NAVRH IS – UML

Jaroslav Žáček
jaroslav.zacek@osu.cz
http://www1.osu.cz/~zacek/

TROCHU HISTORIE NEUŠKODÍ…

- Do roku 1994 chaos ve světě objektově orientovaných metod (několik jazyků pro vizuální modelování, několik metod).
- Metody Booch a OMT pro vizuální modelování, Objectory (Jacobson) mezi metodikami.
- Jedním z prvních pokusů o sjednocení byla metodika Fusion (1994) - do její přípravy nebyli zapojeni tvůrci výše zmíněných metod s největším podílem na trhu (Booch, Jacobson, Rumbaugh). Neujala se.
- Booch a Rumbaugh se spojili ve firmě Rational Corporation a začali pracovat na tvorbě jazyka UML. UML se stal otevřeným standardem.
- 1996 – OMG navrhlo UML jako standard objektově orientovaného jazyka pro vizuální modelování.
- 1997 standard OMG přijat.
- UML přebírá to nejlepší, co doposud vzniklo a integruje to dohromady.

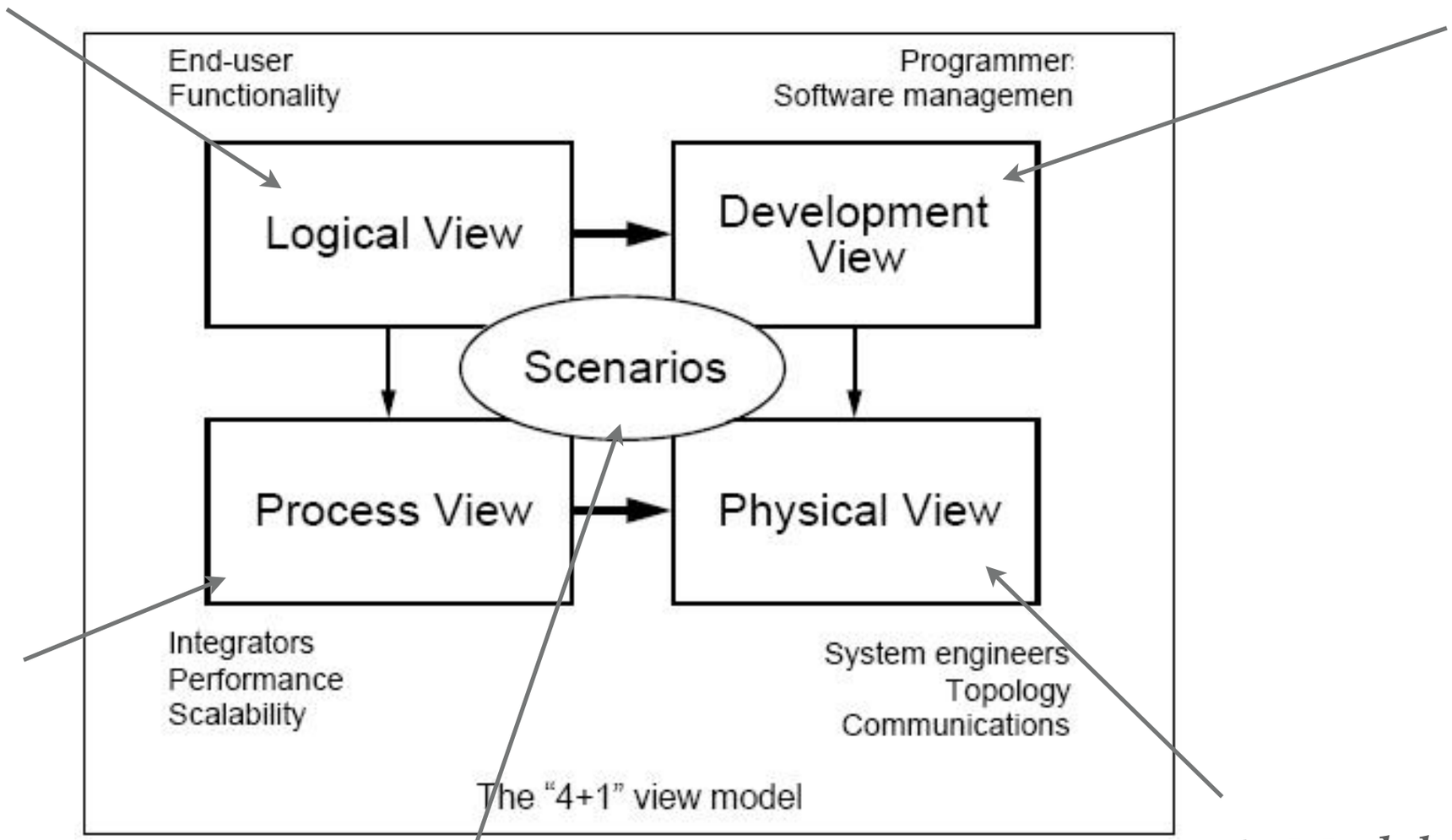
UML

- UML není metodikou ani programovacím jazykem, je to pouze vizuální modelovací nastroj pro objektově orientované systémy.
- S žádnou konkrétní metodikou také není svázán. Lze jej použít se všemi existujícími.
- Nejlépe adaptováno pro použití s UP/RUP.
- UML nabízí vizuální syntaxi pro modelování během celého vývojového cyklu (analýza až nasazení).
- UML je nezávislý na programovacím jazyku. Nejlepší použití je samozřejmě s OO.
- Poslední verze 1.12.2017 (https://www.omg.org/spec/**UML**/2.5.1)

ARCHITEKTURA 4+1

2. Konceptuální chování

3. Balíčky, subsystémy

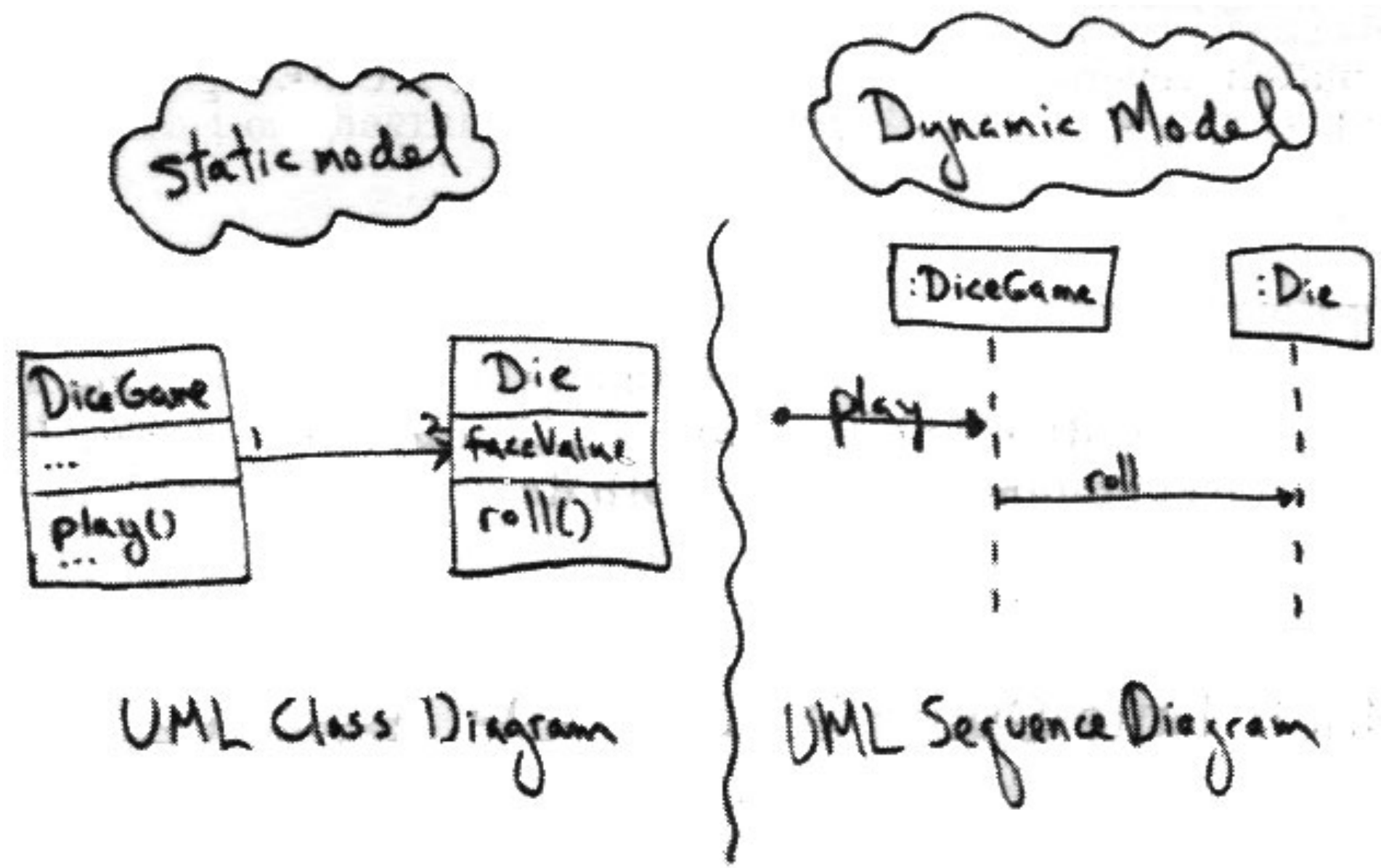


+1 Procesy, thready, systémy

4. Model nasazení

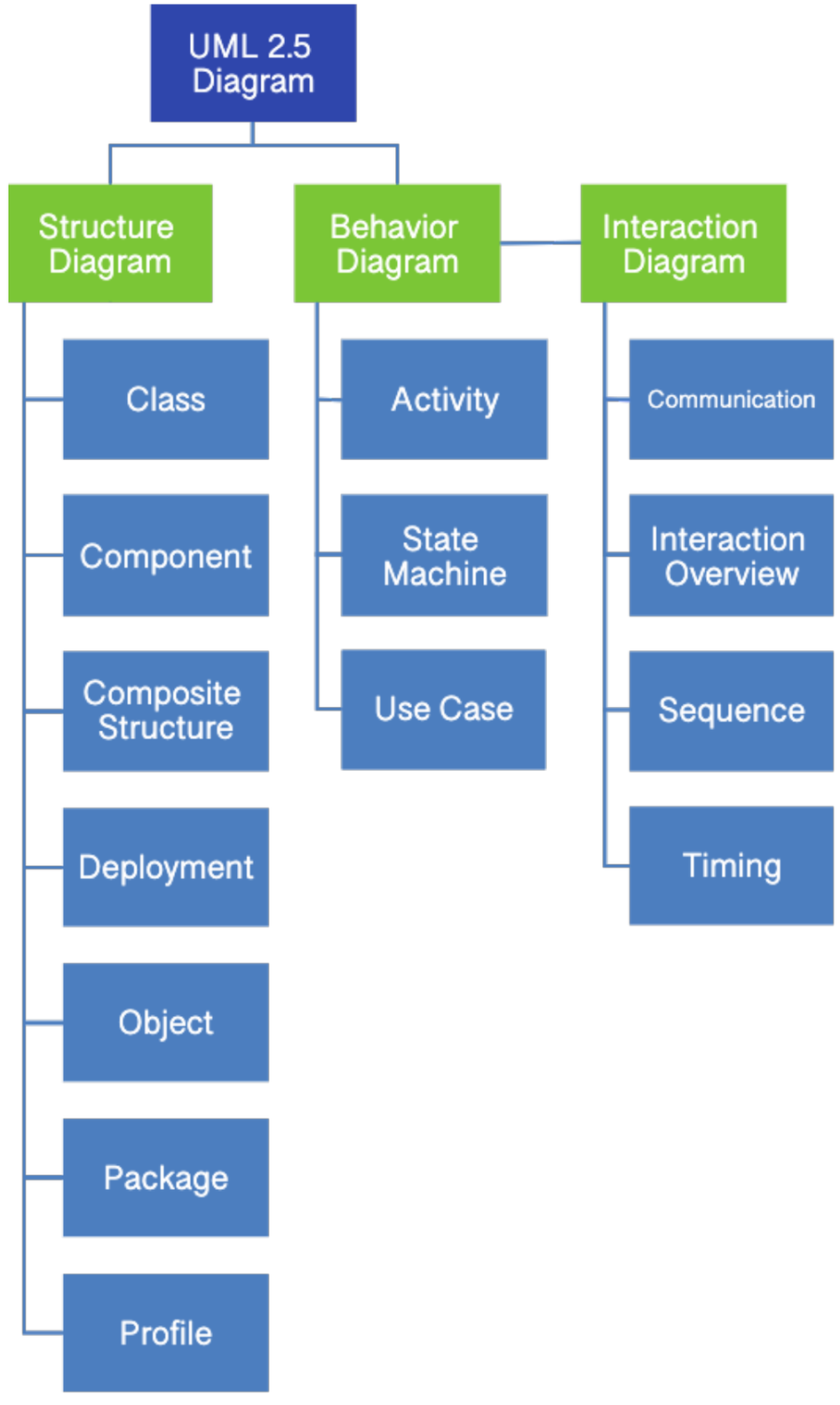
1. Use Case model

TYPY UML


static model
Dynamic Model
DiceGame
...
play()
1
3
Die
faceValue
roll()
:DiceGame
:Die
play
roll
UML Class Diagram
UML Sequence Diagram

TYPY UML

- Statický pohled
  - Class diagram
  - Component diagram
  - Deployment diagram
- Dynamický pohled
  - Object diagram
  - Sequence diagram
  - Collaboration Diagram
  - Statechart diagram
  - Activity diagram
- Funkční pohled
  - Use Case



MODELY

- Základní modely používané v iterativně-inkrementálním vývoji jsou:
  - Use Case
  - Sequence diagram
  - Class diagram

USE CASE DLE OMG

18.2.5.6 Constraints

- binary_associations
  UseCases can only be involved in binary Associations.

```
inv: Association.allInstances()->forAll(a | a.memberEnd.type->includes(self) implies
a.memberEnd->size() = 2)
```

- no_association_to_use_case
  UseCases cannot have Associations to UseCases specifying the same subject.

```
inv: Association.allInstances()->forAll(a | a.memberEnd.type->includes(self) implies
   (
   let usecases: Set(UseCase) = a.memberEnd.type->select(oclIsKindOf(UseCase))-
>collect(oclAsType(UseCase))->asSet() in
   usecases->size() > 1 implies usecases->collect(subject)->size() > 1
   )
)
```

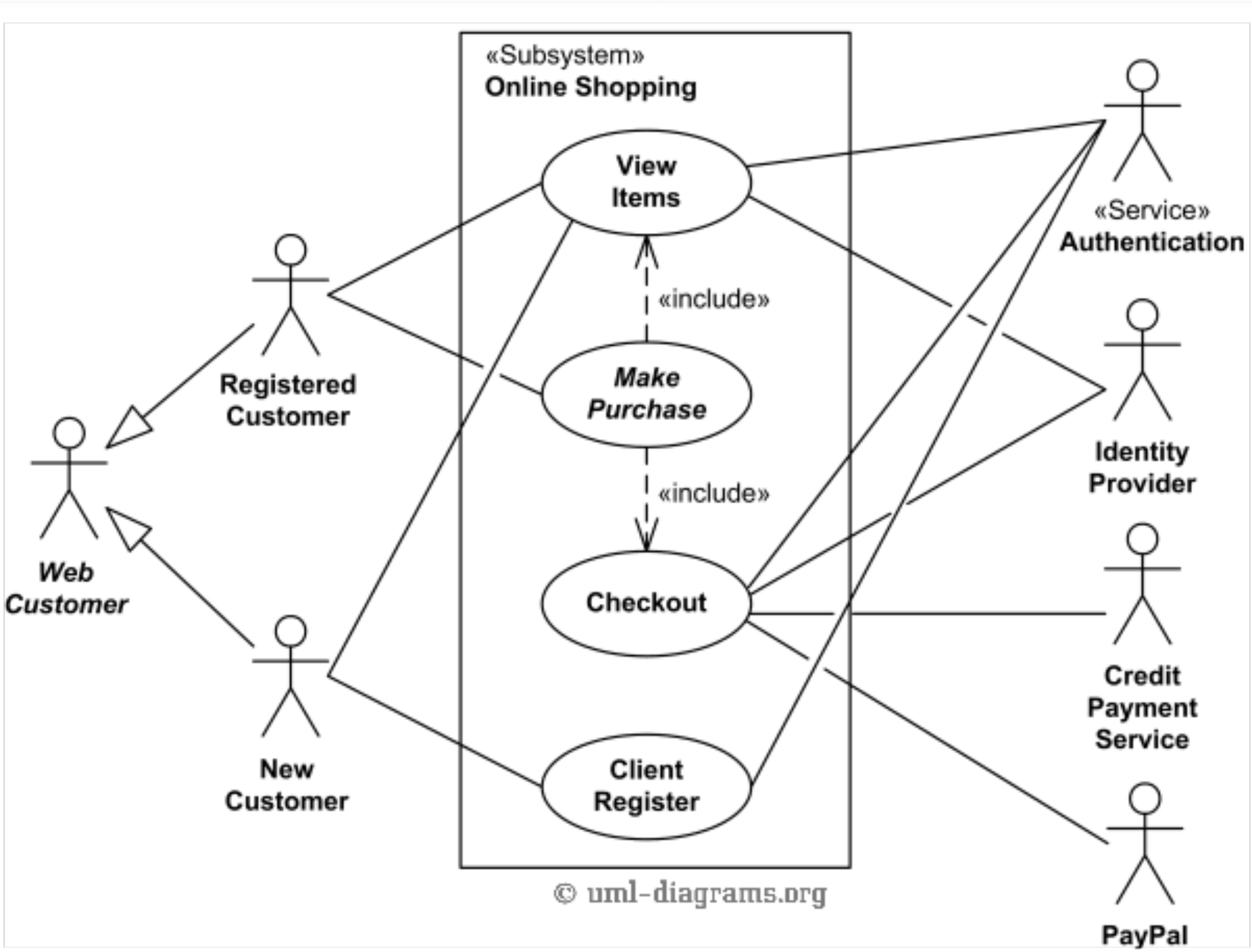
- cannot_include_self
  A UseCase cannot include UseCases that directly or indirectly include it.

```
inv: not allIncludedUseCases()->includes(self)
```

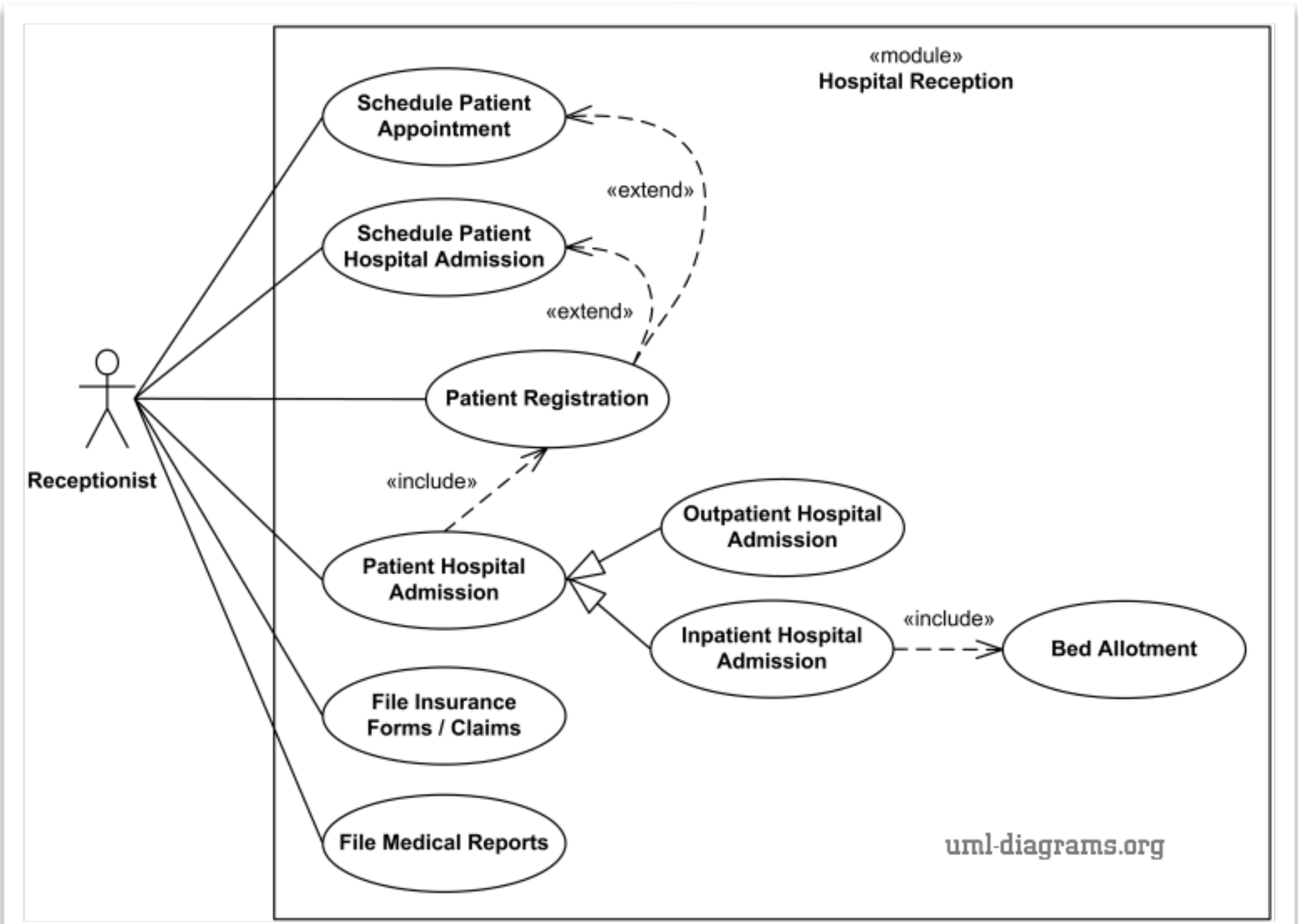
- must_have_name
  A UseCase must have a name.

```
inv: name -> notEmpty ()
```

USE CASE


«Subsystem»
Online Shopping
View Items
«include»
Make Purchase
«include»
Checkout
Client Register
Registered Customer
Web Customer
New Customer
«Service»
Authentication
Identity Provider
Credit Payment Service
PayPal
© uml-diagrams.org

Use Case

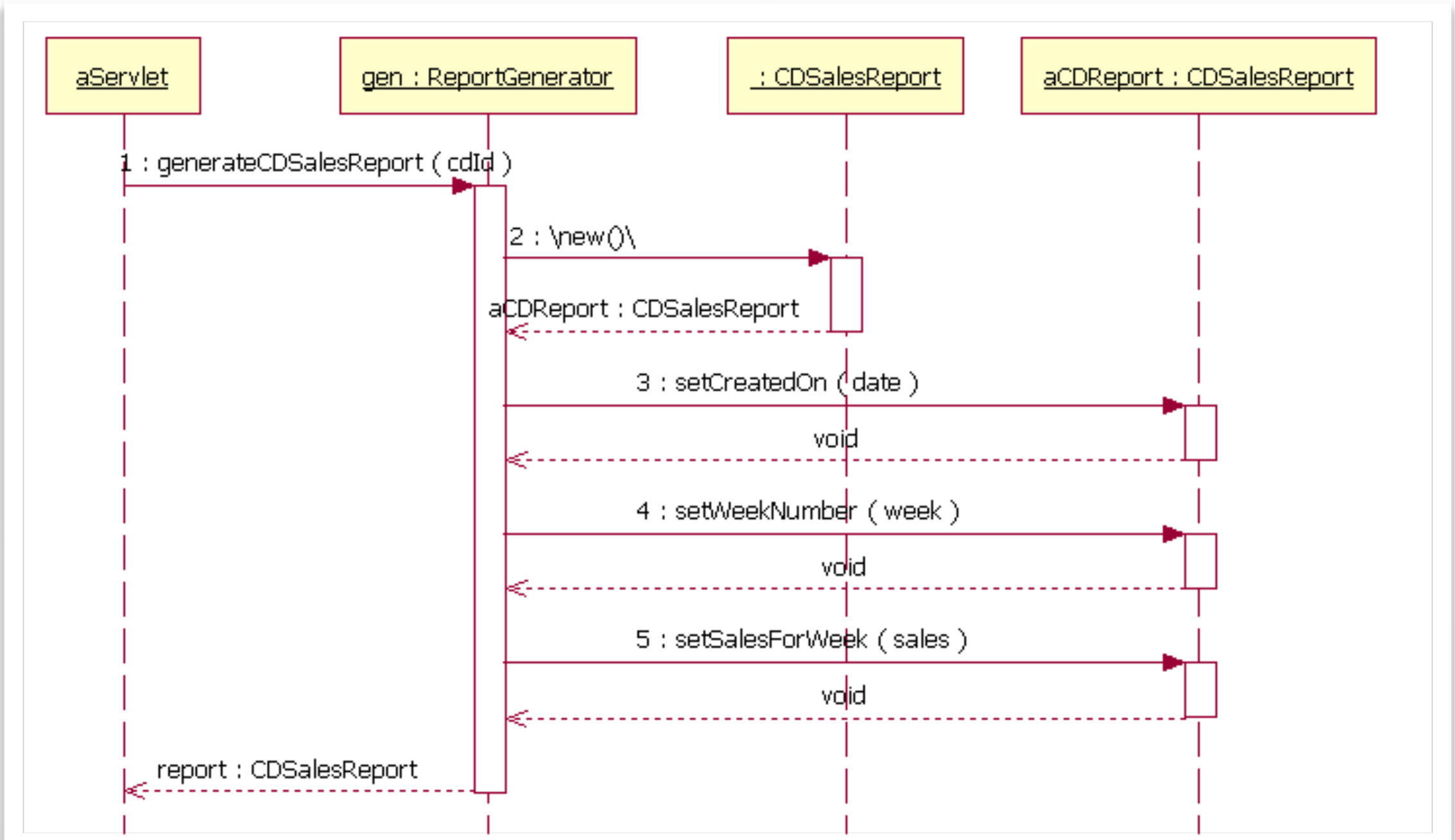

«module»
Hospital Reception
Schedule Patient Appointment
«extend»
Schedule Patient Hospital Admission
«extend»
Patient Registration
Receptionist
«include»
Outpatient Hospital Admission
Patient Hospital Admission
Inpatient Hospital Admission
«include»
Bed Allotment
File Insurance Forms / Claims
File Medical Reports
uml-diagrams.org

USE CASE

- Správné použití:
    - Identifikace funkčních domén.
    - Identifikace aktorů systému.
- Špatné použití:
    - Funkční dekompozice
    - Design Use Cases
    - CRUDL

SEQUENCE DIAGRAM

- *Dynamický pohled - vyskytuje se v analýze i v návrhu*



sd submit_commens

:window

«javascript»
:Comments

«servlet»
:DWRServlet

«service»
:DWRService

«javascript»
:PluckRequest
Batch

«service»
:PluckService

validate()

validate()

«create»

«ajax»
:Proxy

«ajax»

«ajax»

validate()

- form fields validation
- CAPTCHA check

errors

errors

{10..200ms}

«callback»

opt

[no errors]

post_comments()

BeginRequest()

«create»

«ajax»
:Proxy

{10..100ms}

«ajax»

postComments()

{1s..4s}

«callback»

«json»

ref

Handle errors from Pluck

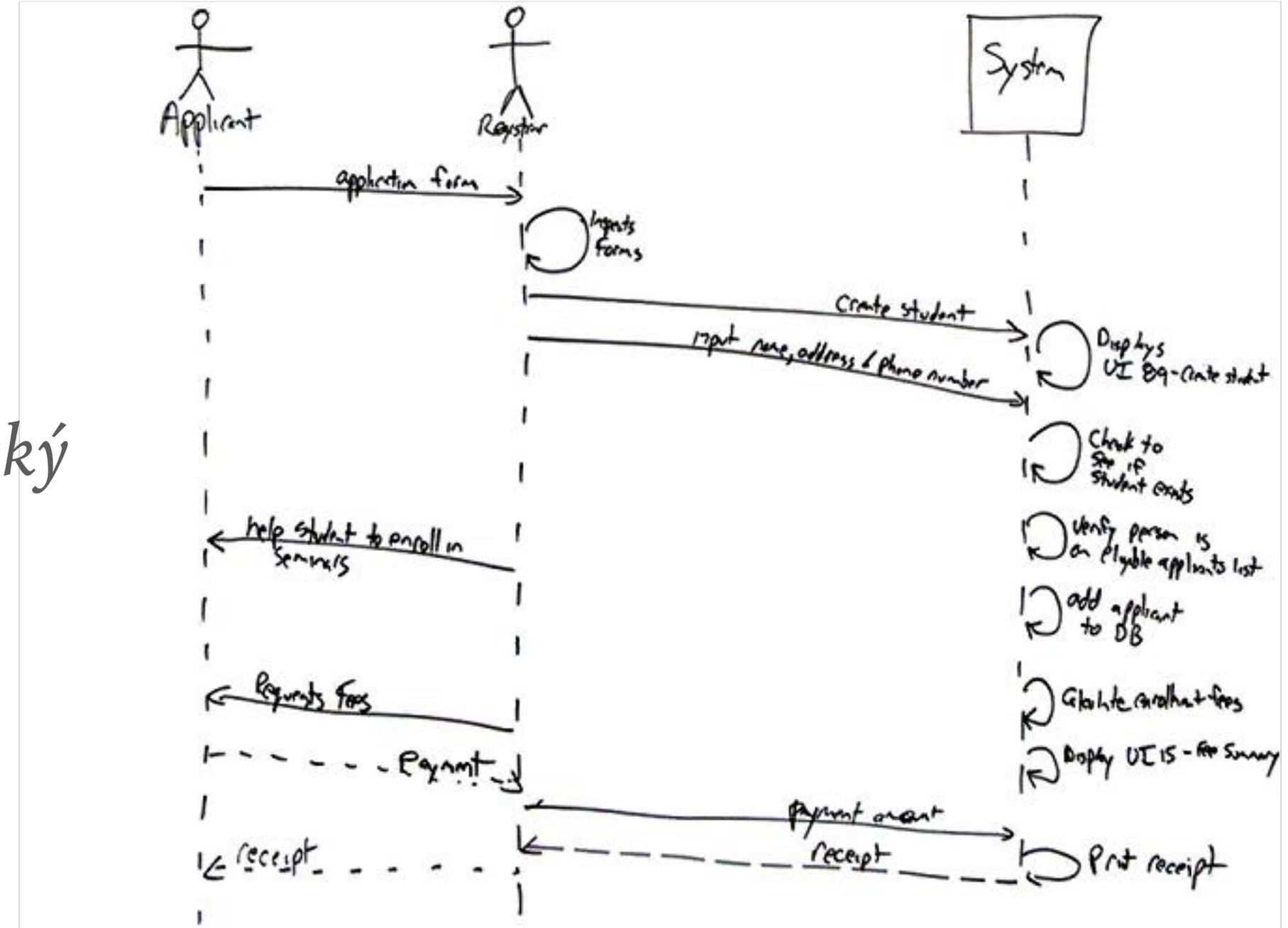
{10..100ms}

ref

Request all comments

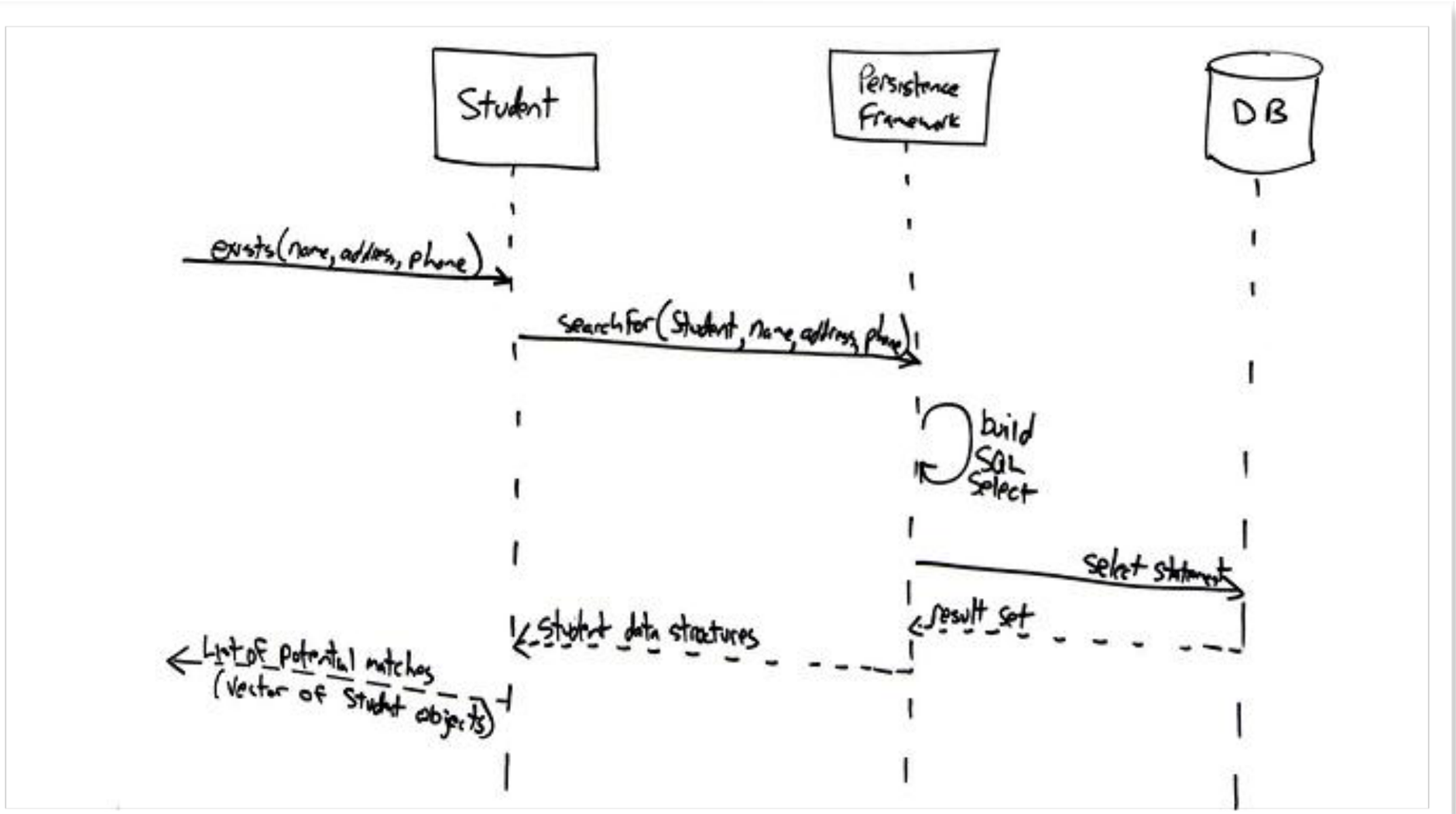
SEQUENCE DIAGRAM MÉNĚ FORMÁLNĚ

Analytický



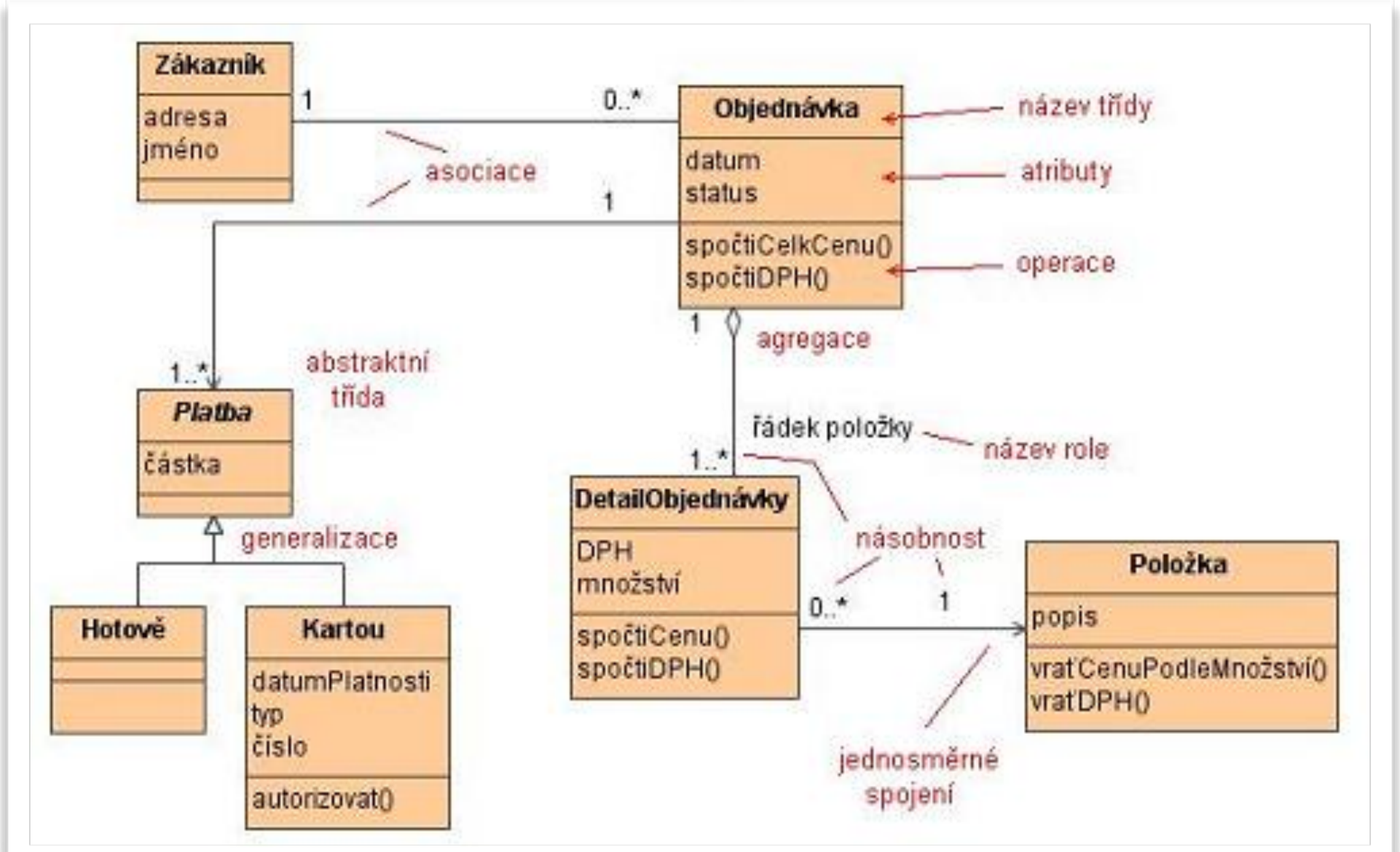
SEQUENCE DIAGRAM MÉNĚ FORMÁLNĚ

Návrhový

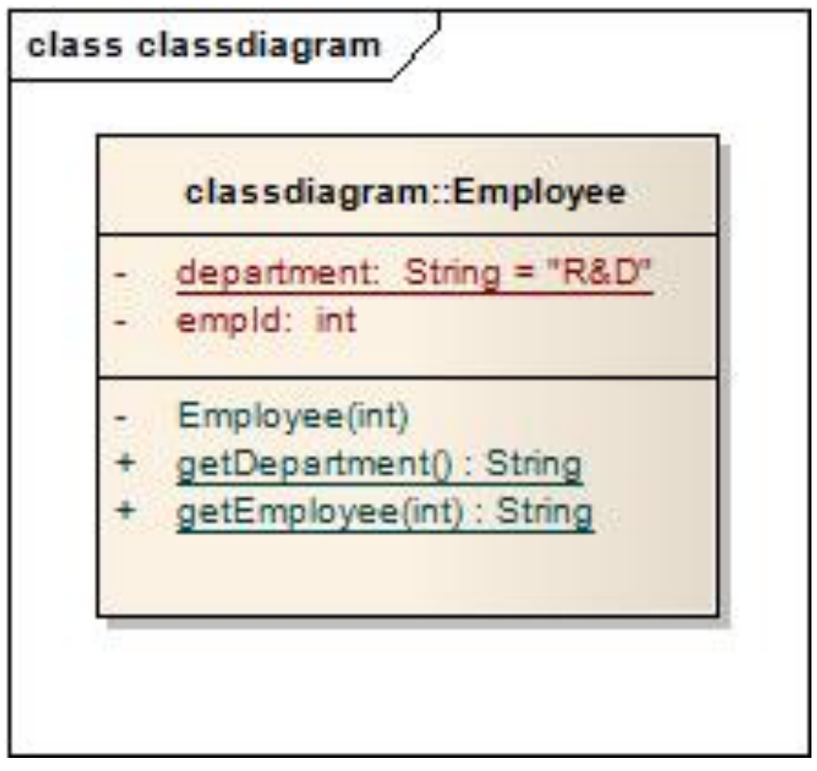


CLASS DIAGRAM

- Statický pohled na systém - třídy a jejich relace

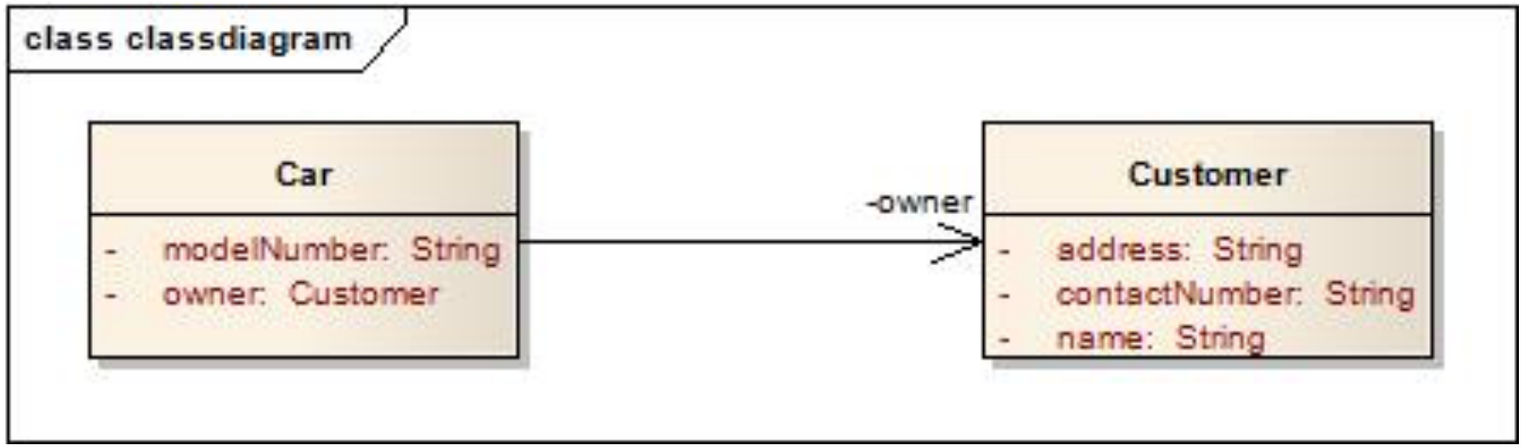


Class diagram




```java
public class Employee {
    private static String department = "R&D";
    private int empId;
    private Employee(int employeeId) {
        this.empId = employeeId;
    }
    public static String getEmployee(int emplId) {
        if (emplId == 1) {
            return "idiotechie";
        } else {
            return "Employee not found";
        }
    }
    public static String getDepartment() {
        return department;
    }
}
```

Class diagram

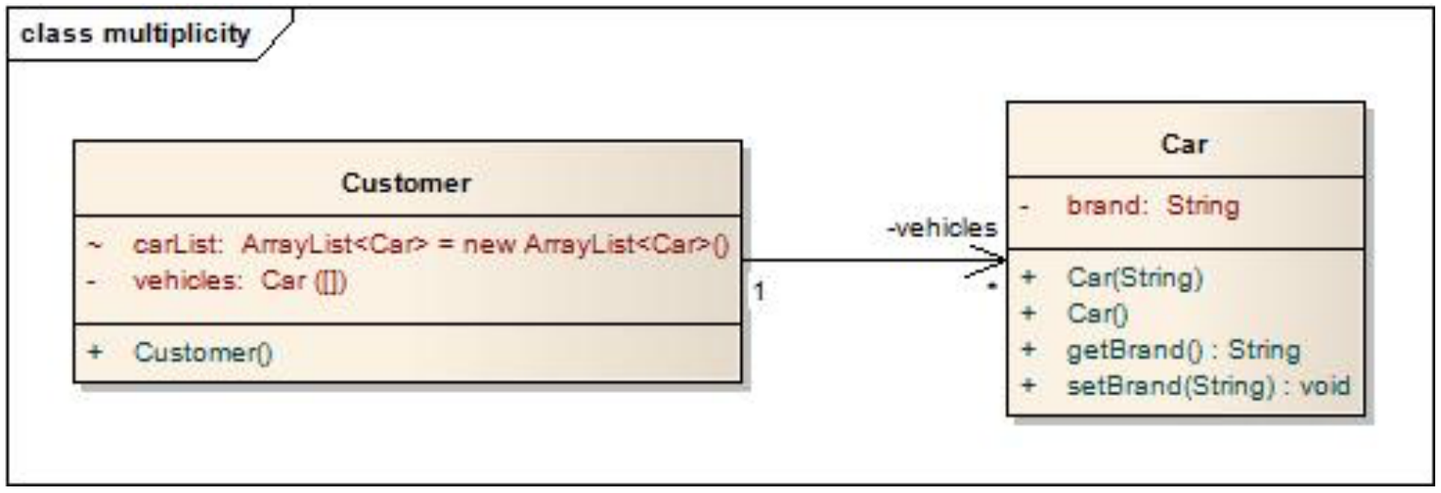



```java
public class Customer {
private String name;
private String address;
private String contactNumber;
}

public class Car {
    private String modelNumber;
    private Customer owner;
}
```

Class diagram


```java
public class Customer {
private String name;
private String address;
private String contactNumber;
private Car car;
}

public class Car {
    private String modelNumber;
    private Customer owner;
}
```

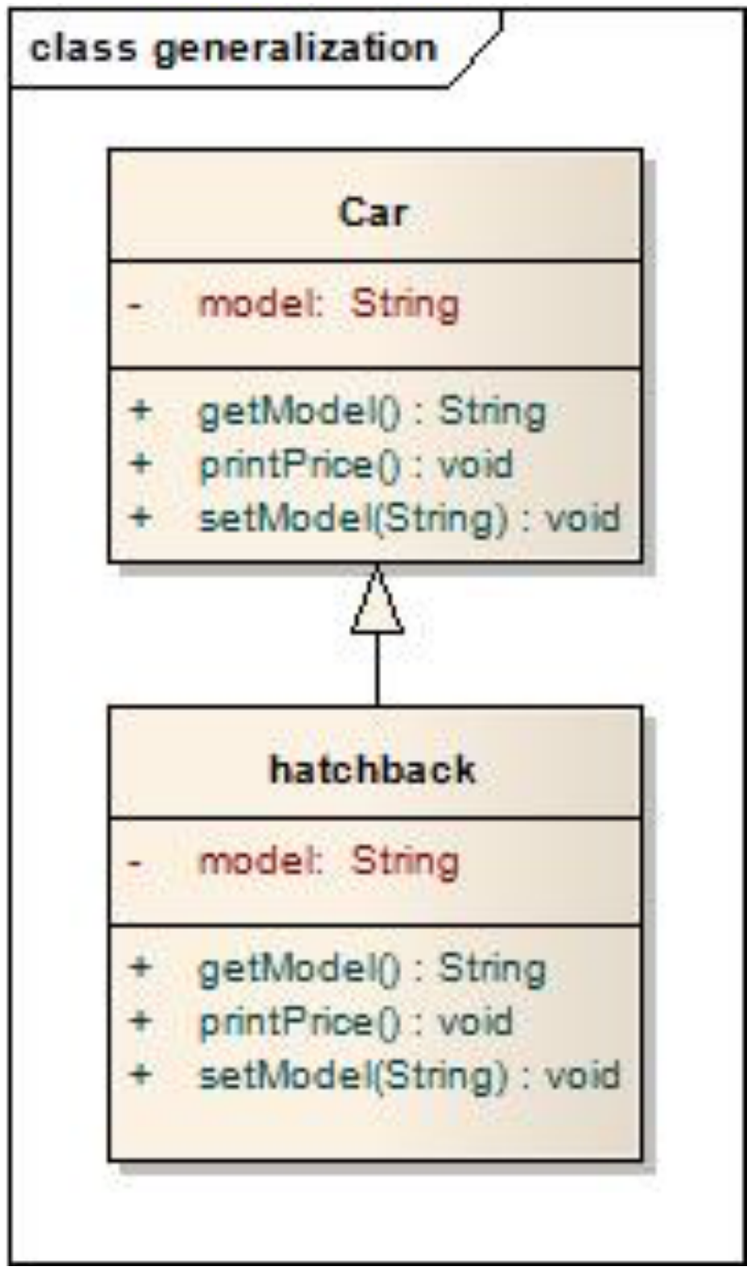
Class diagram




```java
public class Customer {
    private Car[] vehicles;
    ArrayList<Car> carList = new ArrayList<Car>();
    public Customer(){
        vehicles = new Car[2];
        vehicles[0] = new Car("Audi");
        vehicles[1] = new Car("Mercedes");

        carList.add(new Car("BMW"));
        carList.add(new Car("Chevy"));
    }
}
```

```java
public class Car {
    private String brand;

    public Car(String brands){
        this.brand = brands;
    }
    public Car() {
    }
    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

}
```

Class diagram



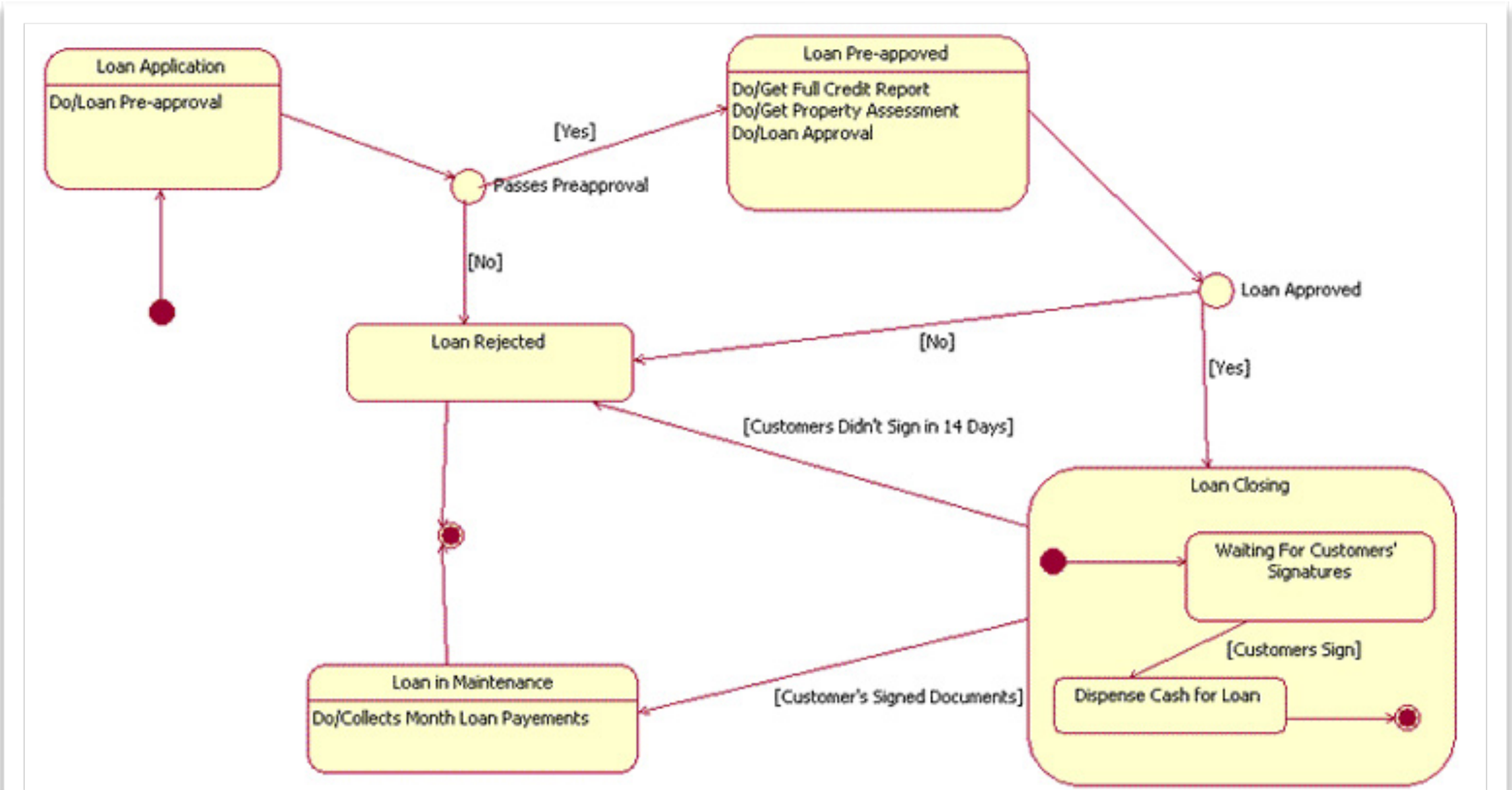

```java
public class Car {
    private String model;
    public void printPrice() {
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }
}

public class hatchback extends Car {
    private String model;
    public void printPrice() {
        System.out.println("Hatchback Price");
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }
}
```

MODELY

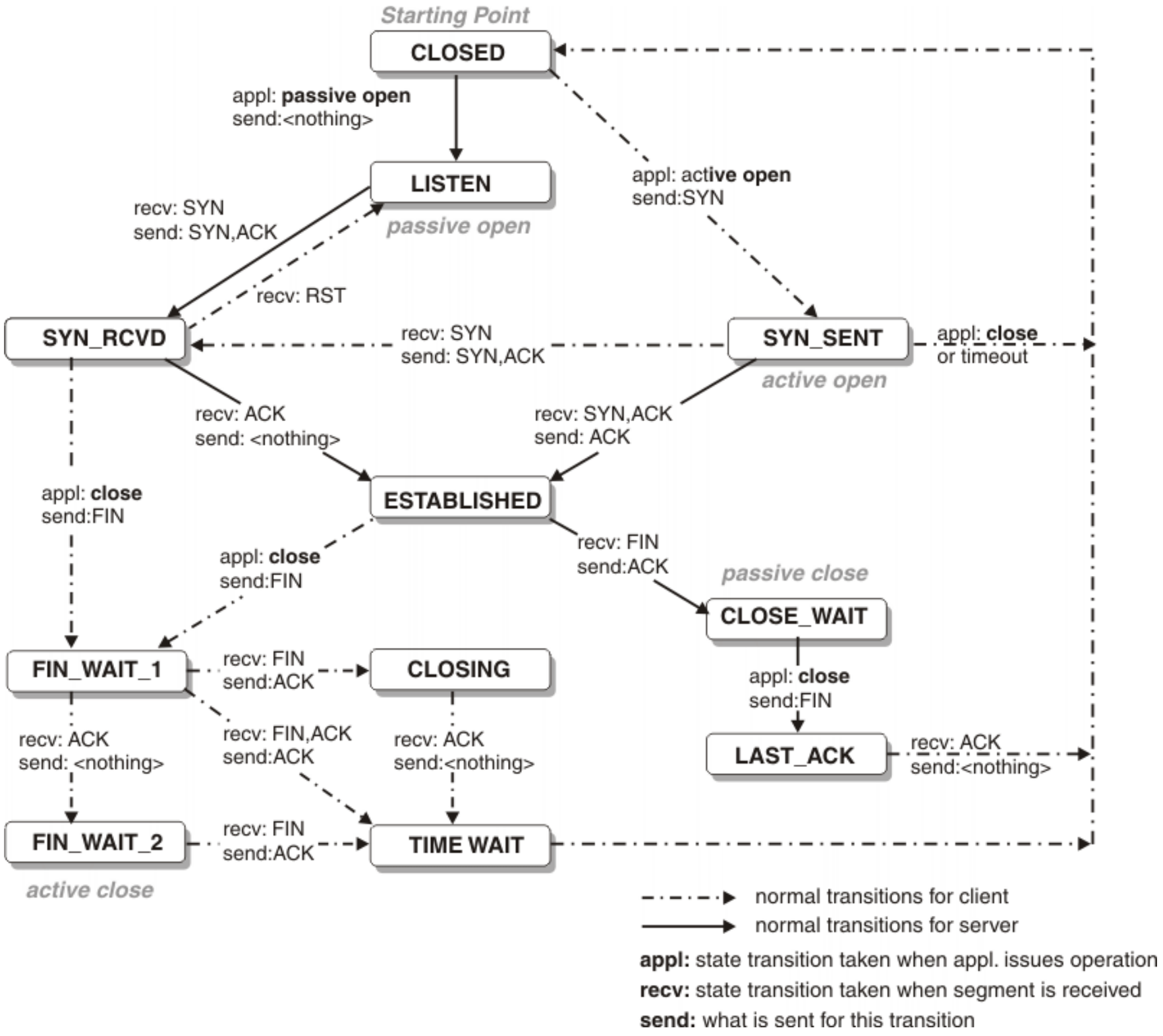
Existují i nějaké jiné?

STATECHART DIAGRAM

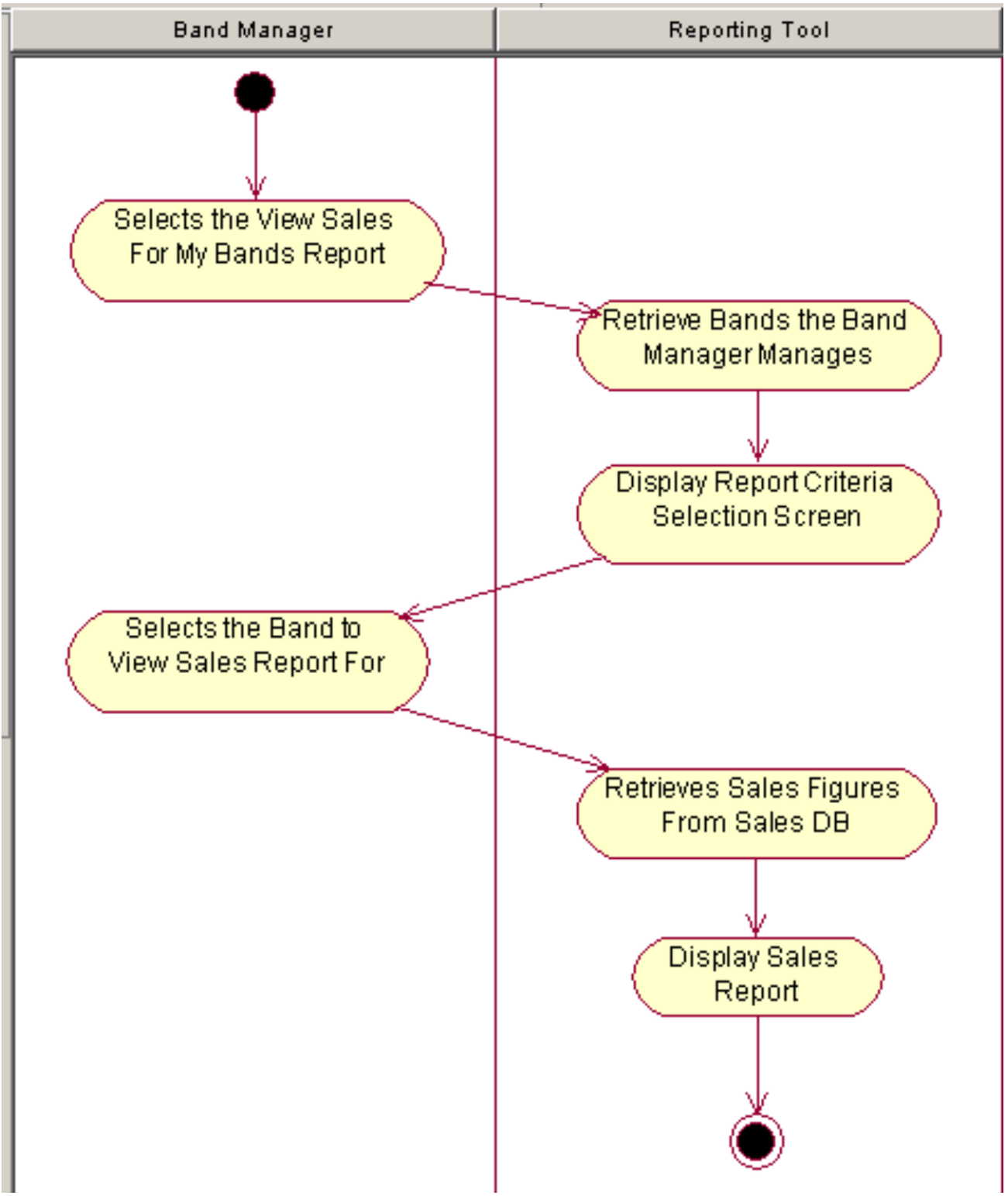
- Změny stavu systému za běhu systému
- Ideální pro popsání stavů protokolu (např. TCP)



STATECHART DIAGRAM



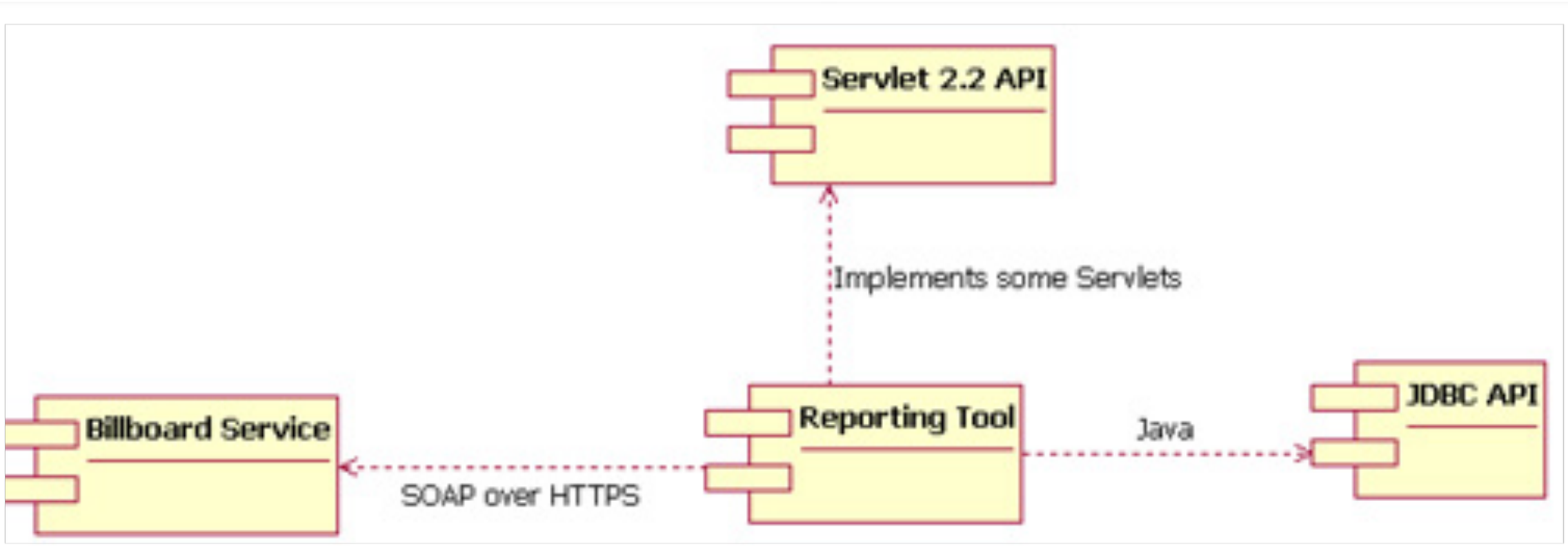
ACTIVITY DIAGRAM



- Popis podnikových procesů
- Tok řídících procesů přes několik objektů

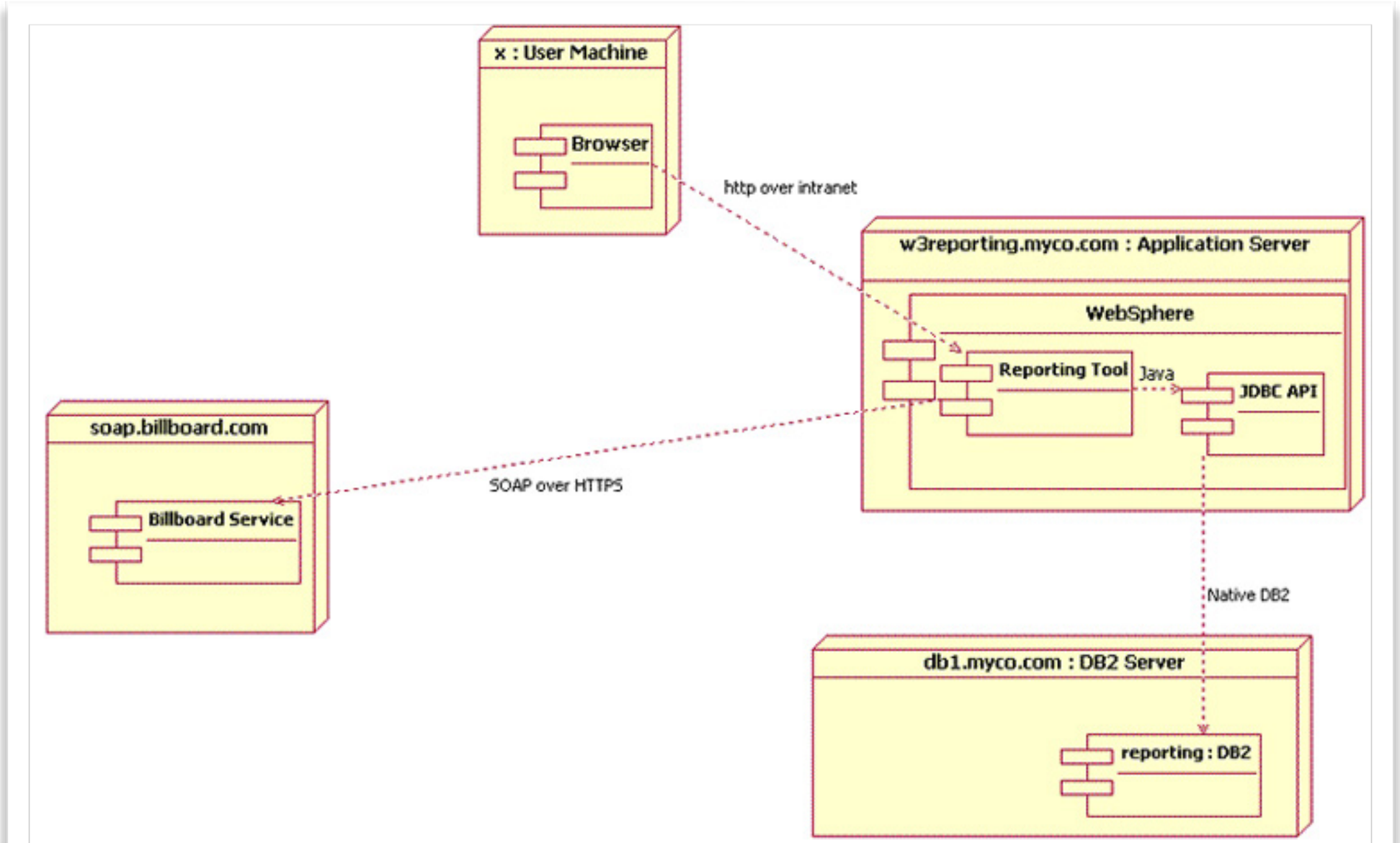
COMPONENT DIAGRAM

- Fyzický pohled na softwarové komponenty a jejich vazby, vztahy, komunikaci



DEPLOYMENT DIAGRAM

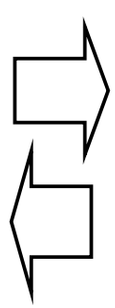
- Jak je softwarový systém nasazen na hardware

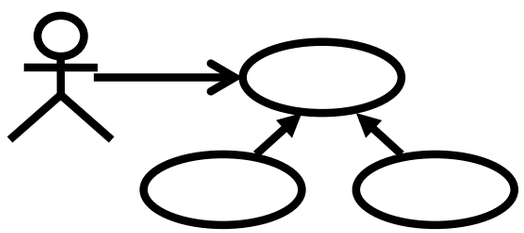


ŘÍZENÍ VÝVOJE IS DLE UML

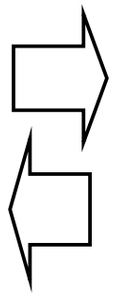
VÝVOJ ŘÍZENÝ USE CASE

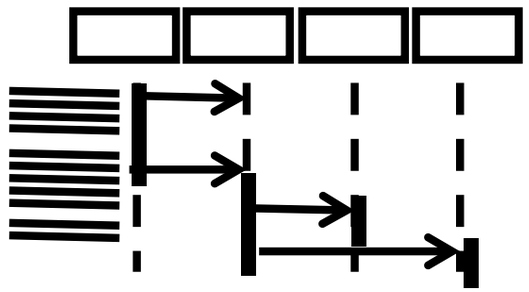
Vision



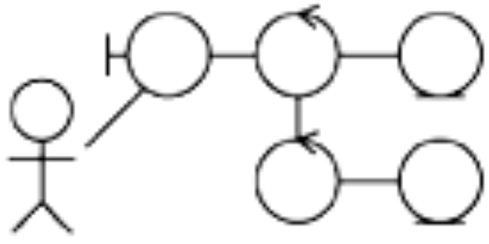


Use-Case Model
+ Popis +
Doplňující spec

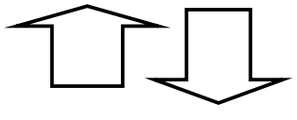


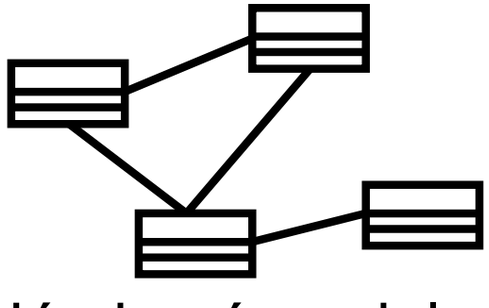


Realizace UC =
Spolupracující entity



Analytické třídy

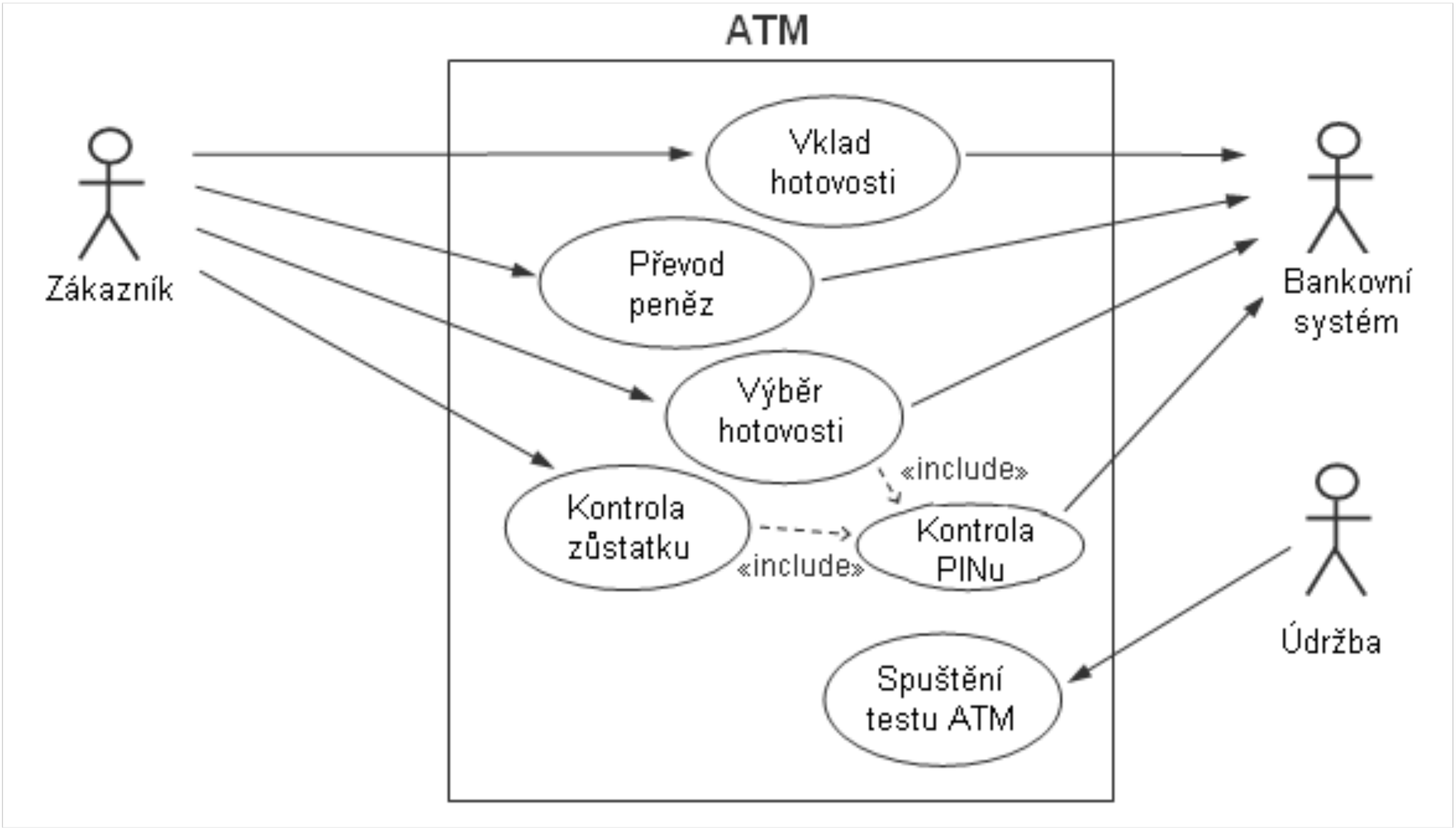




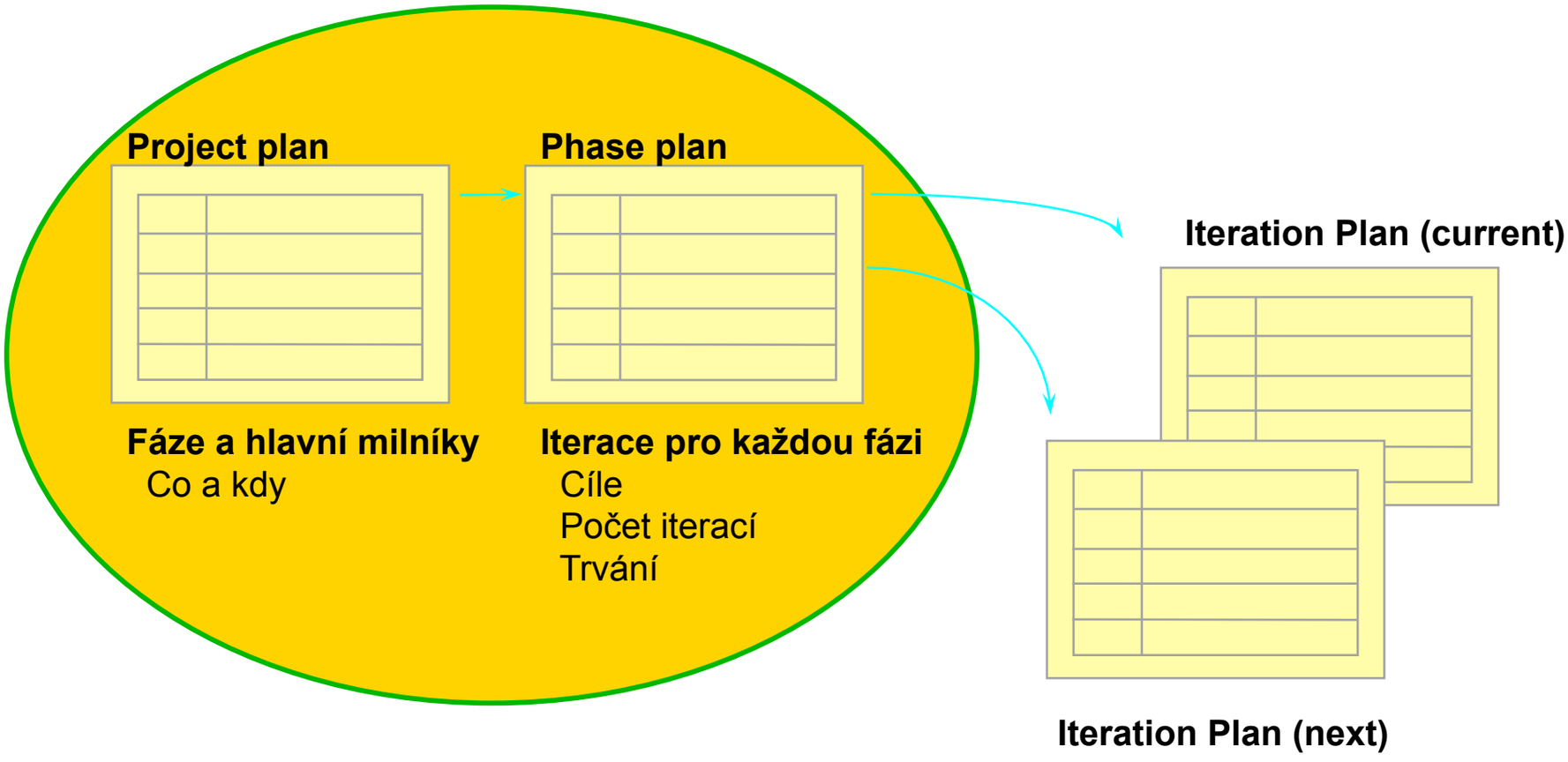
Návrhový model

```java
/* Block comment */
import java.util.Date;
/**
 * Doc comment here for SomeClass
 * @version 1.0
 */
public class SomeClass { // some comment
  private String field = "Hello World";
  private double unusedField = 12345.67890;
  private UnknownType anotherString = "AnotherString";
  public SomeClass() {
   //TODO: something
   int localVar = "IntelliJ"; // Error, incompatible types
   System.out.println(anotherString + field + localVar);
   long time = Date.parse("1.2.3"); // Method is deprecated
  }
}
```

USE CASE MODEL



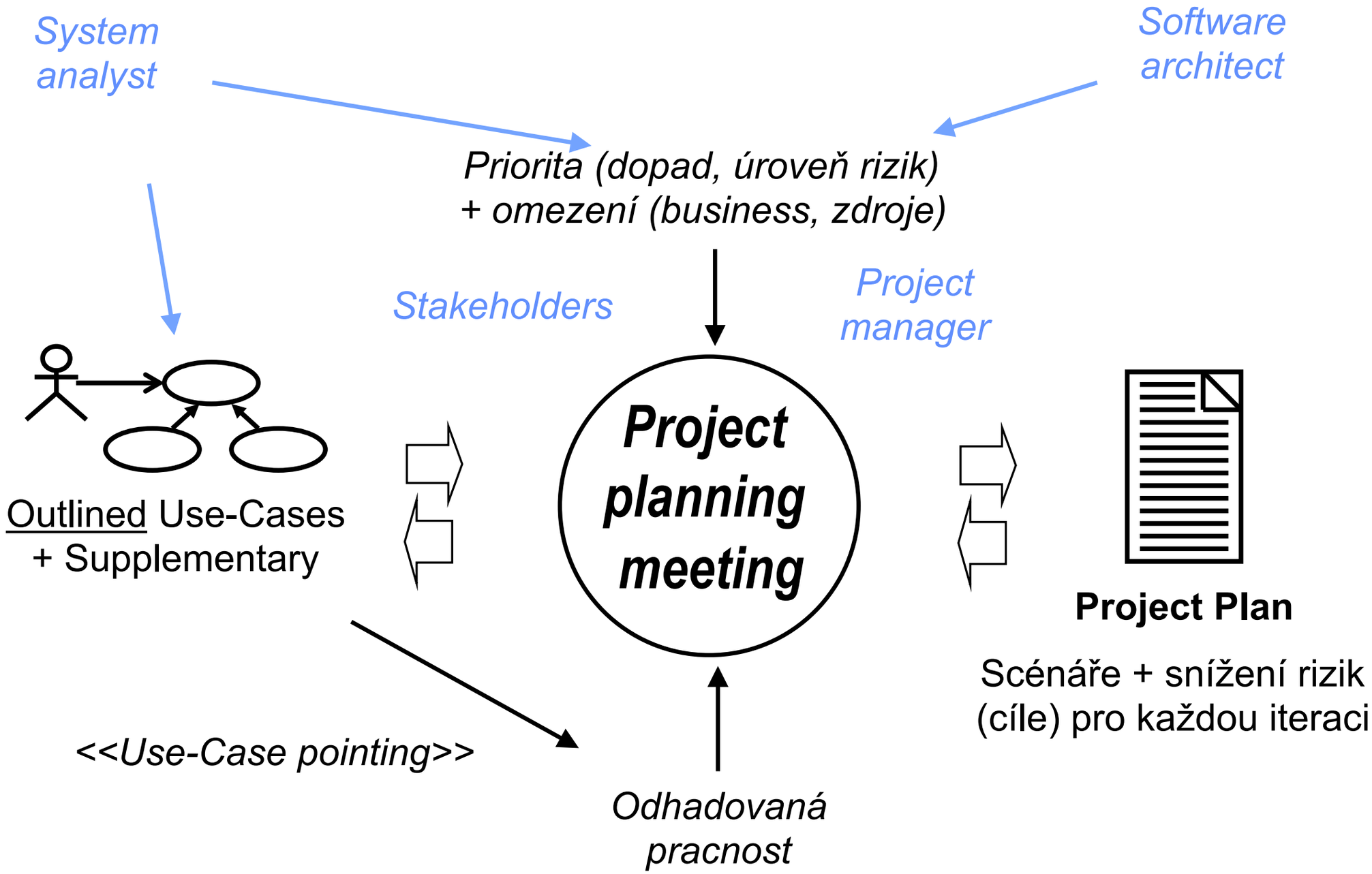
PROJEKTOVÝ PLÁN – DVOUÚROVŇOVÉ PLÁNOVÁNÍ



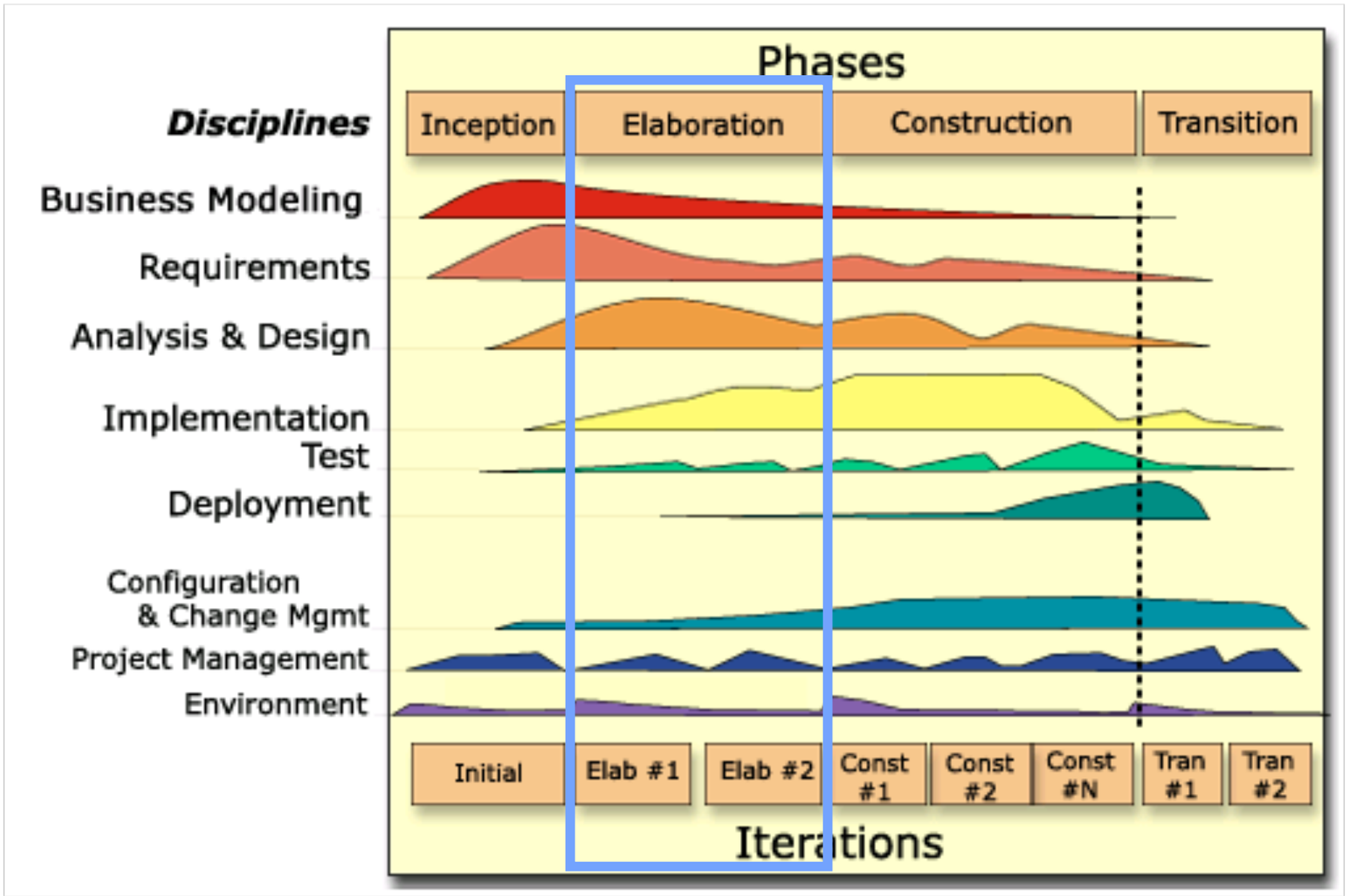
Road-map - hrubý plán

Detailní plánování

CÍLE ITERACE



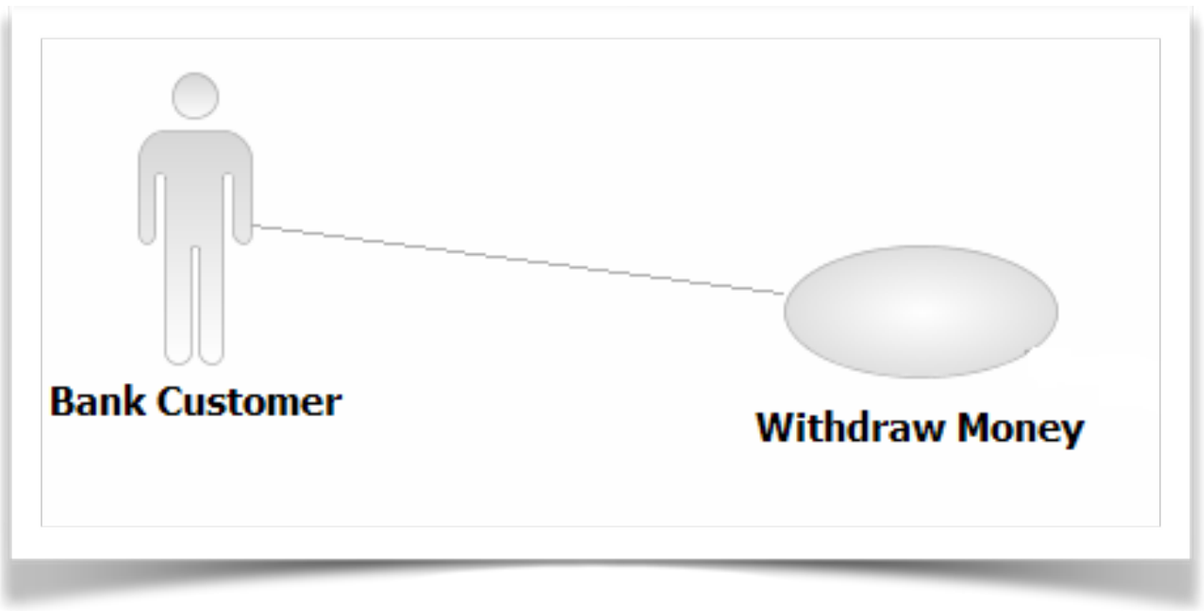
ELABORATION



Elaboration - implementovány use case/scénáře za účelem snížení rizik a ohodnocení architektury.

USE CASES

- Výběr hotovosti
    - Přiřazené rysy - F1, F2, F3
    - Status: Outlined
    - Priorita: HI
    - Riziko: HI
    - Důležitost: HI
- Plánovaná verze: v.1.0.

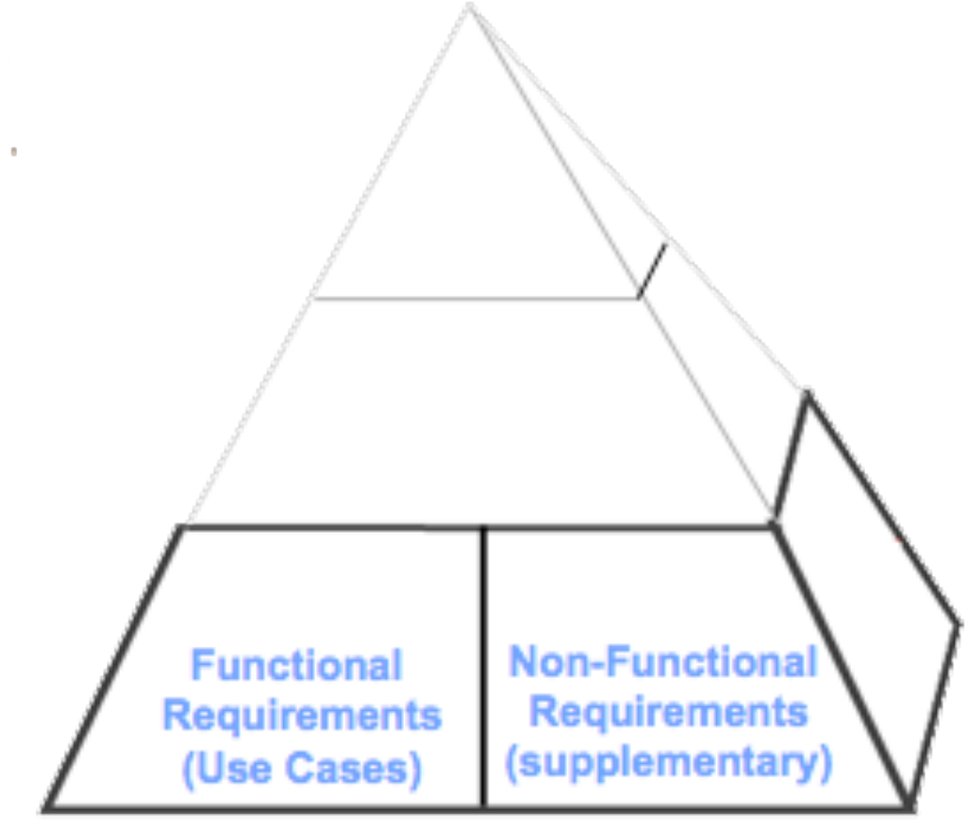


UC – VÝBĚR HOTOVOSTI

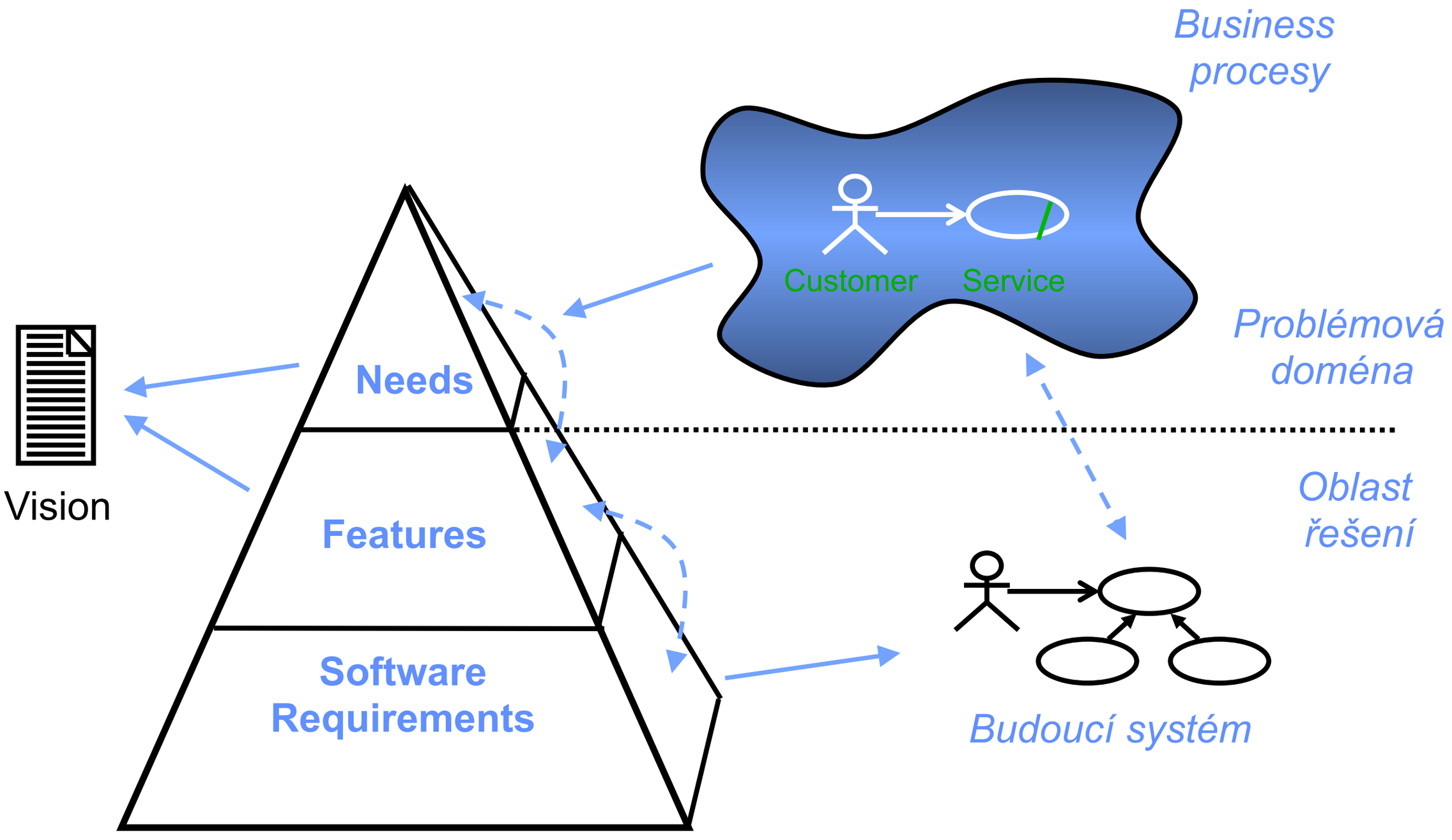
- Basic Flow
  1. Zákazník se identifikuje. *Klíčový a rizikový scénář*
  2. Systém se zeptá na obnos k vybrání.
  3. Zákazník vloží částku.
  4. Systém odečte peníze z účtu a vydá je.
  5. Zákazník peníze odebere.
- Alternative flows: *V průběhu iterace je UC konzultován se zákazníkem*
  - Zákazník chce potvrzení o transakci.
  - Zákazník peníze neodebere - storno transakce.

OSTATNÍ POŽADAVKY?

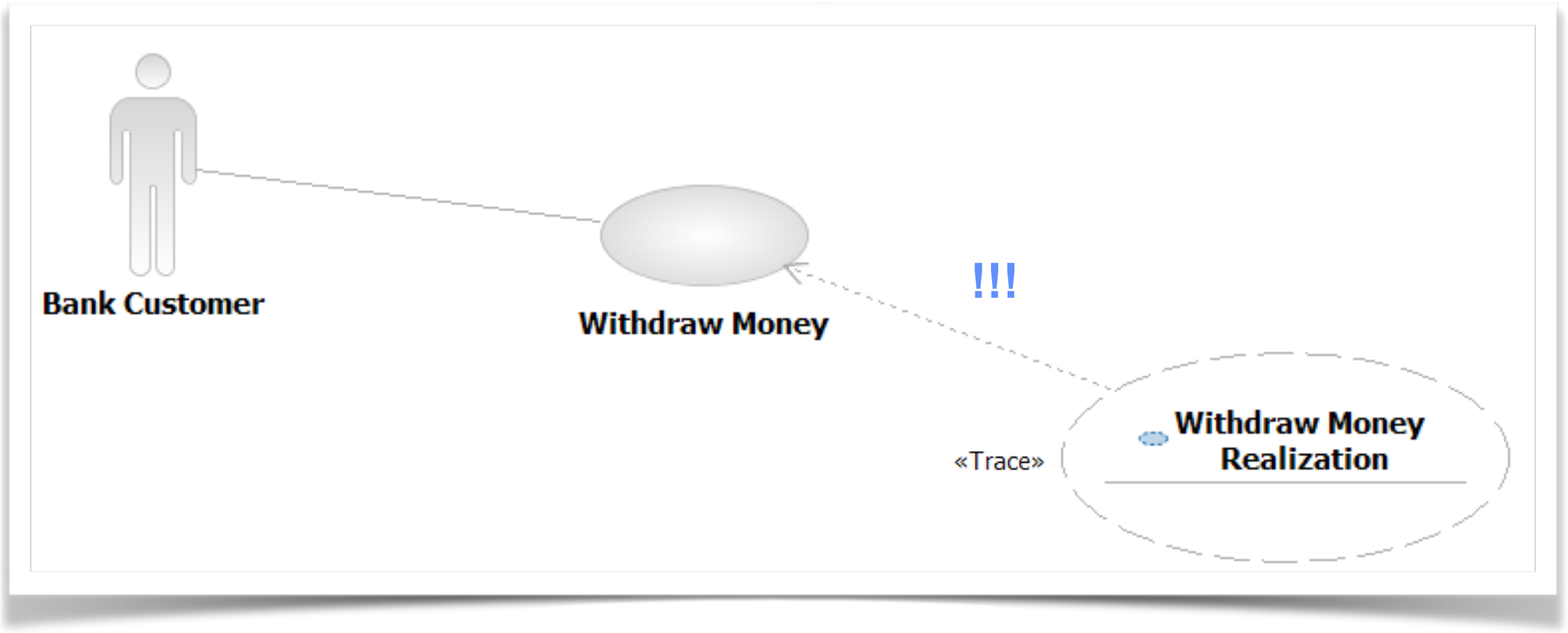
- Systém musí být distribuovaný, proto aby mohl poskytovat bankovní služby daleko od pobočky
- Systém by měl odpovědět na výběr v 90% případů do 10 sec.



MAPA


Business procesy
Customer
Service
Problémová doména
Needs
Features
Software Requirements
Vision
Oblast řešení
Budoucí systém

UC – REALIZACE

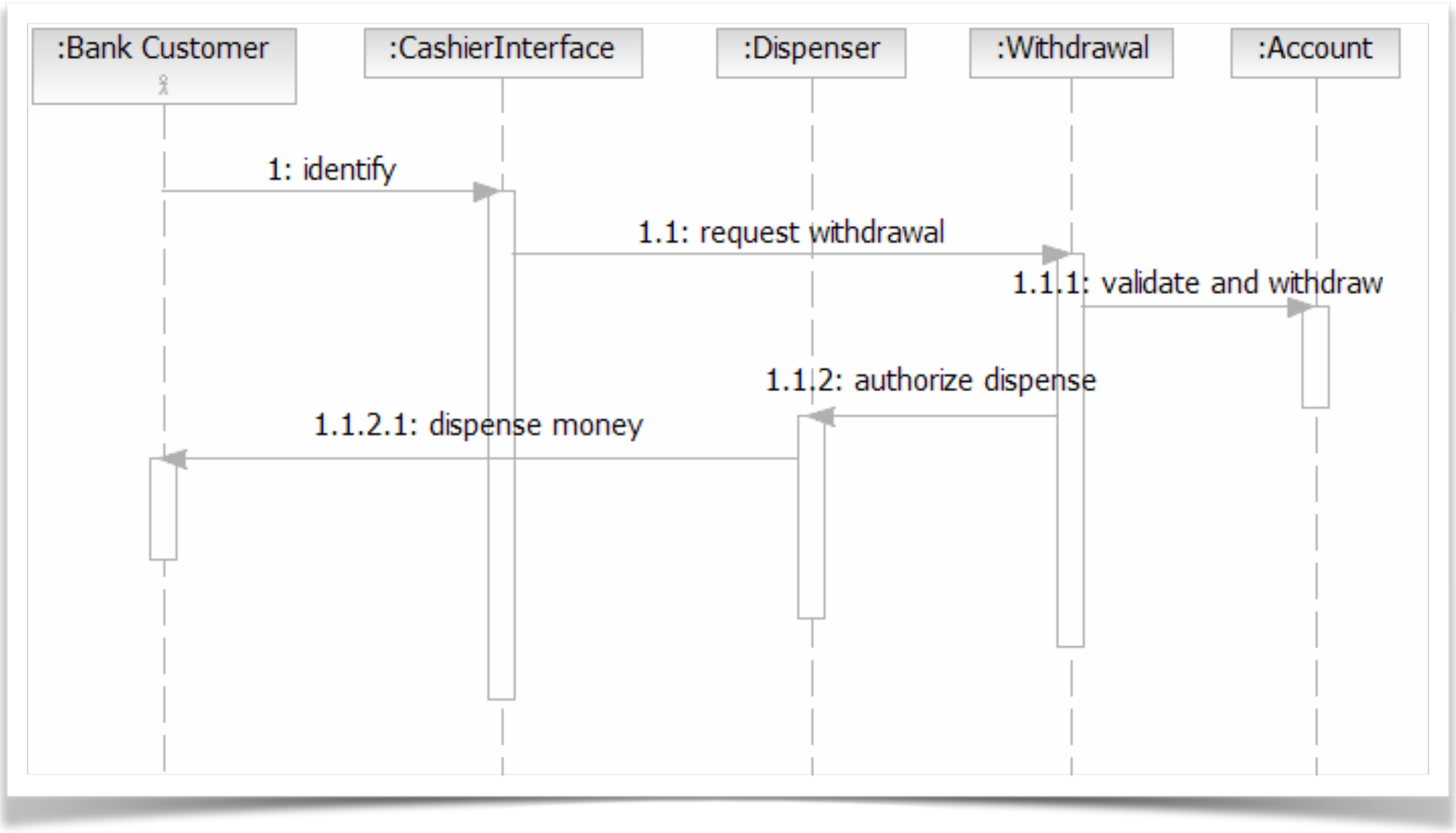


Analýza: Primárním cílem je definovat/zachytit model problémové domény.

Analýza se zaměřuje na to ***CO dělat***. Návrh (Design) se zaměřuje na to ***JAK to udělat***.

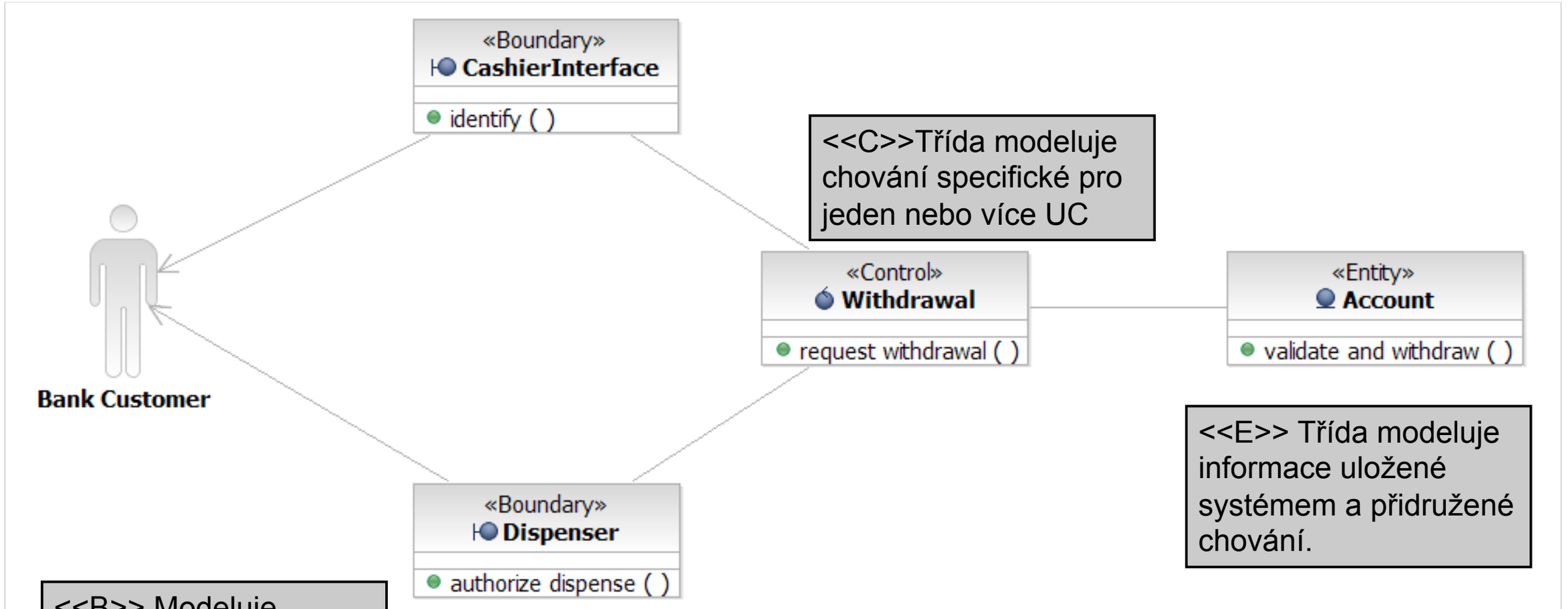
UCR: UC realizace popisuje, jak je daný UC realizován modelem spolupracujících objektů.

UC – VÝBĚR HOTOVOSTI



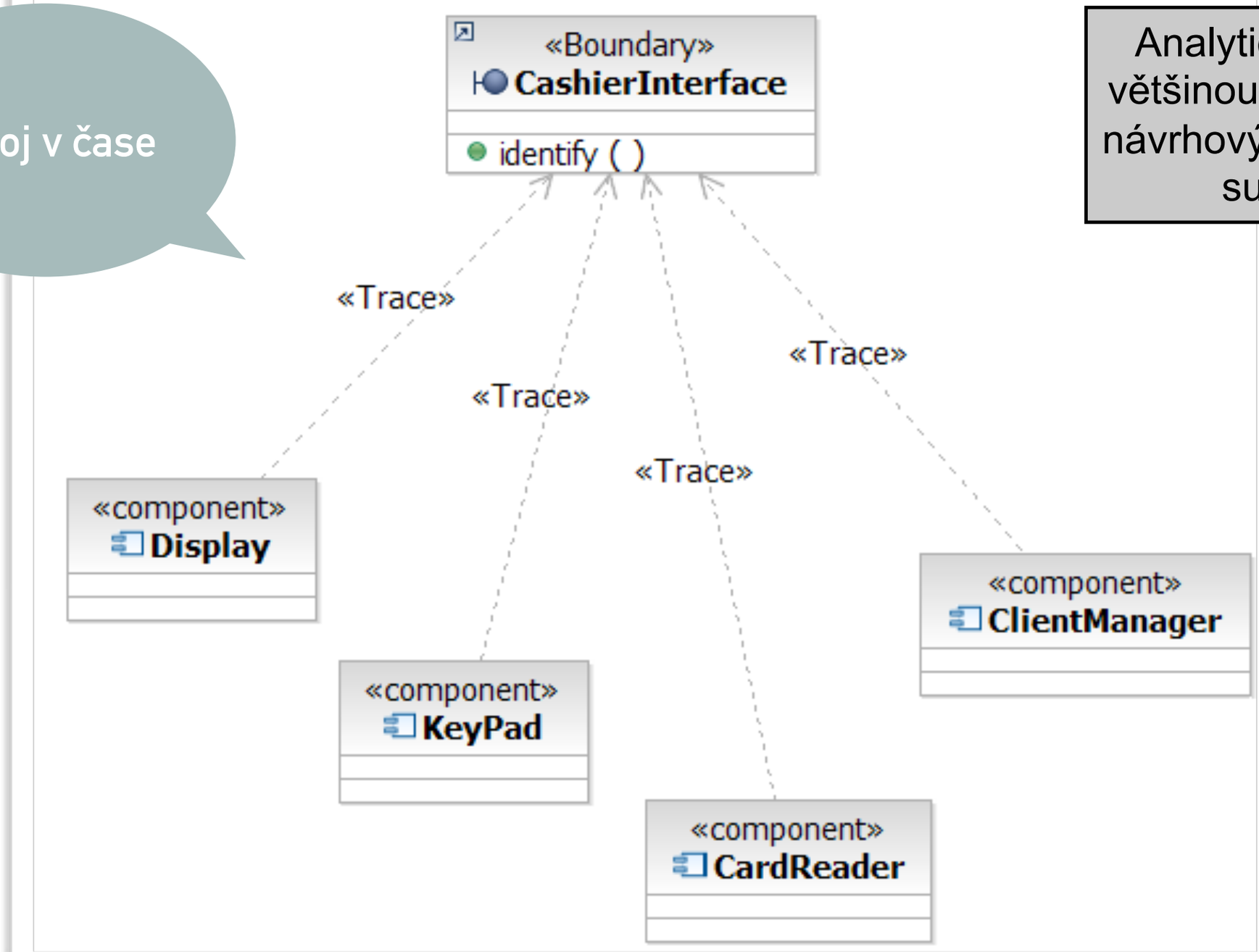
Sequence diagram pro Basic Flow scénáře - analytické třídy

ANALYTICKÉ ELEMENTY



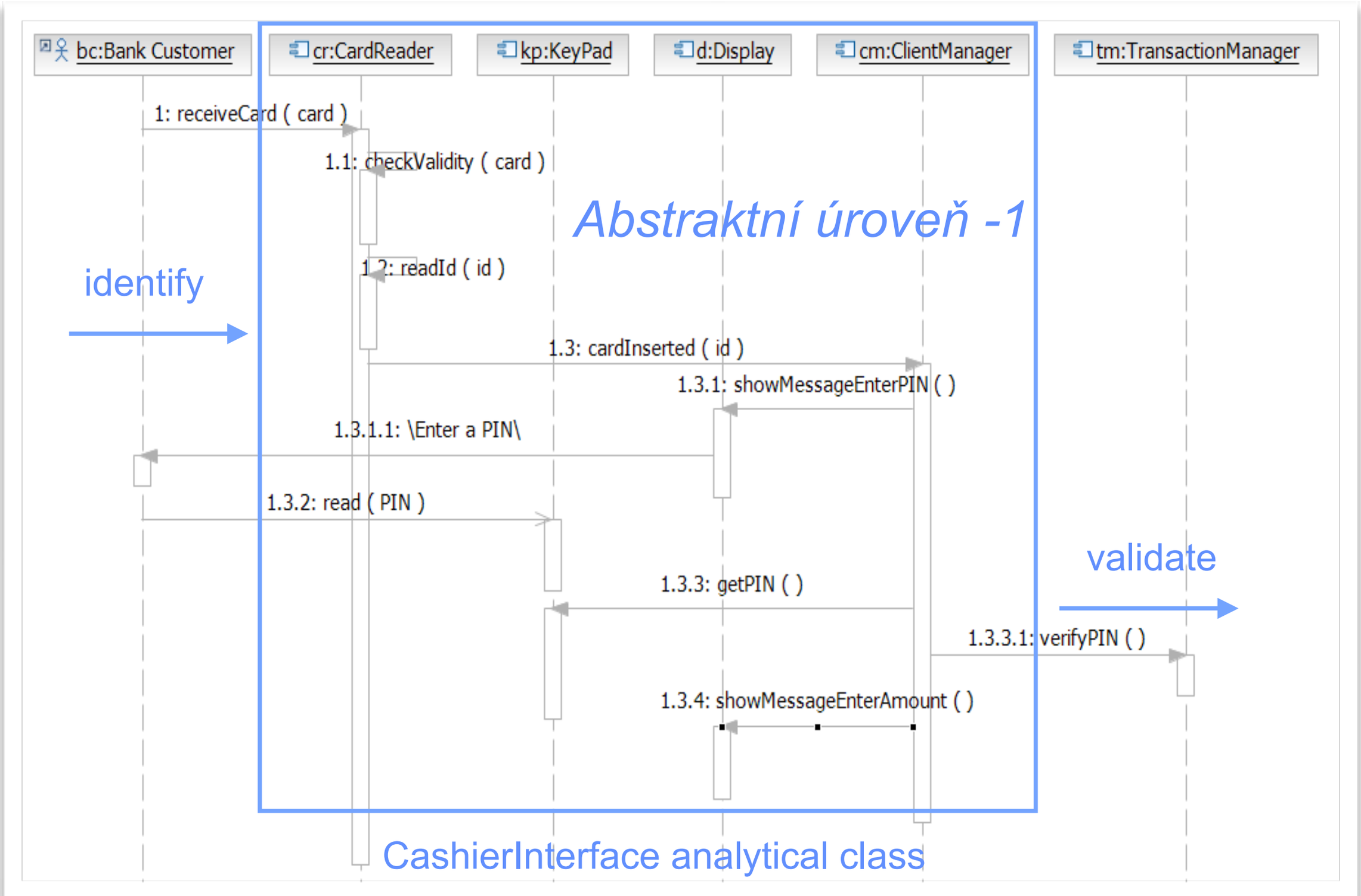
NÁVRH

Vývoj v čase

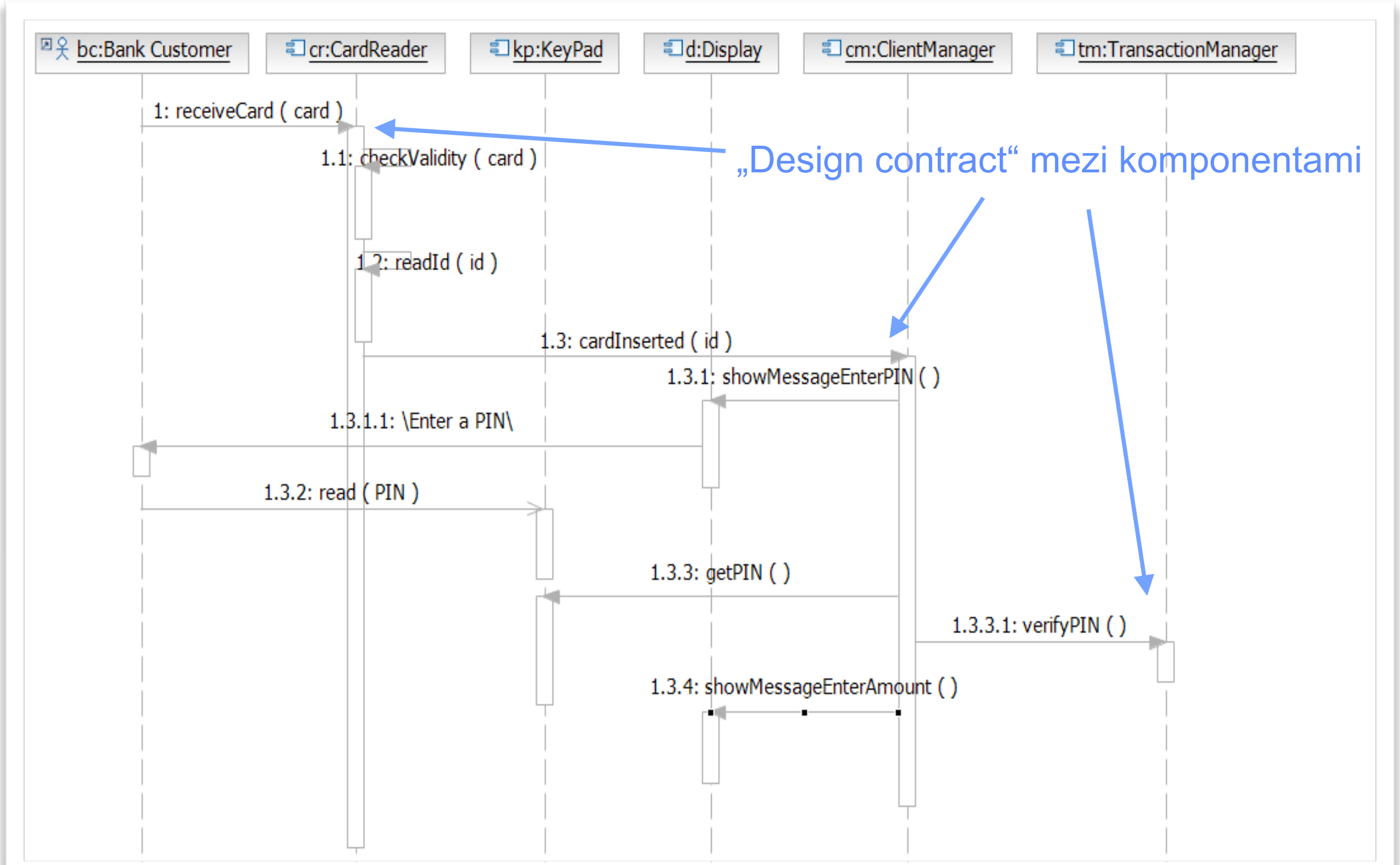


Analytické elementy se většinou vyvinou v několik návrhových elementů (tříd/ subsystémů).

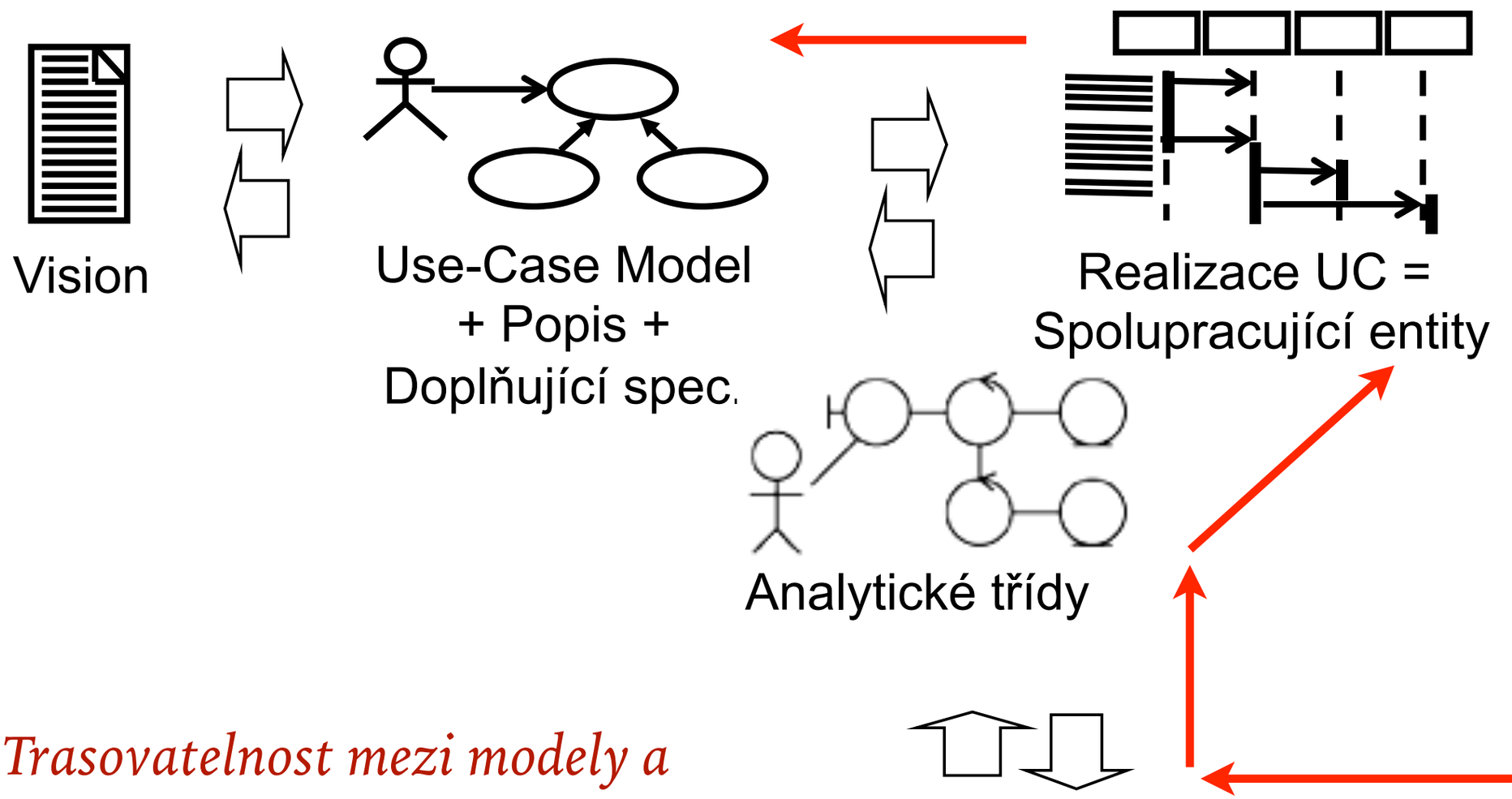
NÁVRH – SEKVENČNÍ DIAGRAM



NÁVRH – SEKVENČNÍ DIAGRAM



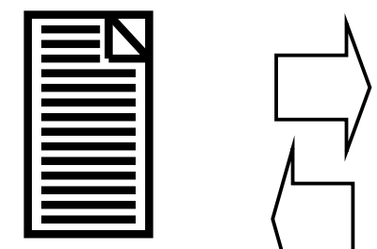
TRASOVATELNOST



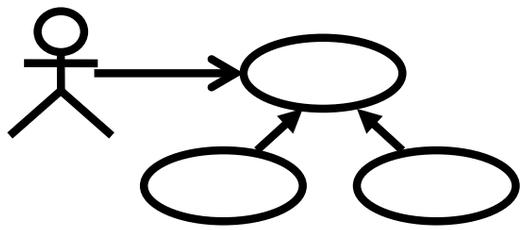
Trasovatelnost mezi modely a komponentami

```java
/* Block comment */
import java.util.Date;
/**
 * Doc comment here for SomeClass
 * @version 1.0
 */
public class SomeClass { // some comment
  private String field = "Hello World";
  private double unusedField = 12345.67890;
  private UnknownType anotherString = "AnotherString";
  public SomeClass() {
   //TODO: something
   int localVar = "IntelliJ"; // Error, incompatible types
   System.out.println(anotherString + field + localVar);
   long time = Date.parse("1.2.3"); // Method is deprecated
  }
}
```

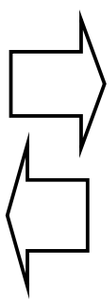
TESTOVACÍ SCÉNÁŘE



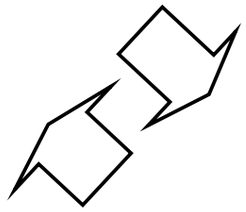
Vision



Use-Case Model
+ Popis +
Doplňující spec.



Test Cases



Test Scripts

Co jsou cíle?

Jaké jsou omezení?

Jaký je scénář?

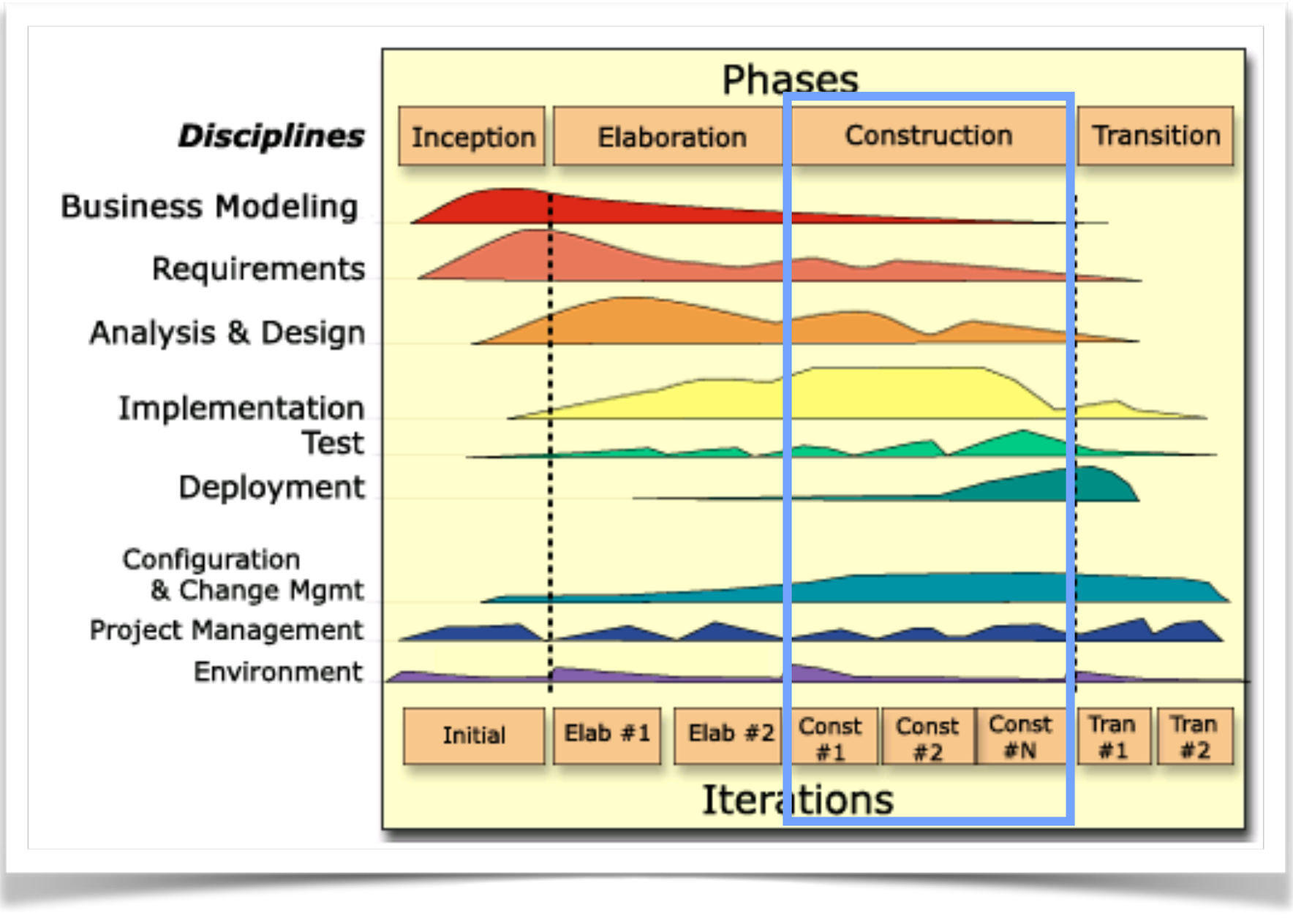
```
/* Block comment */
import java.util.Date;
/**
 * Doc comment here for SomeClass
 * @version 1.0
 */
public class SomeClass { // some comment
  private String field = "Hello World";
  private double unusedField = 12345.67890;
  private UnknownType anotherString = "AnotherString";
  public SomeClass() {
   //TODO: something
   int localVar = "IntelliJ"; // Error, incompatible types
   System.out.println(anotherString + field + localVar);
   long time = Date.parse("1.2.3"); // Method is deprecated
  }
}
```

TEST CASE

Use Case	Scénář	Data a podmínky		
		Obnos	Uživatel	Autorizace
Výběr hotovosti	Basic Flow	10000	Petr S	OK
Storno operace	AF1	-5000	Petr S	OK
…				
…				
…				

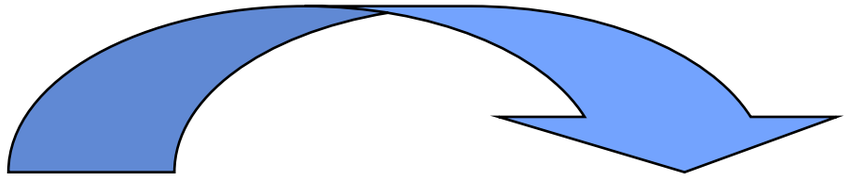
Use Case Driven: **reuse** popisů scénářů (jednotlivých kroků), počátečních a ukončujících podmínek, atd.

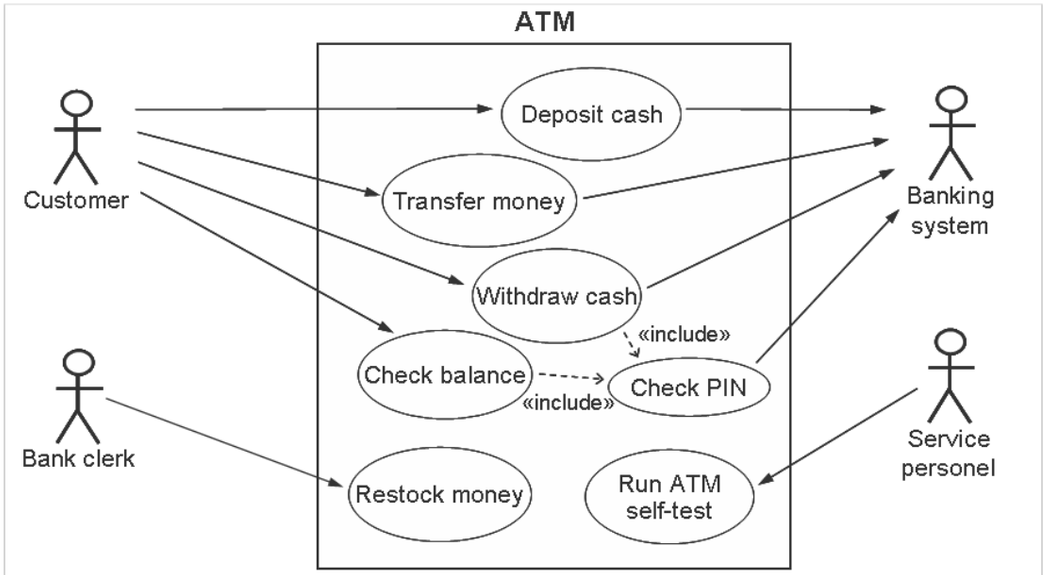
CONSTRUCTION

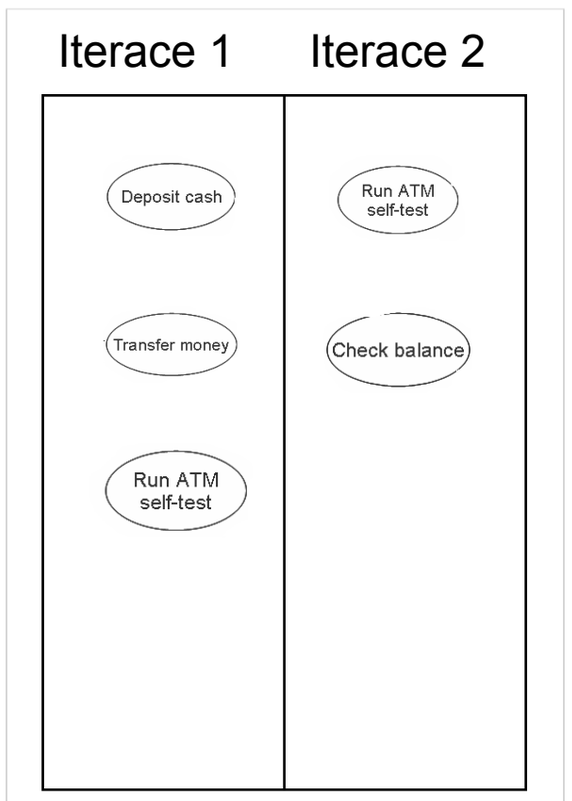


Construction – implementace zbylých Use Case + refaktoring + návrhové vzory

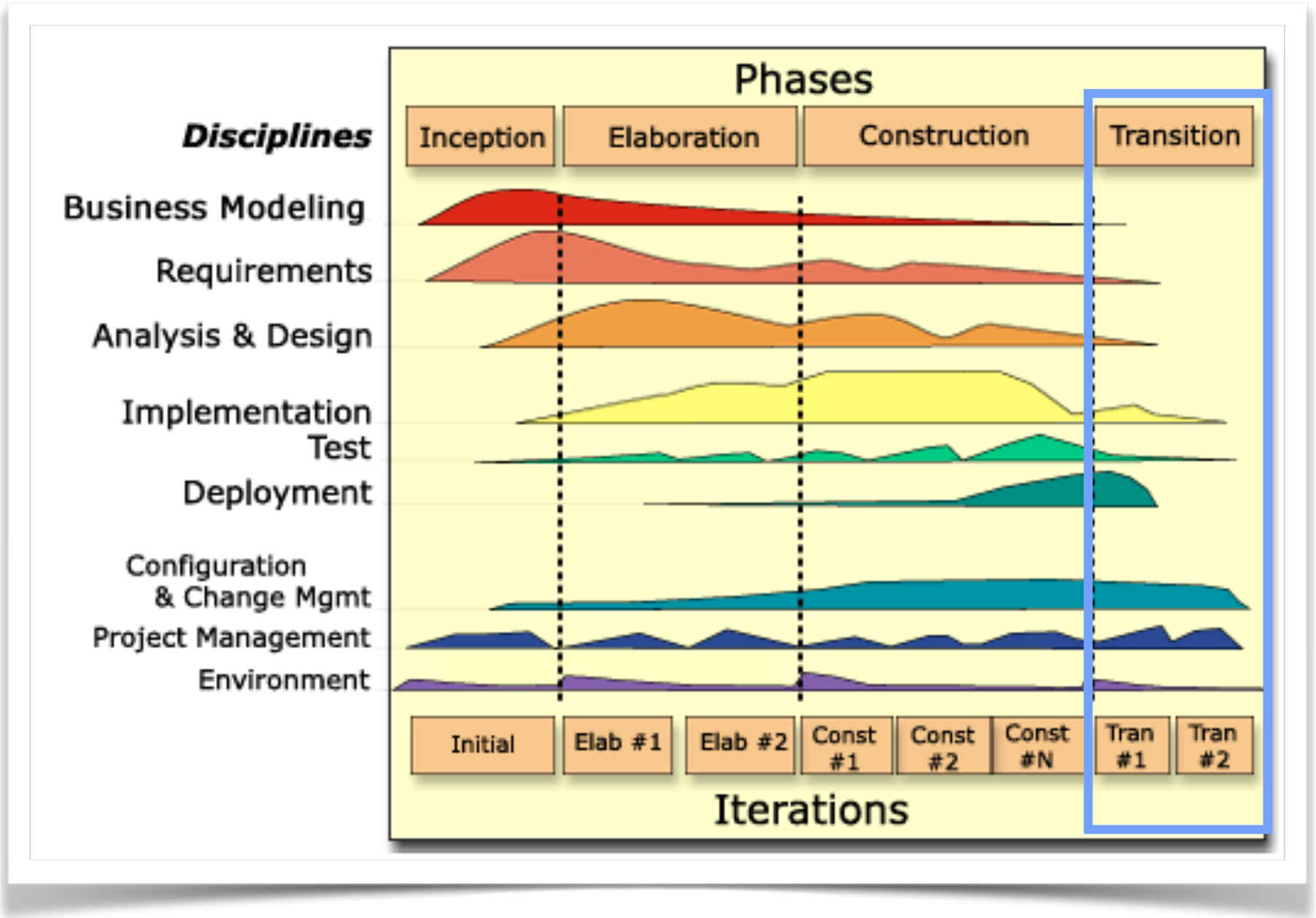
CONSTRUCTION







TRANSITION



Transition – nasazení systému + uživatelské akceptační testy