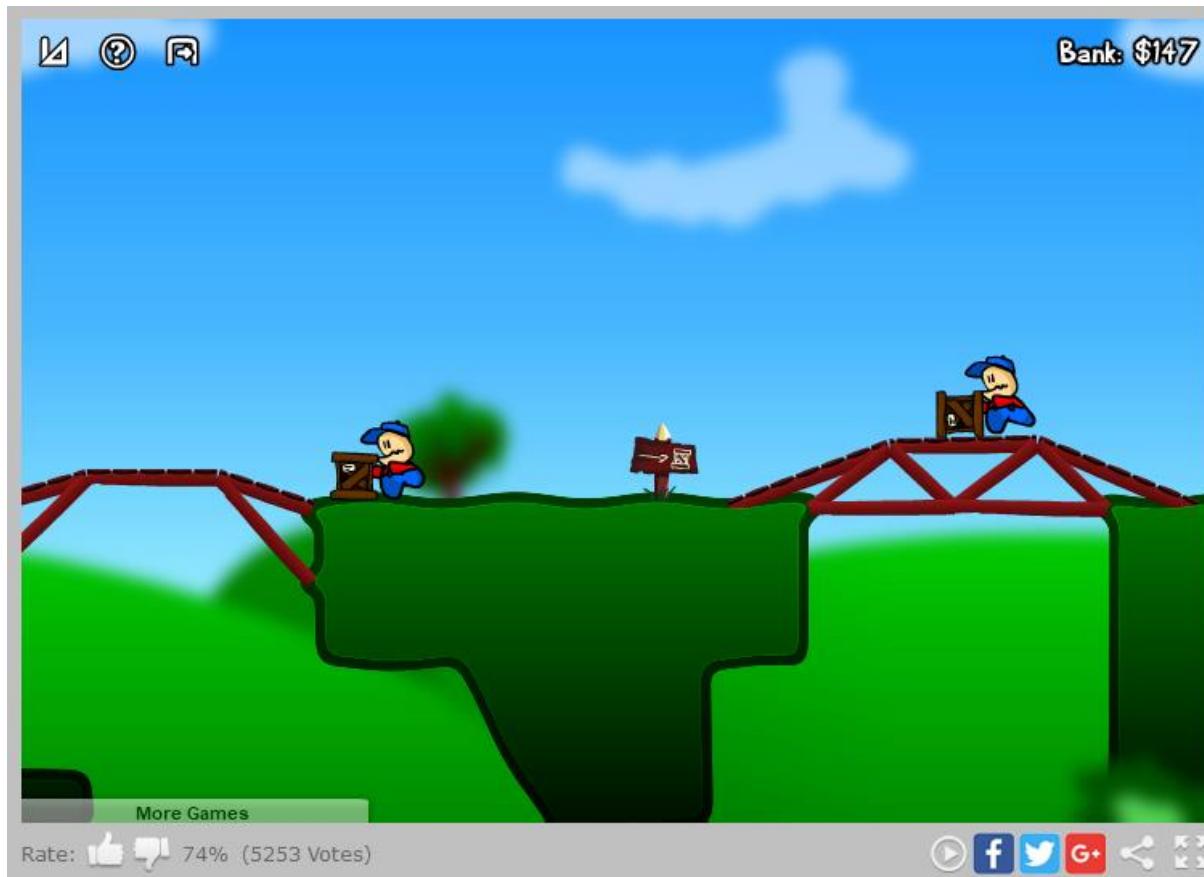
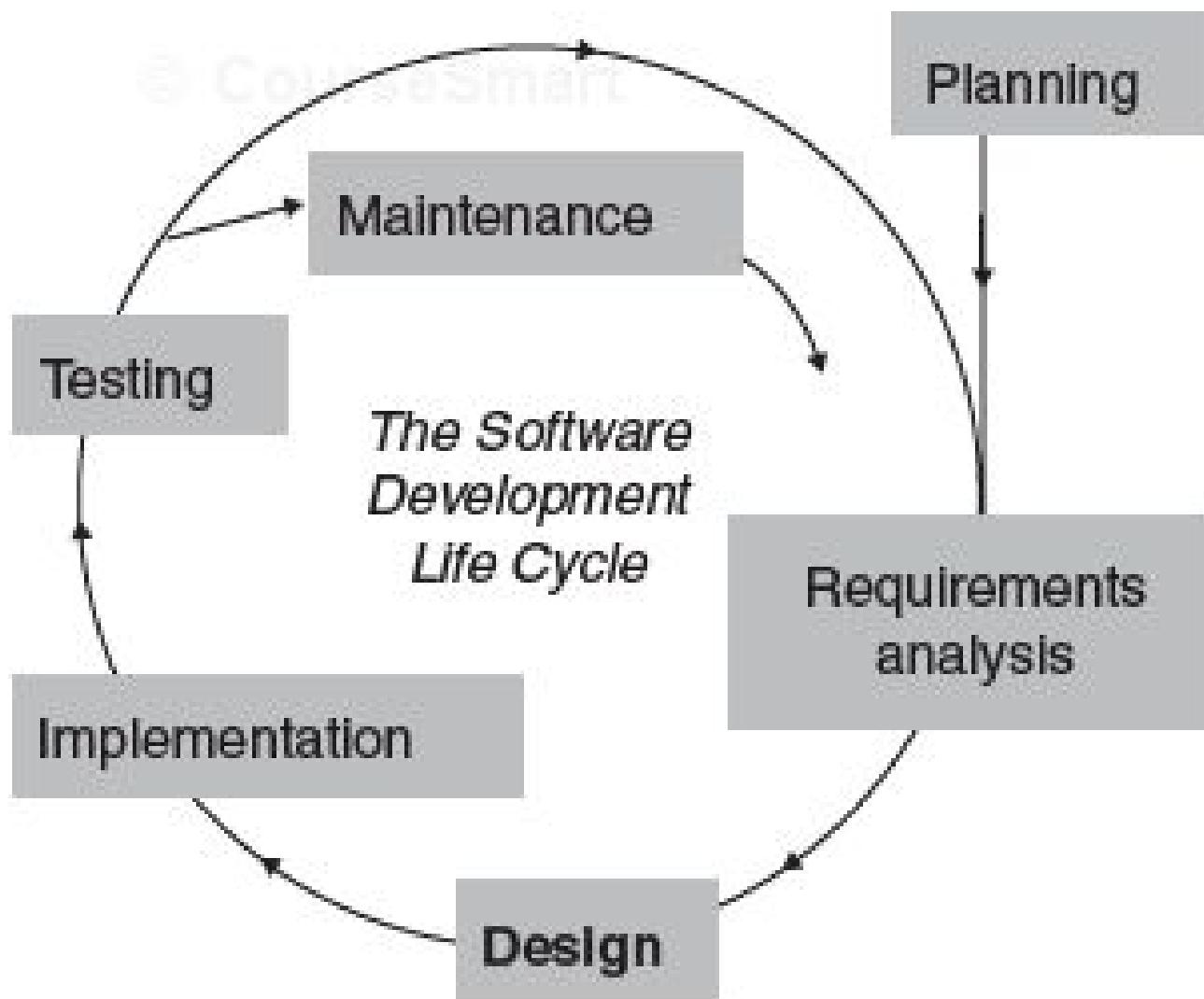


Tvorba informačných systémov

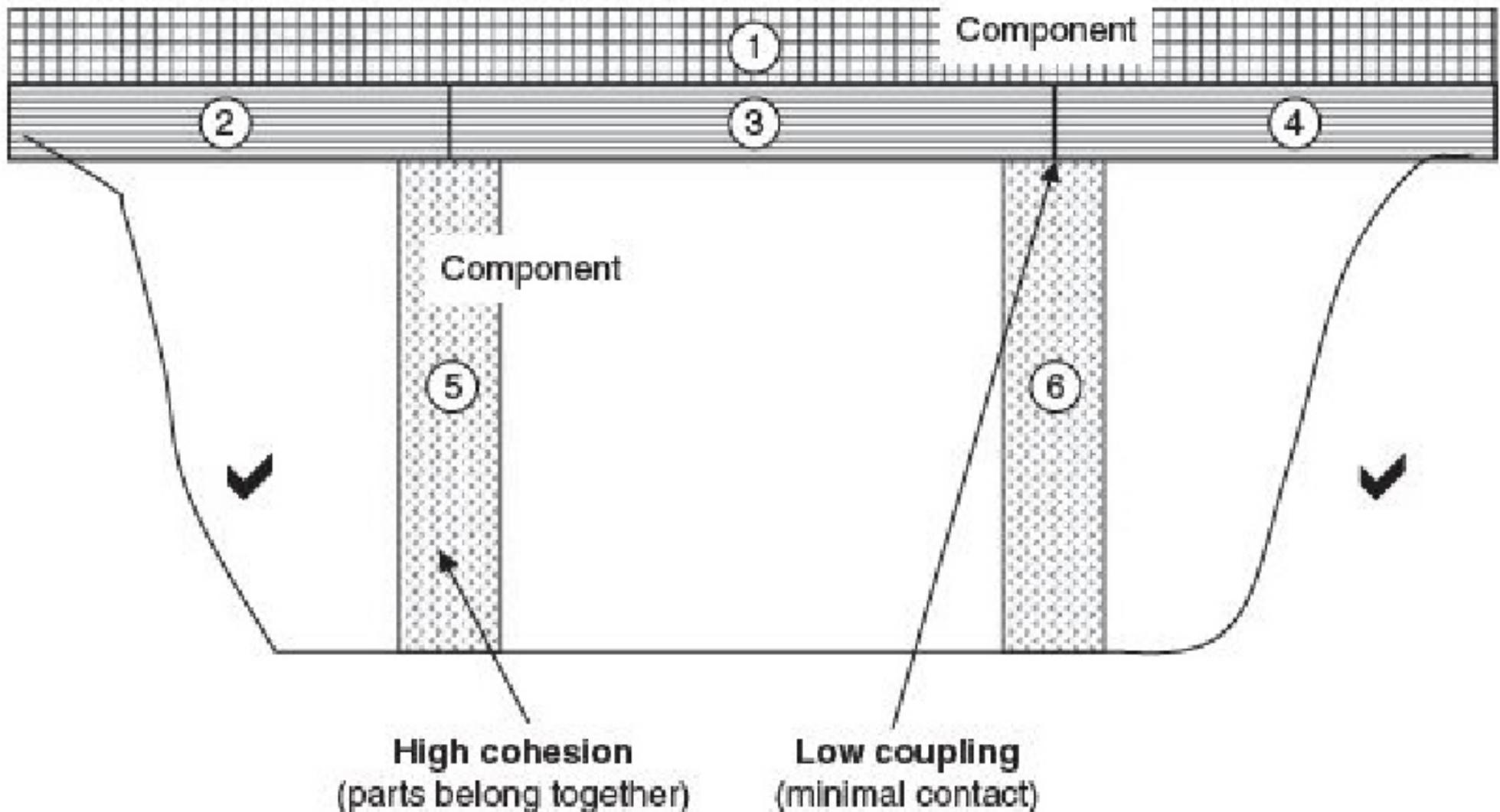
4. prednáška: Návrh IS



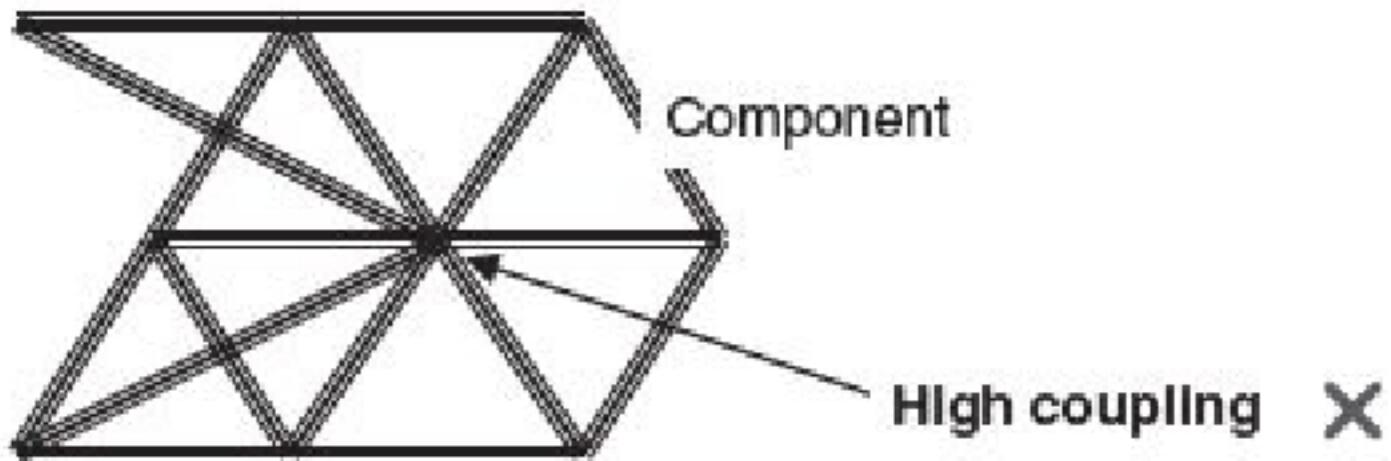


Návrh informačného systému: témy

- Ciele návrhu
- ERD
- DFD
- Princípy OOP
- Objektová normalizácia
- SDD
- Architektonické pohľady
- UML diagramy
- Architektonické štýly
- Návrhové vzory



High cohesion and low coupling—bridge example

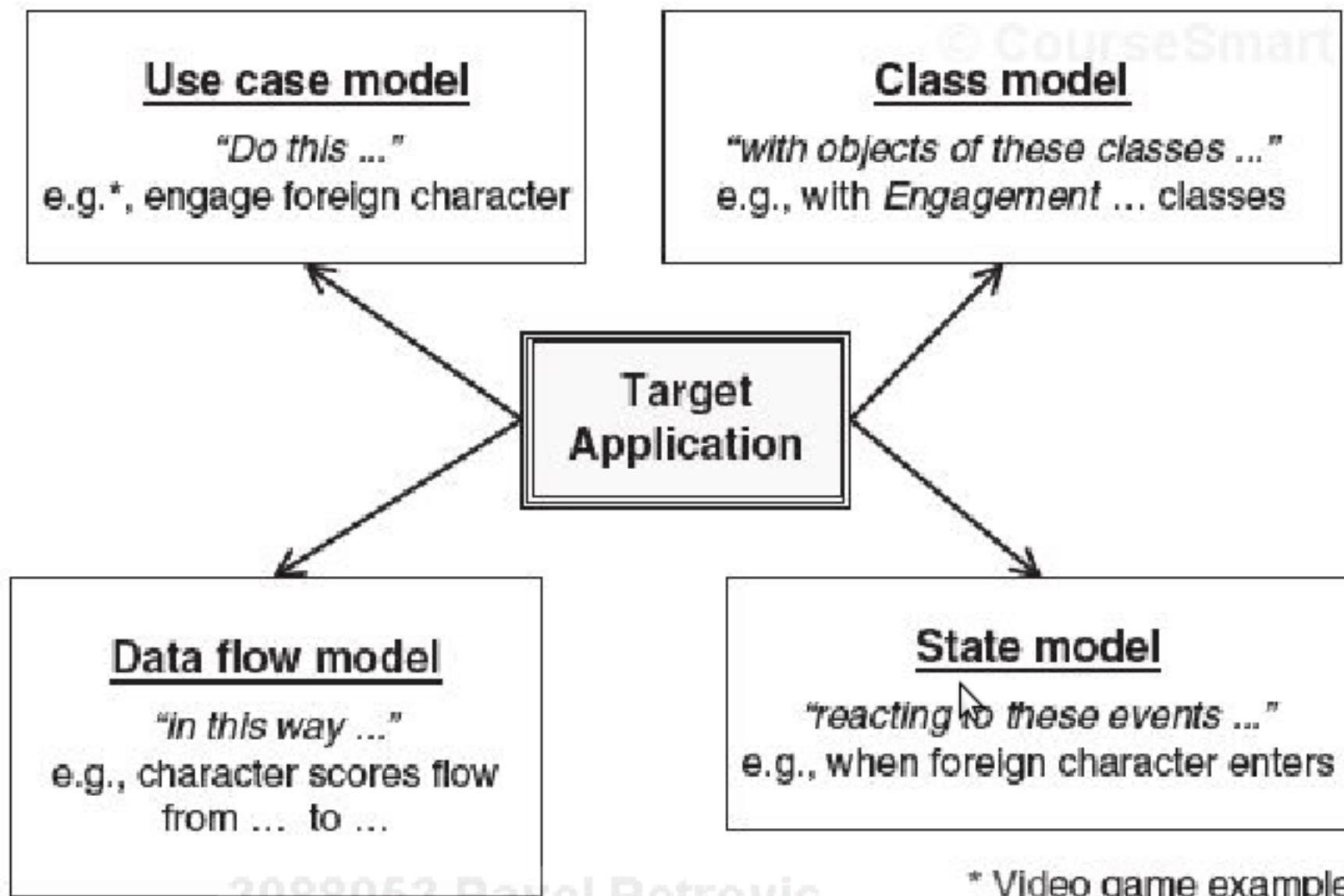


A questionable architecture—high coupling in a truss

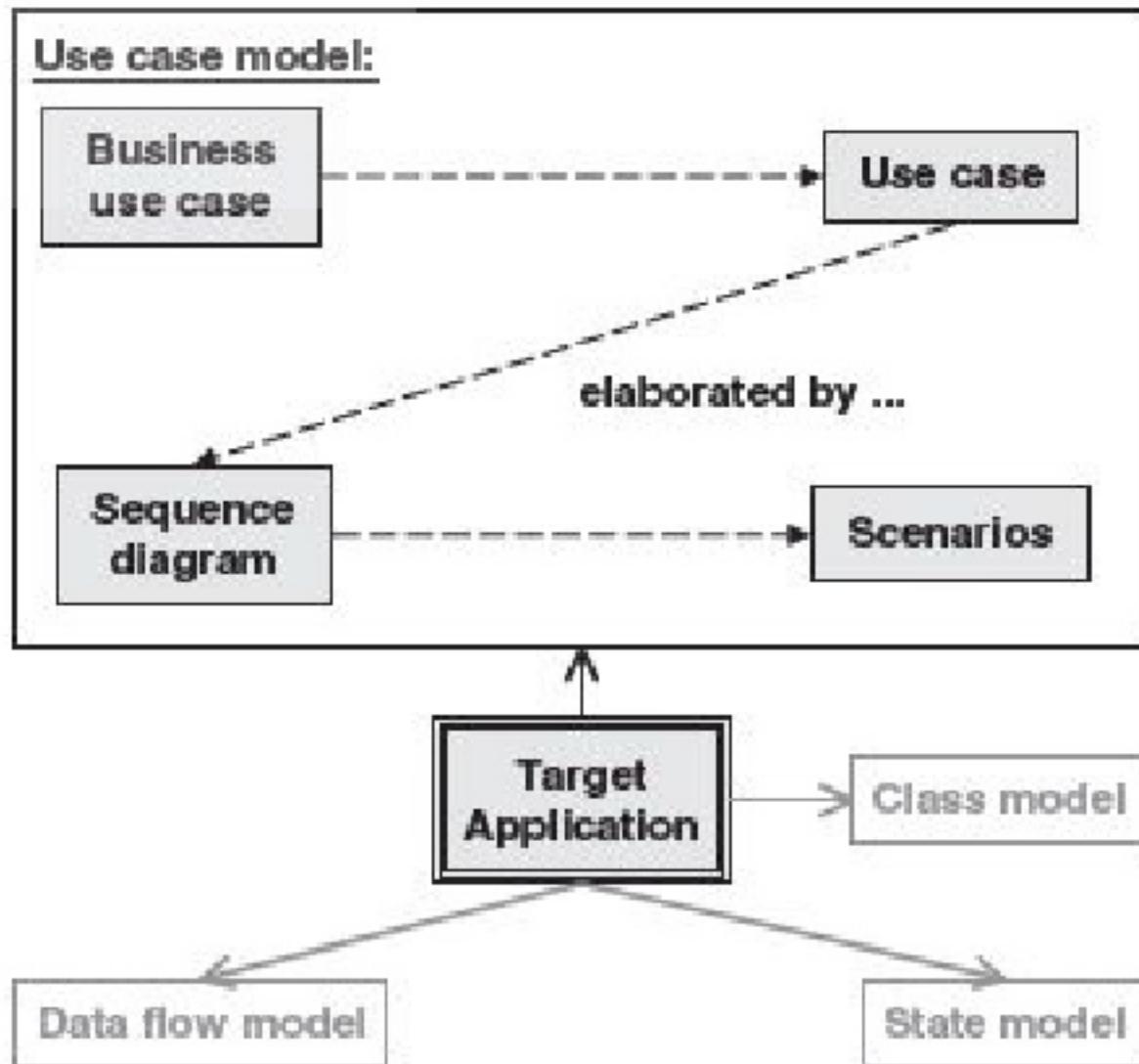
Principal goals of software design:

- *Sufficiency*: handles the requirements
- *Understandability*: can be understood by intended audience
- *Modularity*: divided into well-defined parts
- *Cohesion*: organized so that like-minded elements are grouped together
- *Coupling*: organized to minimize dependence between elements
- *Robustness*: can deal with wide variety of input
- *Flexibility*: can be readily modified to handle changes in requirements
- *Reusability*: can use parts of the design and implementation in other applications
- *Information hiding*: module internals are hidden from others
- *Efficiency*: executes within acceptable time and space limits
- *Reliability*: executes with acceptable failure rate

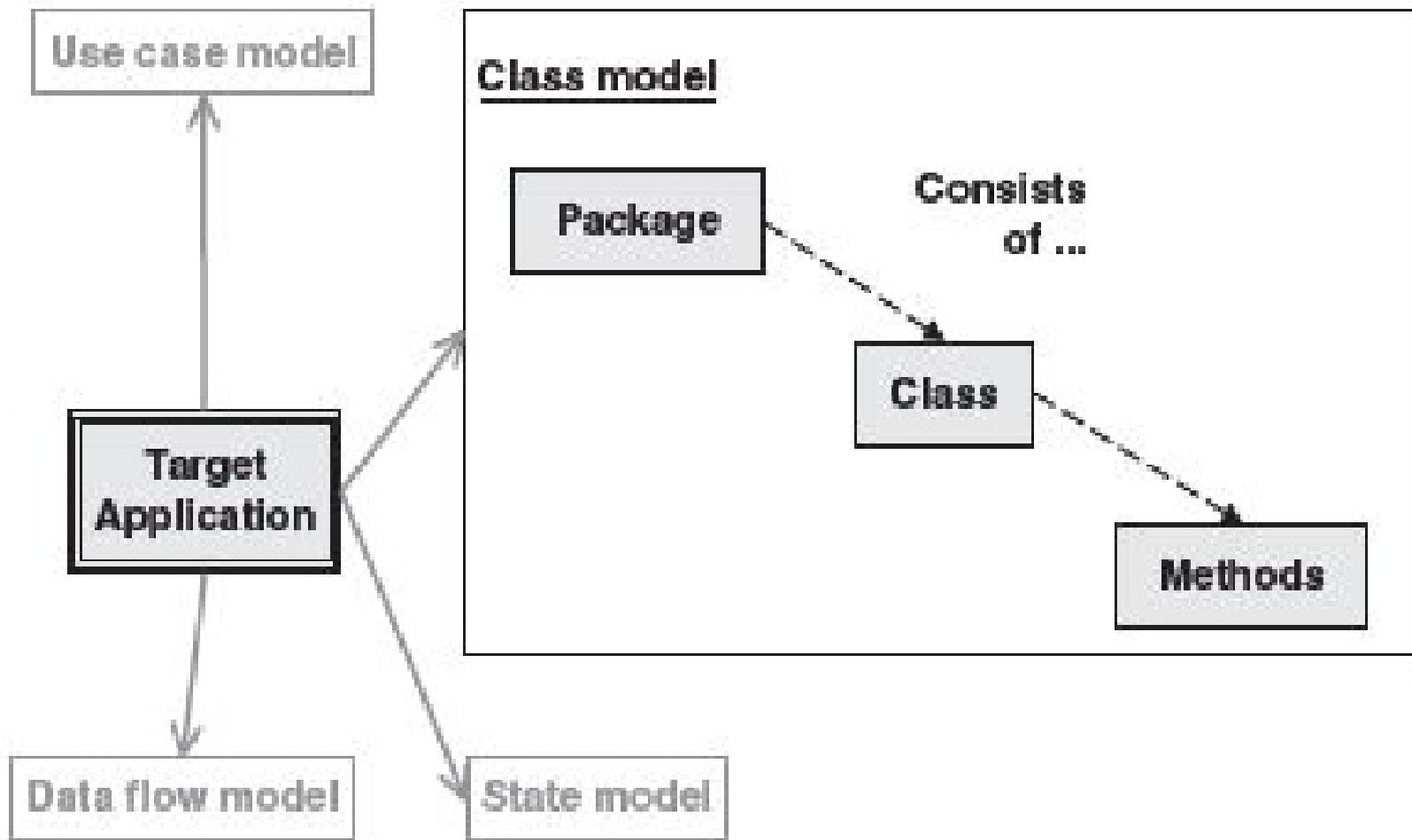
To express requirements, architecture, and detailed design



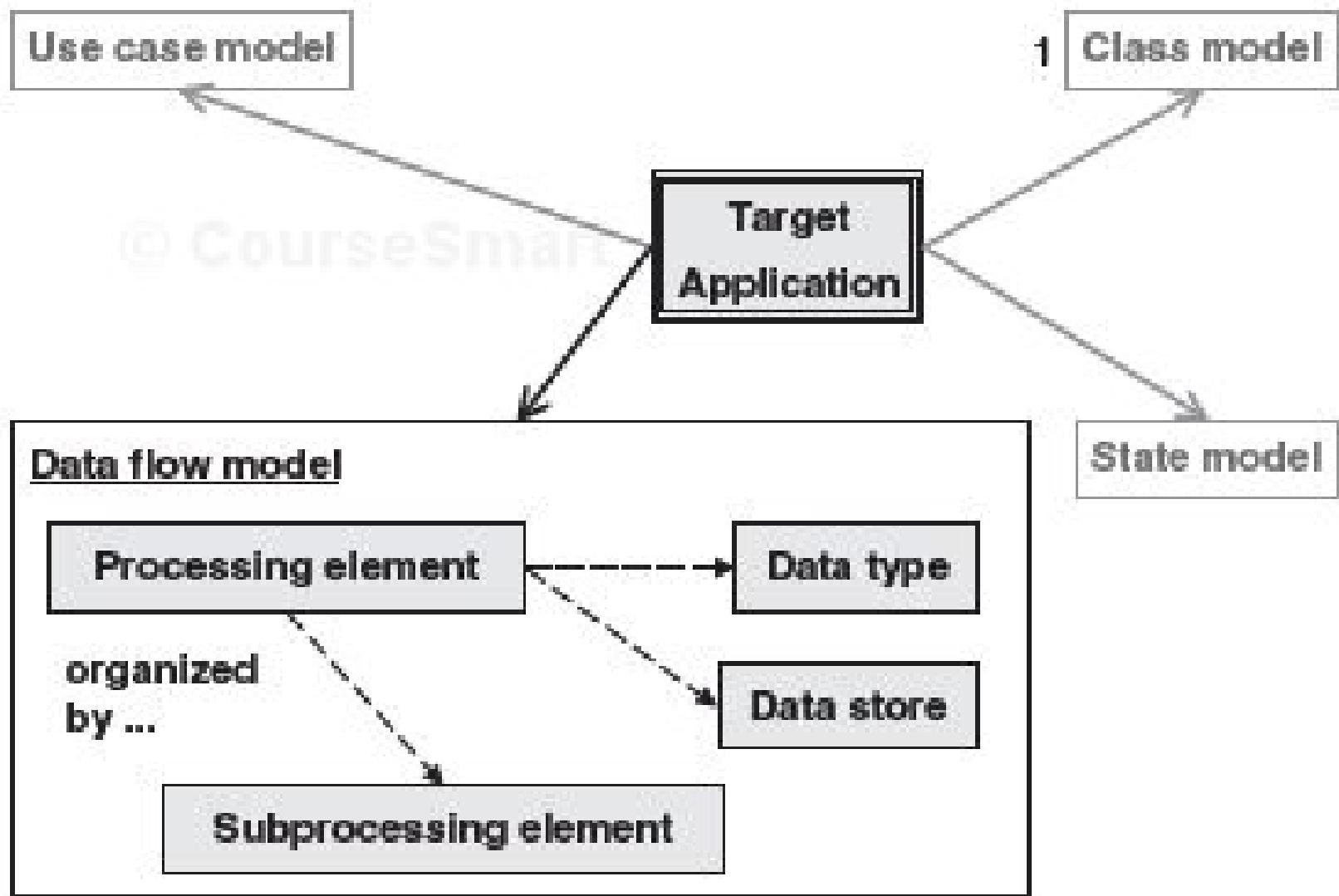
* Video game example



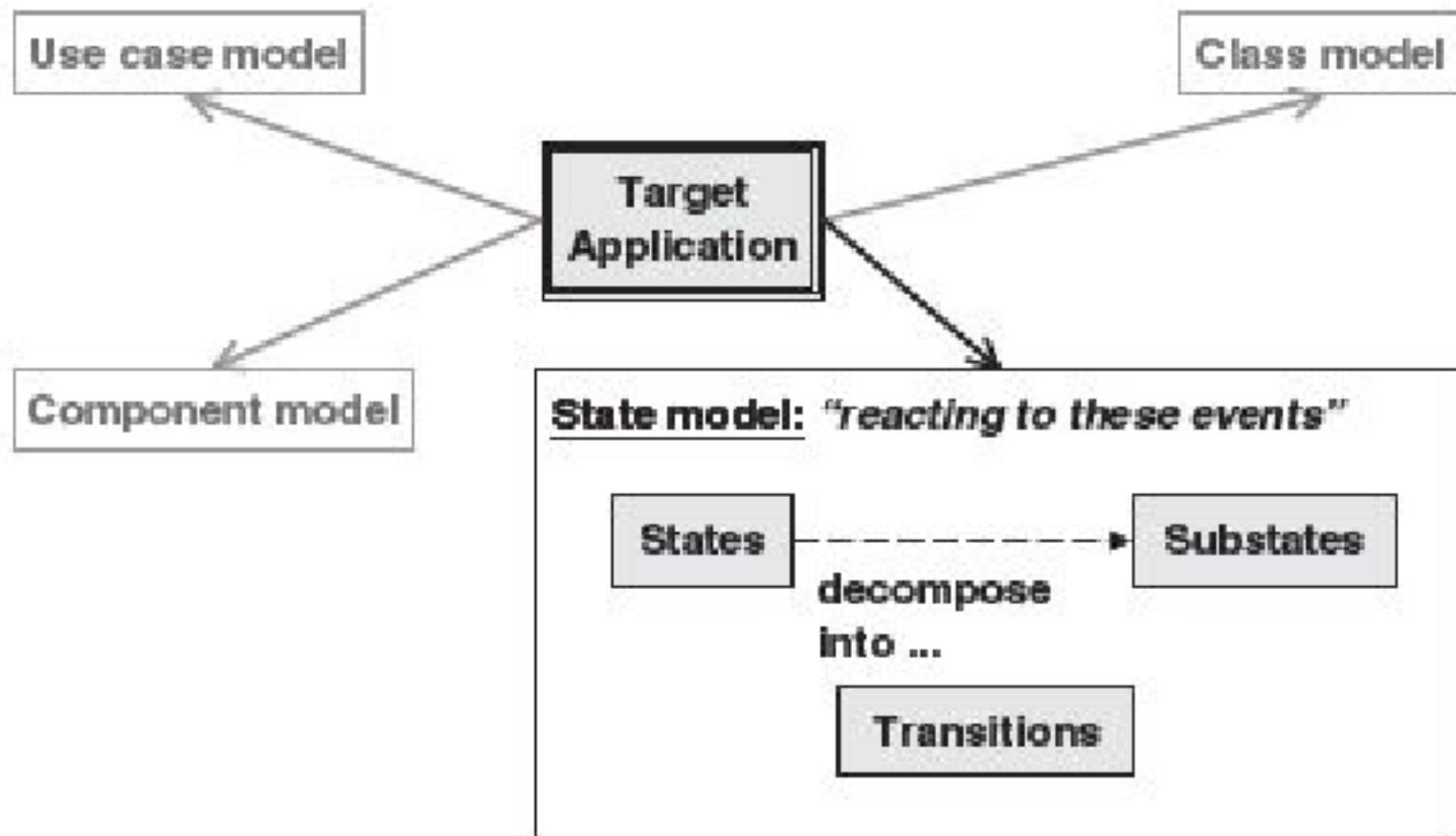
The role of use case models in design



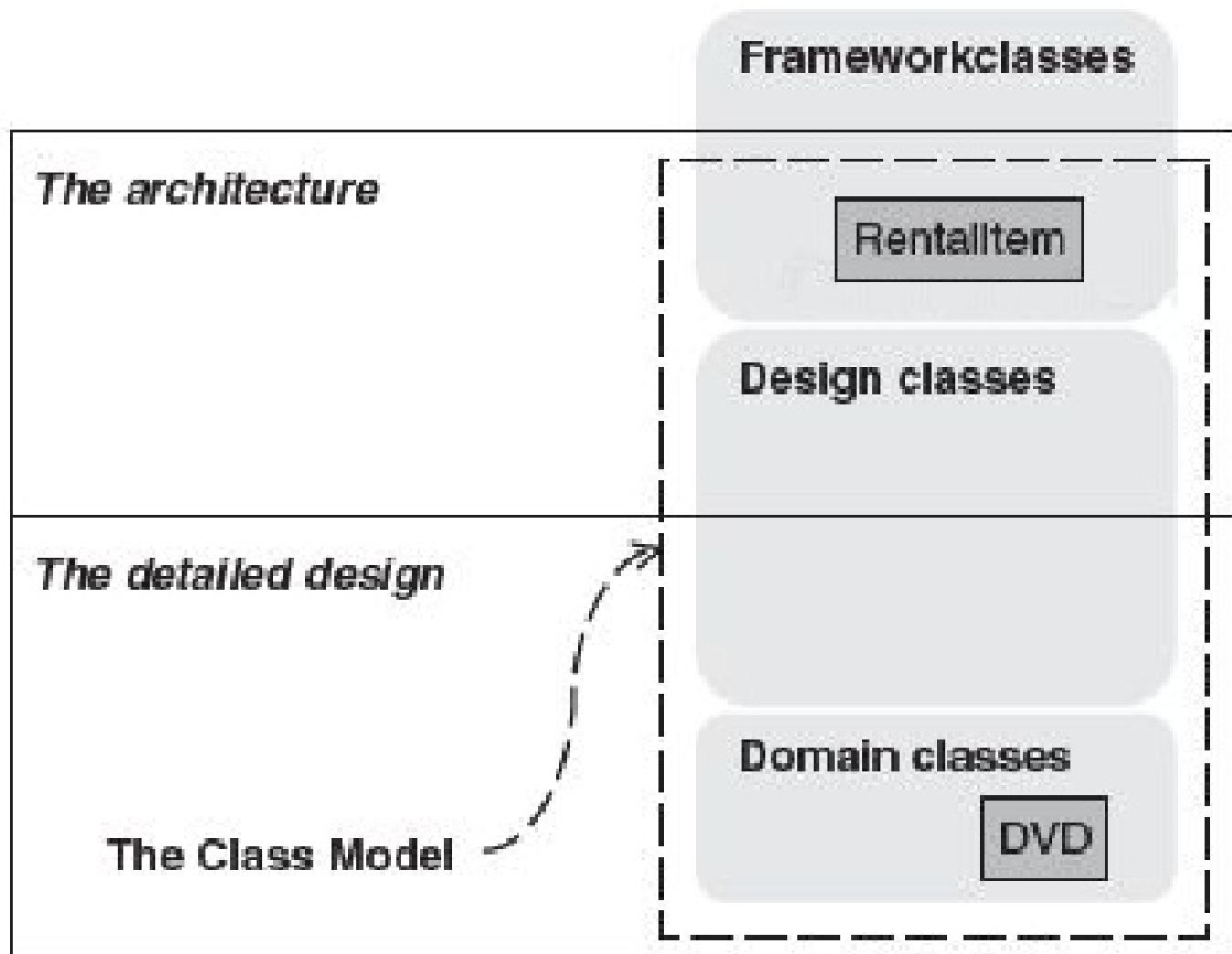
The role of class models in design



The role of component models in design

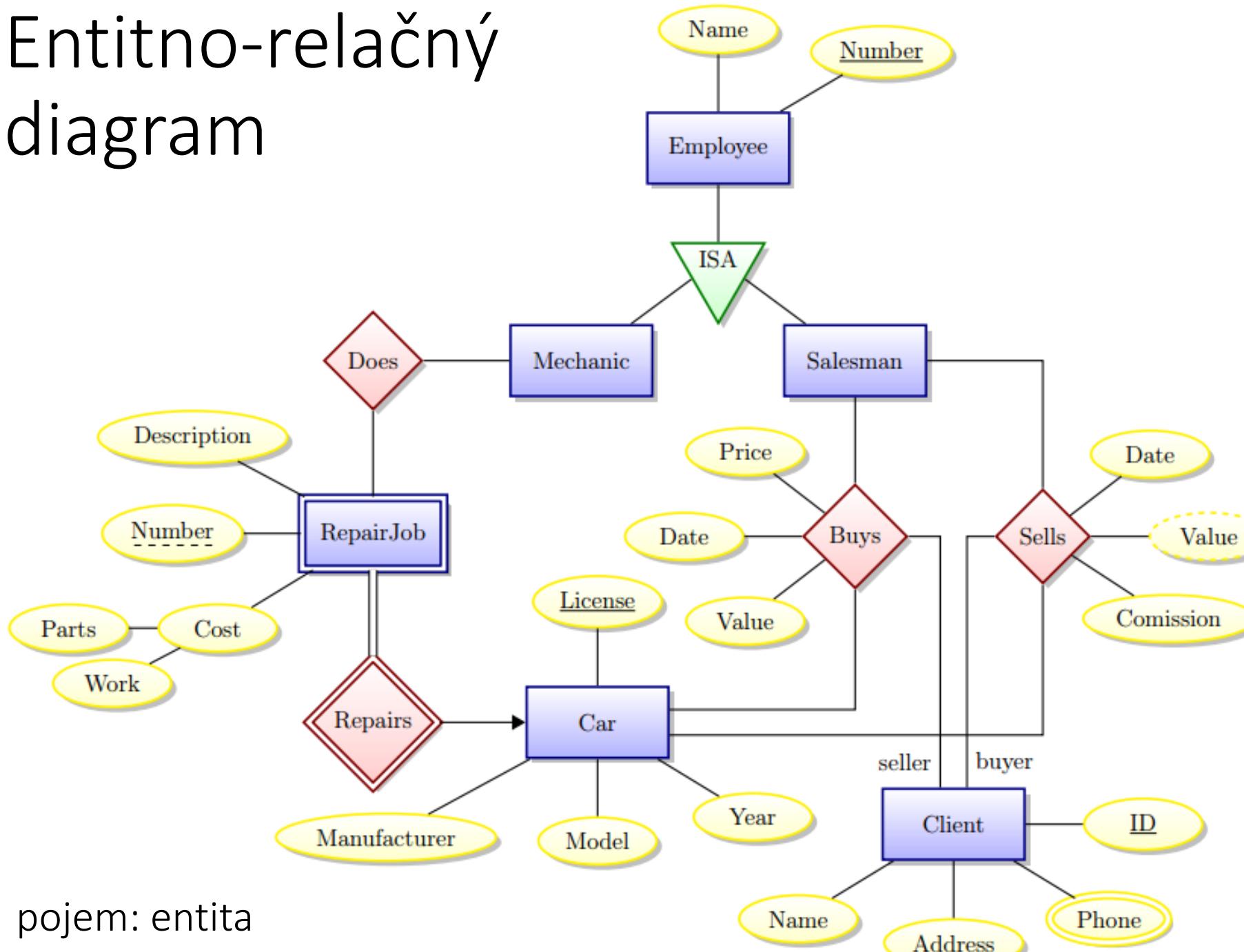


The role of state models in design

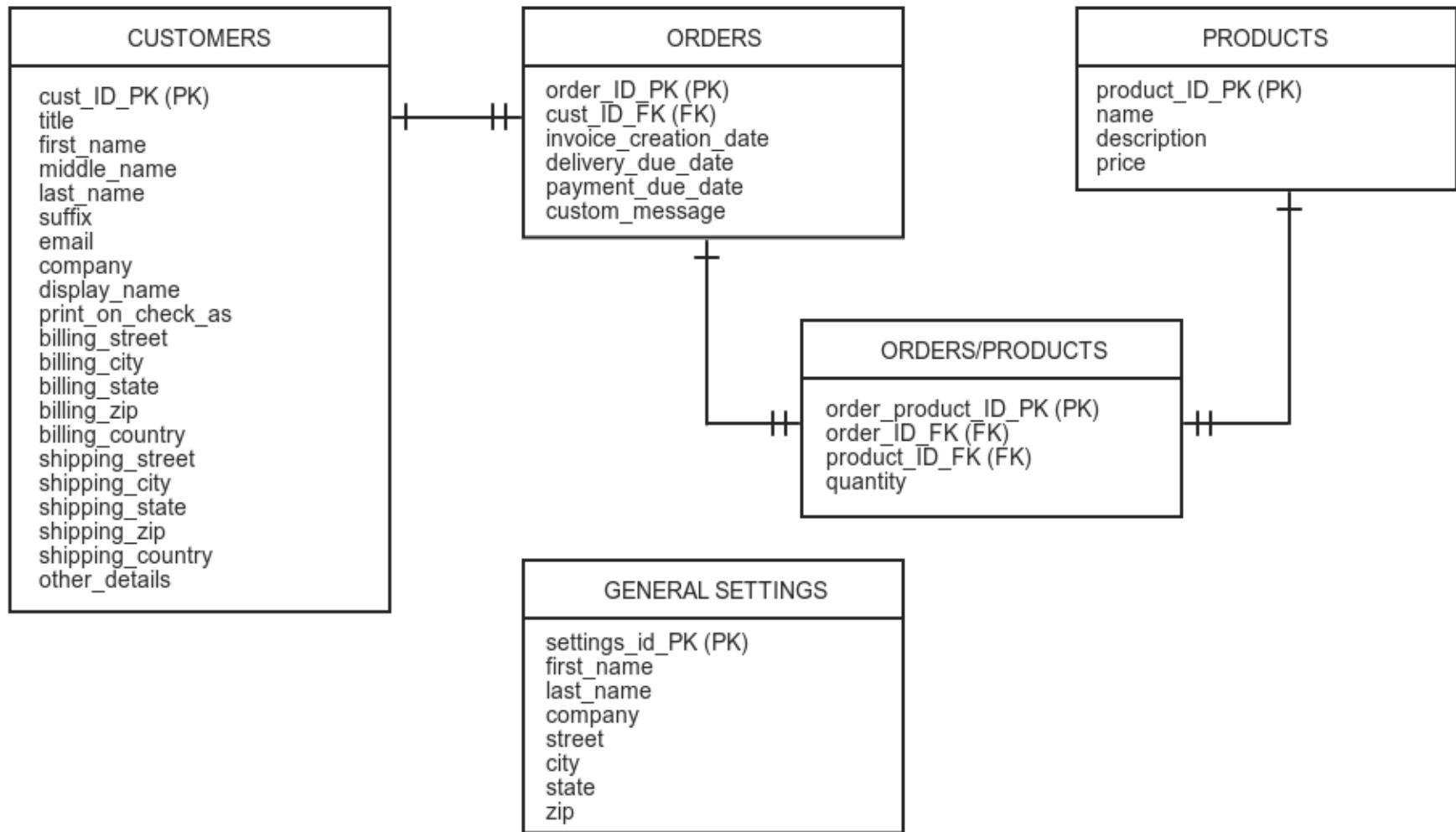


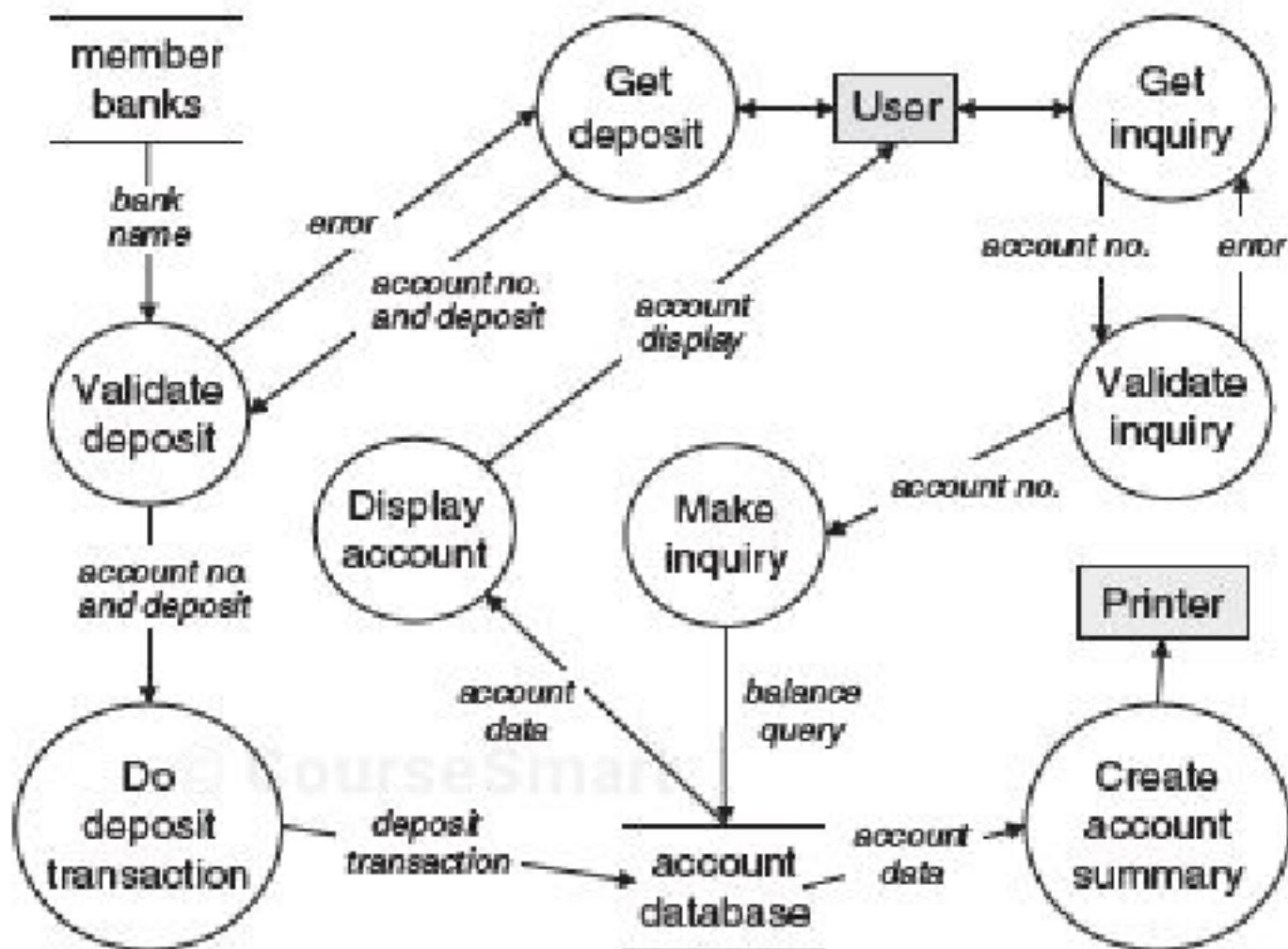
Class model vs. architecture and detailed design

Entitno-relačný diagram



pojem: entita





Partial data flow diagram for an ATM application

Princípy OOP

90 roky 20.storočia

v centre pozornosti sú DÁTA!

operácie nad dátami sú blízko dát

OOP = inheritance + polymorphism + encapsulation

abstrakcia

abstraktný dátový typ (ADT)

ako funguje polymorfizmus?

Kritika: príliš zväzujúce a statické hierarchie tried
nové trendy – nahradzovať objekty a triedy a dedičnosť
inými konceptami a mechanizmami (jazyk GO)

Návrh dátového modelu: objektová normalizácia

1ONF

definice

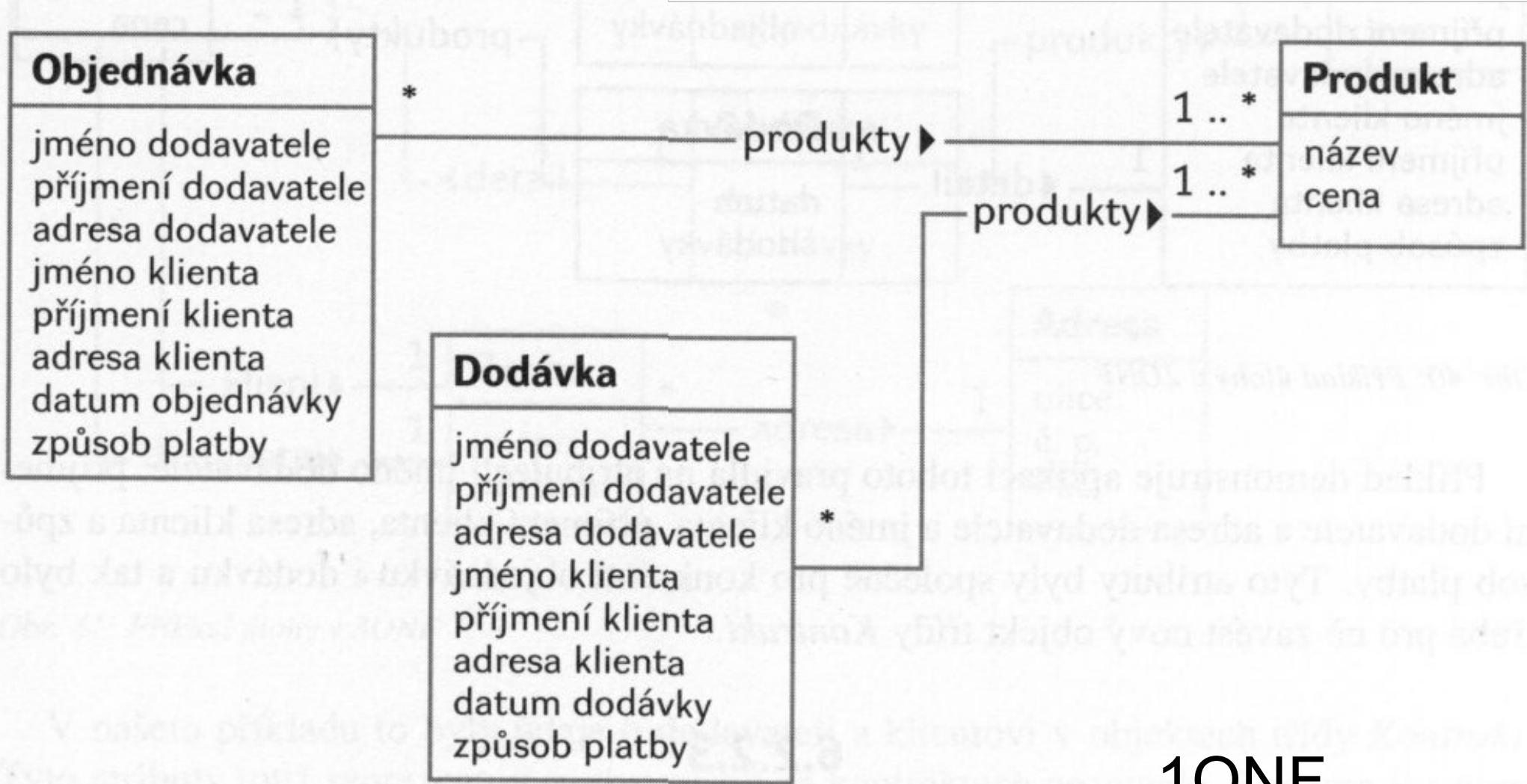
Třída je v první objektové normální formě (1ONF), jestliže její objekty neobsahují skupinu opakujících se atributů. Takové atributy je třeba vyčlenit do objektů nové třídy a skupinu opakujících se atributů nahradit jednou vazbou na kolekci objektů této nové třídy. Schéma je v 1ONF, jestliže všechny třídy objektů v něm jsou v 1ONF.

Návrh dátového modelu: objektová normalizácia

Objednávka	Dodávka
jméno dodavatele	jméno dodavatele
příjmení dodavatele	příjmení dodavatele
adresa dodavatele	adresa dodavatele
jméno klienta	jméno klienta
příjmení klienta	příjmení klienta
adresa klienta	adresa klienta
datum objednávky	datum objednávky
způsob platby	způsob platby
název prvního produktu	název prvního produktu
cena prvního produktu	cena prvního produktu
název druhého produktu	název druhého produktu
cena druhého produktu	cena druhého produktu
název třetího produktu	název třetího produktu
cena třetího produktu	cena třetího produktu

pôvodný model...

Návrh dátového modelu: objektová normalizácia



1ONF

Návrh dátového modelu: objektová normalizácia

2ONF

definice

Třída je v druhé objektové normální formě (2ONF), jestliže její objekty neobsahují atribut nebo skupinu atributů, které by byly sdílené s nějakým jiným objektem. Sdílené atributy je třeba vyčlenit do objektu nové třídy a ve všech objektech, kde se vyskytovaly, nahradit vazbou na tento objekt nové třídy. Schéma je v 2ONF jestliže všechny třídy objektů v něm jsou v 2ONF.

Návrh dátového modelu: objektová normalizácia



2ONF

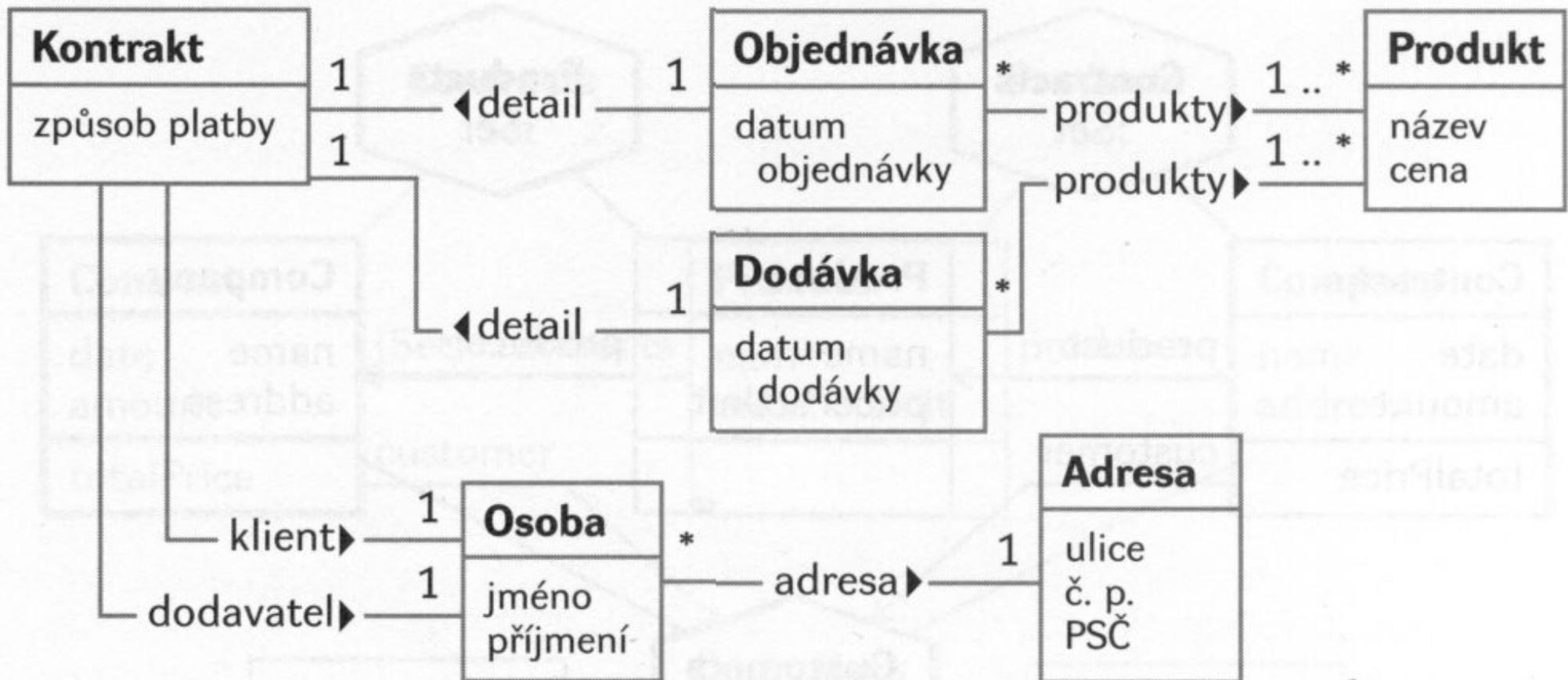
Návrh dátového modelu: objektová normalizácia

3ONF

definice

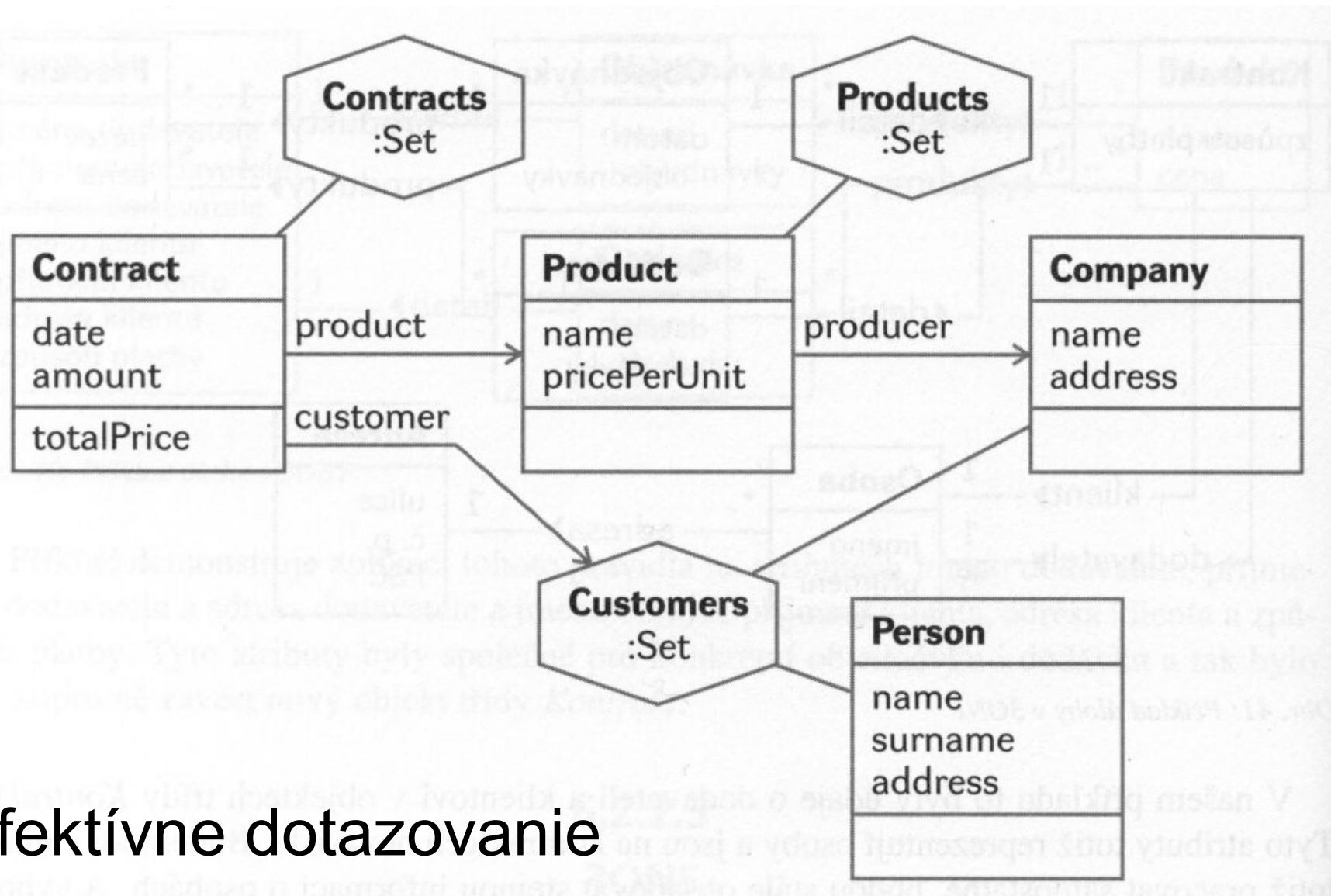
Třída je ve třetí objektové normální formě, jestliže její objekty neobsahují atribut nebo skupinu atributů, které mají samostatný význam nezávislý na objektu, ve kterém jsou obsaženy. Pokud takové atributy existují, je třeba je vyčlenit do objektu nové třídy, a v objektu, kde byly obsaženy, nahradit vazbou na tento objekt nové třídy. Schéma je v 3ONF, jestliže všechny třídy objektů v něm jsou v 3ONF.

Návrh dátového modelu: objektová normalizácia



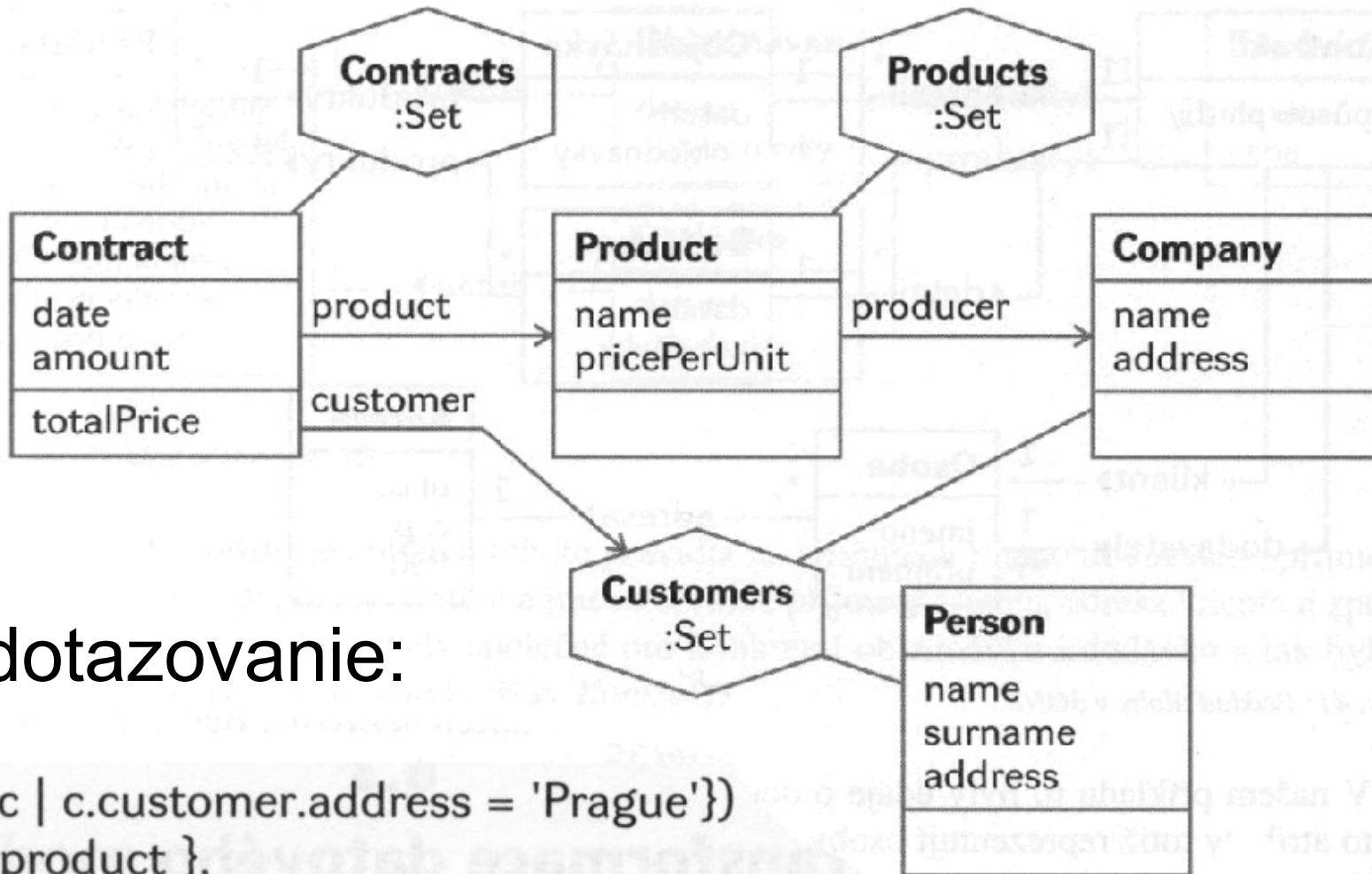
3ONF

Návrh dátového modelu: objektová transformácia



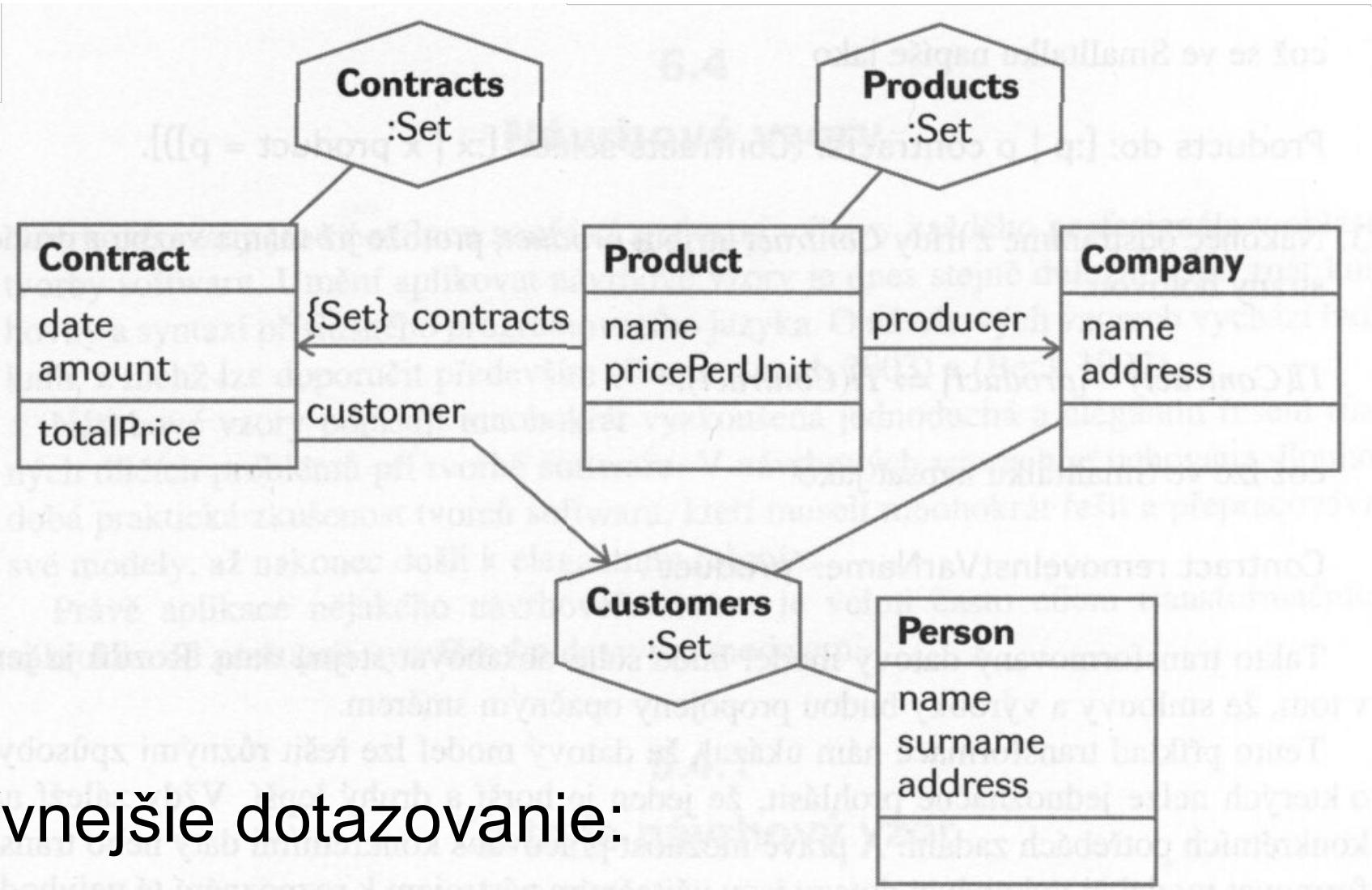
neefektívne dotazovanie

Návrh dátového modelu: objektová transformácia



Jak je tedy vidět, výrobky objednávané zákazníky z Prahy získáme tak, že nejprve vybereme smlouvy zákazníků z Prahy a potom z výsledku sesbíráme výrobky.

Návrh dátového modelu: objektová transformácia



efektívnejšie dotazovanie:

```
Products select: {p | p.contracts.*.customer.address = 'Prague'}
```

Návrh dátového modelu: refaktORIZÁcia

Príklady:

1. Přesun metody do nadtíedy. Protože se metoda z nadtíedy dědí, zůstává funkčnost instancí beze změny.
2. Přesun deklarace datové položky do nadtíedy. Protože se deklarace z nadtíedy dědí, zůstává funkčnost instancí beze změny.
3. Rozdělení kódu jedné metody na dvě metody, kde kód jedné metody obsahuje volání druhé metody.
4. Operace inverzní k operacím 1., 2. a 3.
5. Přejmenování jména třídy nebo metody nebo datové složky včetně přejmenování všech volání této třídy, metody nebo datové složky.
6. Vykonání transformace podle pravidla nějaké normální formy.

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, acronyms,
and abbreviations

2. References

3. Decomposition description

3.1 Module decomposition

 3.1.1 Module 1 description

 3.1.1 Module 2 description

3.2 Concurrent process
decomposition

 3.2.1 Process 1 description

 3.2.2 Process 2 description

3.3 Data decomposition

 3.3.1 Data entry 1 description

 3.3.2 Data entry 2 description

Architecture

4. Dependency description

4.1 Intermodule dependencies

4.2 Interprocess dependencies

4.3 Data dependencies

5. Interface description

5.1 Module interface

 5.1.1 Module 1 description

 5.1.2 Module 2 description

5.2 Process interface

 5.2.1 Process 1 description

 5.2.2 Process 2 description

6. Detailed design

6.1 Module detailed design

 6.1.1 Module 1 detail

 6.2.2 Module 2 detail

6.2 Data detailed design

 6.2.1 Data entity 1 detail

 6.2.2 Data entity 2 detail

UML in a nutshell...

- Množstvo spôsobov, ako tímu vývojárov vysvetliť architektúru, UML je priemyselný štandard, ktorý by informatici mali vedieť použiť
- Modelovací jazyk, model = zjednodušená skutočnosť
- UML:
 - stavebné prvky,
 - pravidlá na ich kombinovanie
 - bežné mechanizmy

Stavebné prvky (building blocks)

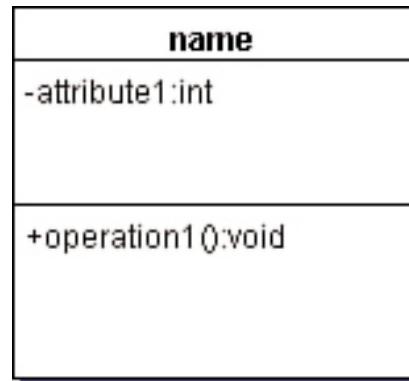
- Elements
- Relationships
- Diagrams

Elements:

- Structural (static parts)
- Behavioral
- Grouping
- Annotational

Structural Elements

Class



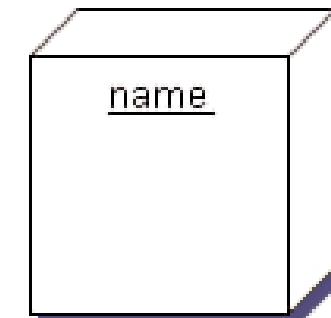
Interface



Component



Node



(conforms to and provides realization of a set of interfaces)

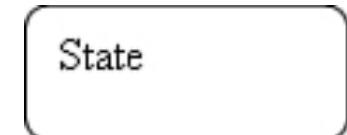
(computational resource existing at runtime, with some memory and CPU)

Behavioral Elements

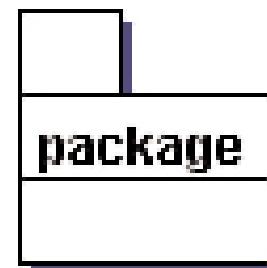
Interaction



State



Grouping Element

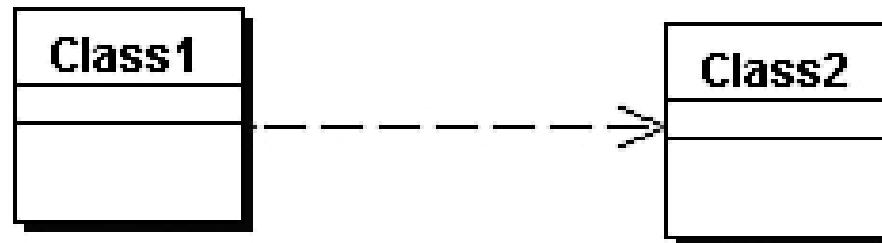


Annotational Element



Relationships

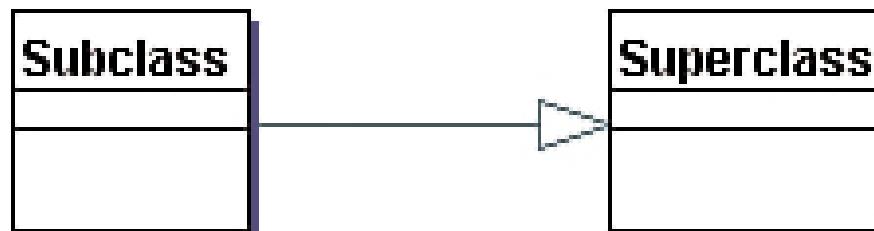
Dependency



Aggregation



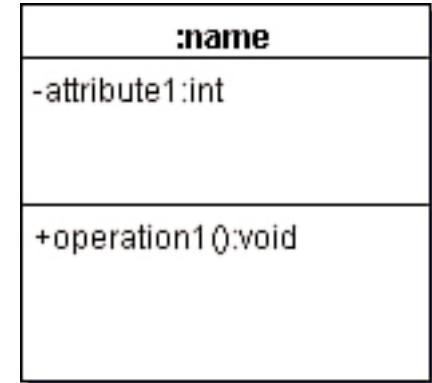
Generalization



Common Mechanisms

Specifications – textual description

Adornments – public(+), private(-),
protected(#)



Common divisions - notation about instances:

instance

:class_of_anonymous_instance

a:circle

Extensibility – UML is open

- Stereotypes
- Tagged values
- Constraints

Diagrams

Structure diagrams

class, component, deployment, package diagrams

Behavior diagrams

activity, statechart, use-case, communication
diagrams

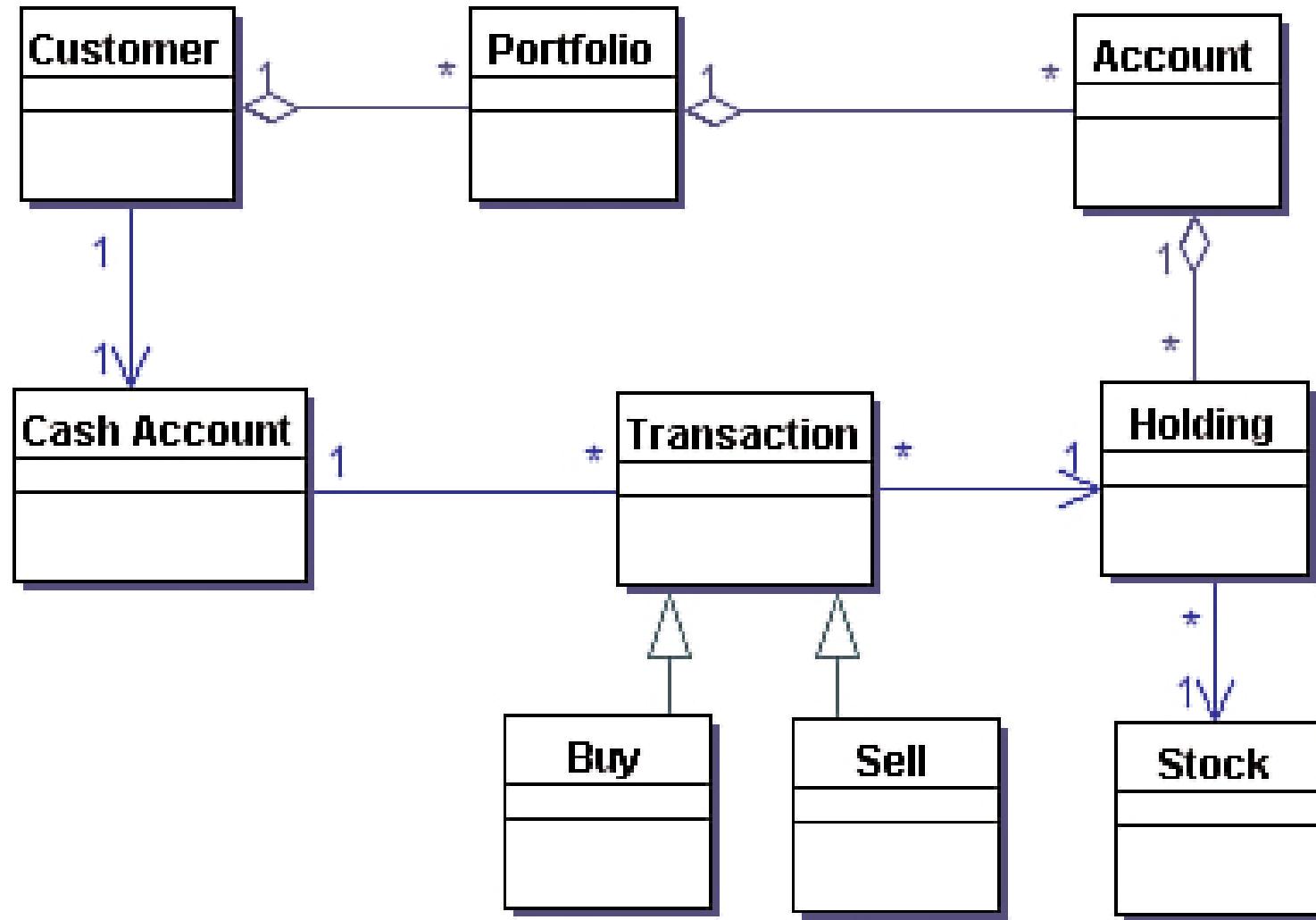
Interaction diagrams

sequence diagrams, collaboration diagrams

Class Diagram

Shows set of classes, interfaces, and collaborations and their relationships
static design view of a system

Class Diagram



Component Diagram

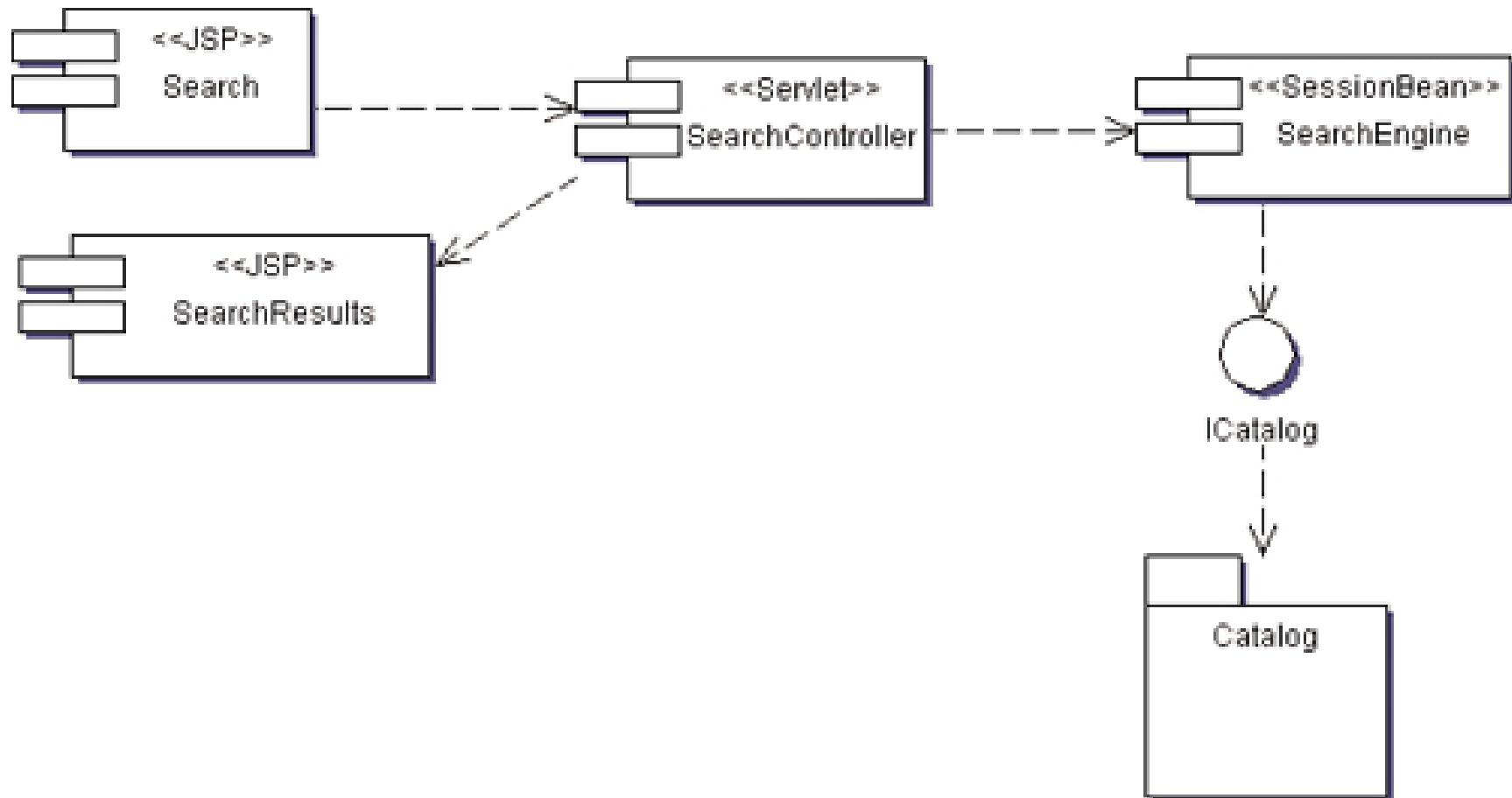
Organizations and dependencies among a set of components

Static implementation view of a system

Component diagram maps to classes, interfaces or collaborations

communicates the overall pattern for the system to be followed by developers (e.g. MVC or DAO)

Component Diagram



Deployment Diagram

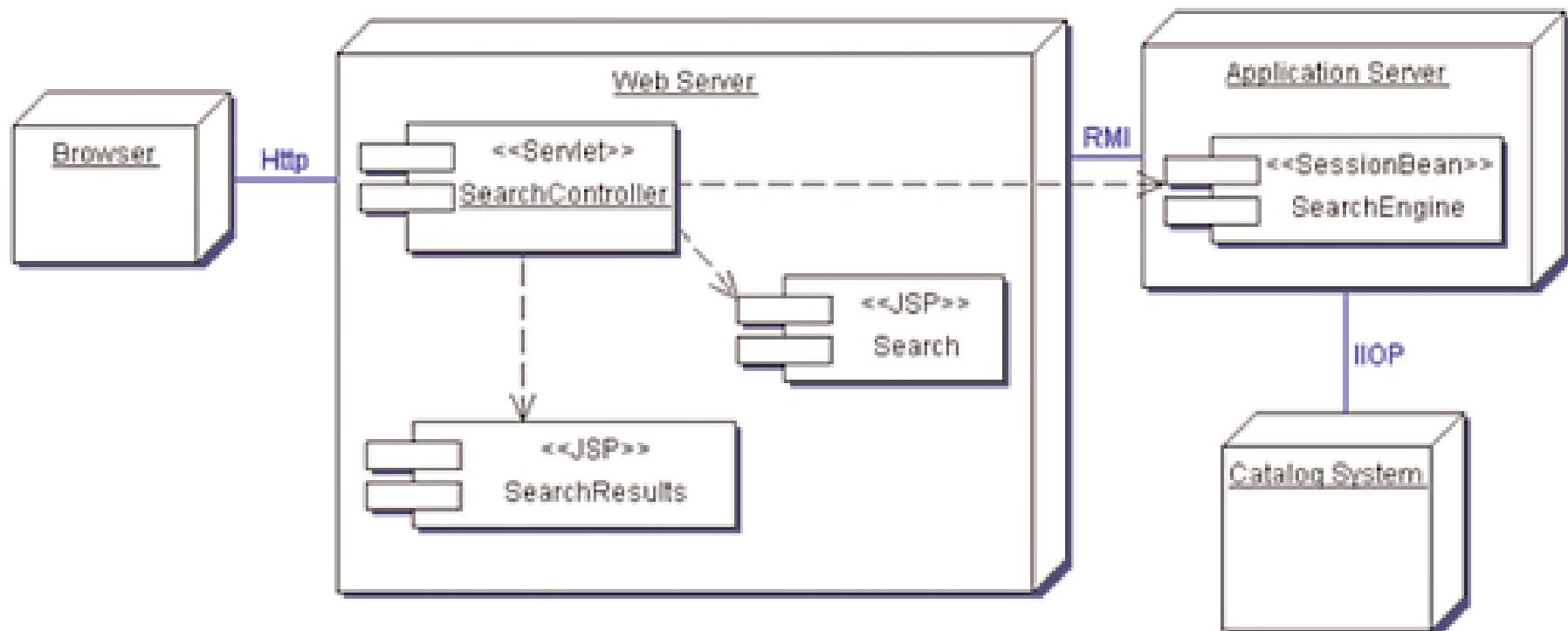
Configuration of run-time processing nodes and the components that live on these nodes

Static deployment view of an architecture

Nodes typically enclose one or more components

Helps developers to understand the boundaries of the system components and what protocol they will use to communicate with dependent components

Deployment Diagram

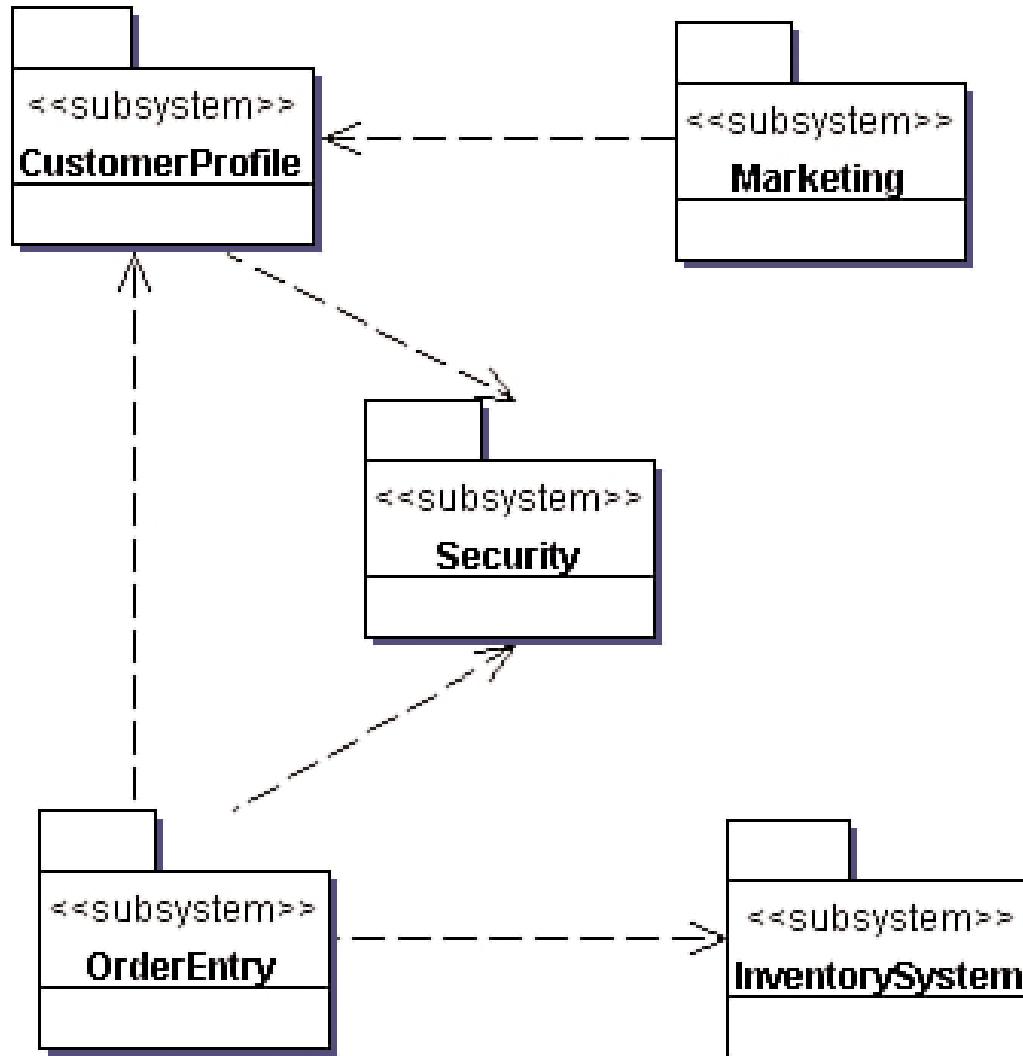


Package Diagram

Special kind of class diagram

Organization of system in groups

Package Diagram



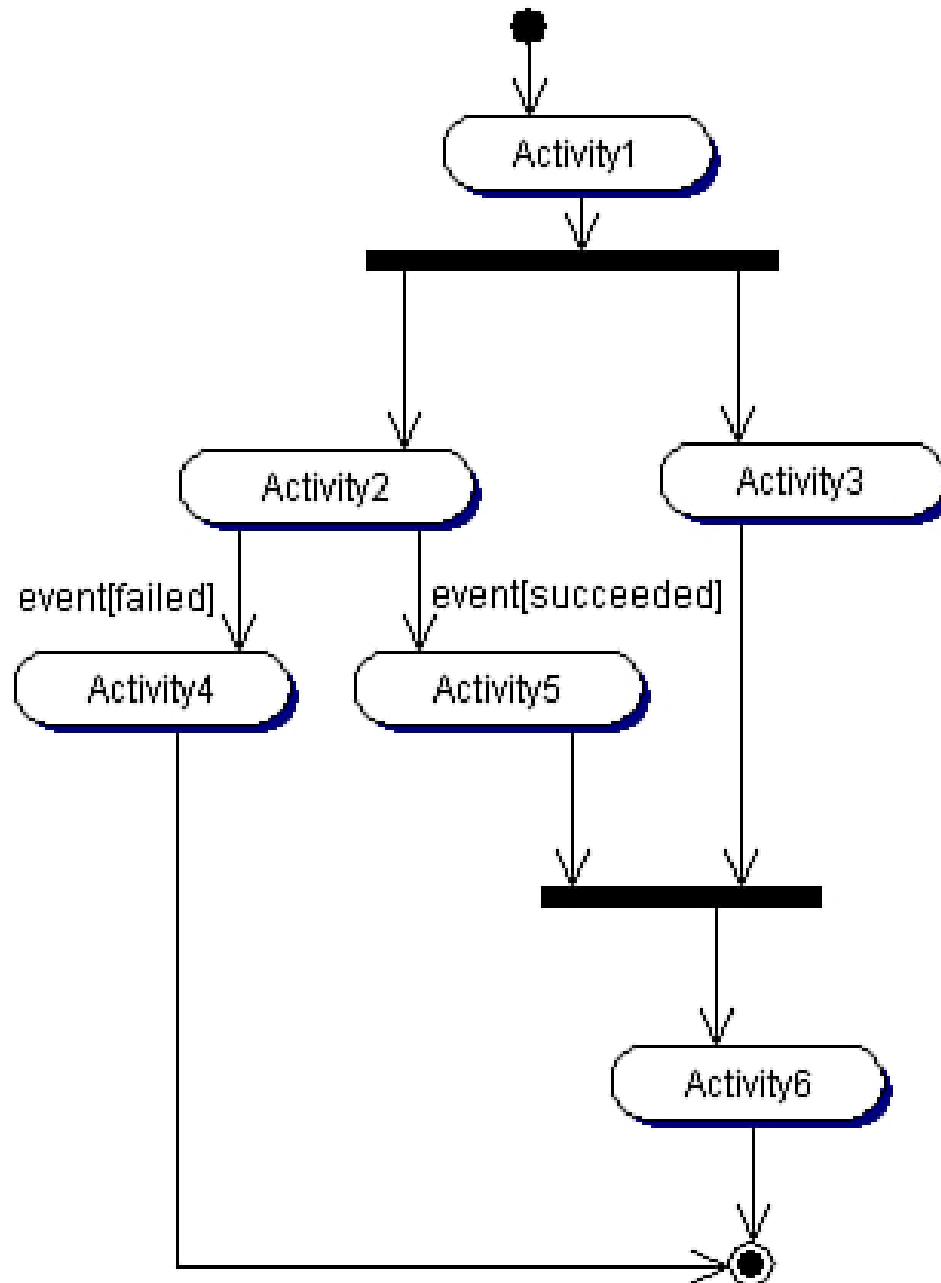
Activity Diagram

Special kind of statechart showing flow from activity to activity within a system

Dynamic view of the system

Modelling system function and emphasizing the flow of control among objects

Activity Diagram



Statechart Diagram

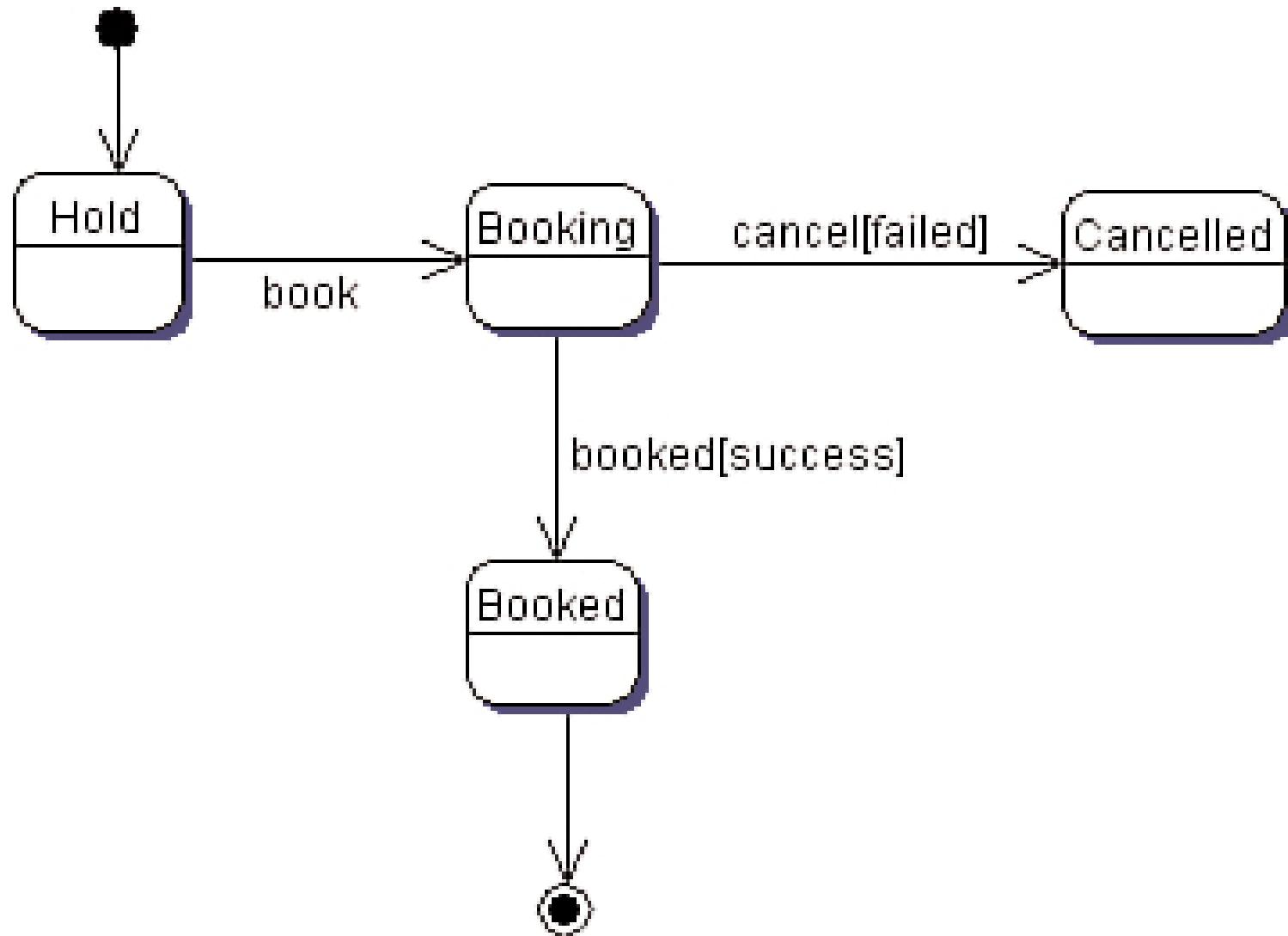
State machine consisting of states, transitions, events, and activities

Dynamic view of the system

Used for modelling of behavior of interface, class, or collaboration

Event-ordered behavior of an object, reactive systems

Statechart Diagram



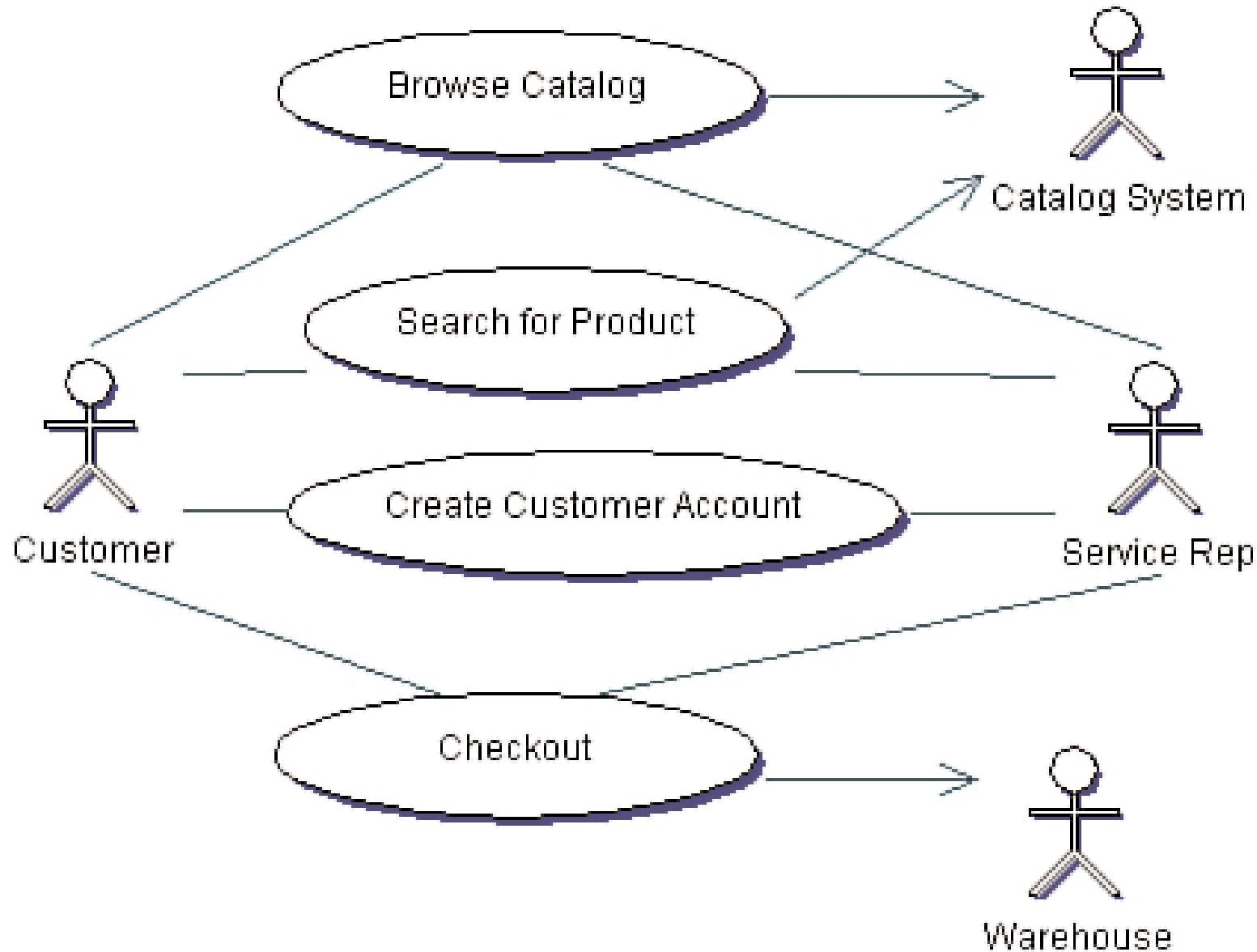
Use-Case Diagram

Set of use cases and actors and their relationships

Static use-case view of a system

Model behaviors of a system

Use-Case Diagram



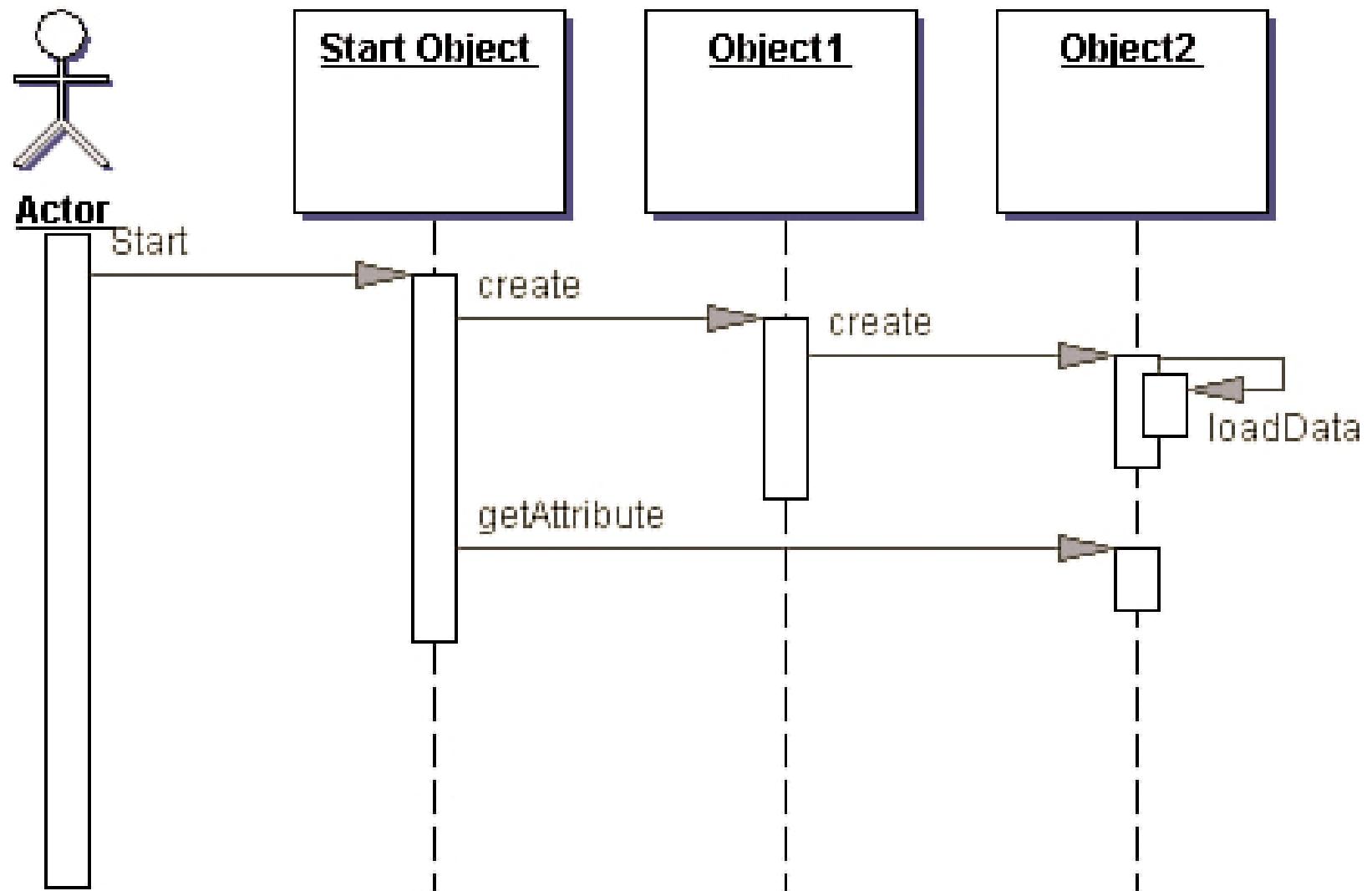
Sequence Diagram

Interaction of a set of objects and their relationships, including the messages dispatched among them

Interaction diagram emphasizing the time ordering of messages.

Communicates flow of events between objects, useful in cases of complex interactions.

Sequence Diagram



Collaboration Diagram

Interaction of a set of objects and their relationships, including the messages dispatched among them

Interaction diagram emphasizing the structural organization of the objects that send and receive messages. Isomorphic with sequence diagram.

Collaboration Diagram

