientação por objectos para modelo relacional
 - ... estas decisões podem ser apoiadas por ferramentas
 - *modelo orientado a objectos* – *SGBD OO*, na suporte à persistência,
 - suporte directo à persistência das classes de análise
- Nesta passagem pode ser necessário incluir,
 - especialistas na área da modelação e administração de bases de dados

... do estereótipo *control* às Classes de Desenho

- Esta passagem "é delicada" – estas classes encapsulam,
 - a coordenação das actividades dos restantes objectos
 - as sequências de acções
 - ... por vezes a própria lógica de negócio (*business logic*)
- É necessário considerar os seguintes características,
 - *distribuição* – se as sequências de acções têm que ser distribuídas e geridas entre diversos nós numa rede, pode ser necessário,
 - para cada um dos nós, construir classes de desenho separadas
 - *desempenho* – pode não se justificar ter diferentes classes de desenho para realizar as *control class*
 - a *control class* pode ser realizada pela mesma classe de desenho que realiza uma classe *boundary* e/ou *entity*
 - *transacção* – as *control class* muitas vezes encapsulam mecanismos transaccionais; as correspondentes *design class* devem incorporar
 - a tecnologia de gestão transaccional adoptada

Modelo de Desenho e Subsistemas

- Modelo de Desenho
 - pode conter um grande número de classes
- Em sistemas com centenas ou milhares de classes,
 - concretizar *use cases* apenas com classes não é realizável
 - é impossível compreender um sistema de grande dimensão,
 - sem recorrer a mecanismos de agrupamento (*higher-order grouping*)
 - as classes devem ser agrupadas em subsistemas
- Subsistemas, pode ser identificados de dois modos,
 - *bottom-up* – "empacotando" as classes identificadas
 - *top-down* – antes de se identificarem classes, o sistema é dividido em grandes blocos (subsistemas) e definidas as suas interfaces (implica algum "à vontade" com o problema) – em seguida, os subsistemas são repartidos pelos analistas, e cada um procede à identificação das classes do seu subsistema

Subsistema

- Agrupamento, semanticamente útil, de
 - Classes
 - outros Subsistemas
- Um subsistema
 - fornece um conjunto de interfaces
 - utiliza outro conjunto de interfaces
- *service subsystem* (subsistema serviço, ou simplesmente serviço)
 - designa, na hierarquia de subsistemas, os de mais baixo nível
 - representam uma unidade funcional tratável (um todo)
 - *manageable unit of functionality*
 - usado para modelar grupos de classes onde normalmente
 - as alterações se propagam à maioria do grupo
 - só deve ser possível instalar um *service subsystem* na sua totalidade

Modelo de Desenho – Subsistemas

