# Metody popisu systému, základy UML

## Strukturovaný přístup

Klasickou metodou analýzy a návrhu informačních systémů je **strukturovaný přístup**, navržený v 70. letech (Tom DeMarco, Ken Orr, Larry Constantine, Vaughn Frick, Ed Yourdon, Steven Ward, Peter Chen…).

Základy strukturovaného přístupu položil Tom deMarco v roce 1979 v práci „Strukturovaná analýza a specifikace systémů“. Mezi jeho základní doporučení patří:

- rozdělit systém na subsystémy;
- používat grafické znázornění (grafické modely) systému;
- před implementací vytvořit logický model systému.

Strukturovaný přístup modeluje systém pomocí následujících prostředků:

1. ERD – Entity Relationship Diagram
2. DFD – Data Flow Diagram
3. FSD – Function Structure Diagram
4. STD – State Diagram
5. DD – Data Dictionary
6. Structure Chart
7. Flow Diagram

Oproti objektově orientovanému přístupu k analýze a návrhu strukturovaná analýza odděluje datový model od modelu chování a funkcí. Oproti UML v objektově orientovaném modelování je tedy obtížnější udržet konzistenci návrhu. Na druhé straně ale datový model vyjádřený pomocí ERD dává rychlejší orientaci pro návrh tabulek v relační databázi.

## Filozofie strukturovaného návrhu

- Produkty analýzy musí být udržovatelné
- Velké problémy rozděleny na menší
- Použití grafického vyjádření
- Odlišení logické a fyzické úrovně
- Logický model má za cíl seznámení uživatele se systémem před jeho vytvořením a implementací
- Člení projekt na malé dobře definované aktivity
- Určuje posloupnost těchto aktivit a vzájemnou interakci
- Snaha vytvořit specifikaci, které rozumí uživatelé i návrháři

## Úrovně modelu systému z hlediska abstrakce

- **Konceptuální model** – rozpoznání základních datových objektů a jejich vztahů –návrh – co je obsahem systému, návrh je nezávislý na technologickém prostředí
- **Logický model** – relační schéma (včetně integritních omezení) – určuje jakou mají data strukturu, není zatížen konkrétní implementací. Může být vyjádřen formou ERD a DFD diagramů.
- **Fyzický datový model** – implementace v konkrétním databázovém produktu

Úrovně modelů ze strukturované analýzy odpovídají těmto oblastem:

ANALÝZA – DESIGN – IMPLEMENTACE

## Strukturovaný přístup – data a funkce

Strukturovaný přístup k analýze a návrhu pohlíží na informační systém ze dvou různých úhlů pohledu:

– z pohledu **dat**

– z pohledu **funkcí**

Každý má svoji specifickou logiku (ERD a DFD).

Abstrakce **část - celek (agregace).** Tato abstrakce se typicky používá ve funkčním modelu, kde se dělí systém na subsystémy, části subsystémů atd. Pro agregaci je typická principiální neomezenost dělení.

Abstrakce **specifický podtyp - obecný typ (generalizace)**. Tato abstrakce se typicky používá v datovém modelu, kde je možné uvažovat o jednotlivých specifických variantách nadřízeného pojmu (entity, objektu). Na rozdíl od agregace není nadřízený celek definován jako souhrn podřízených částí, ale jako nositel jejich společných vlastností (atributů).

Je důležité, že tyto dva základní typy abstrakce jsou vzájemně neslučitelné a tím tvoří jádro základního rozporu mezi funkčním a datových modelem.

Objektově orientované metody se pokoušejí tento rozpor překonat zapouzdřením obou - dat i funkcí - do jediného objektu.

## DFD (Data Flow Diagram)

Cílem DFD je modelování datových nebo řídicích toků v systému (v grafické podobě).

DFD diagramy popisují funkce systému.

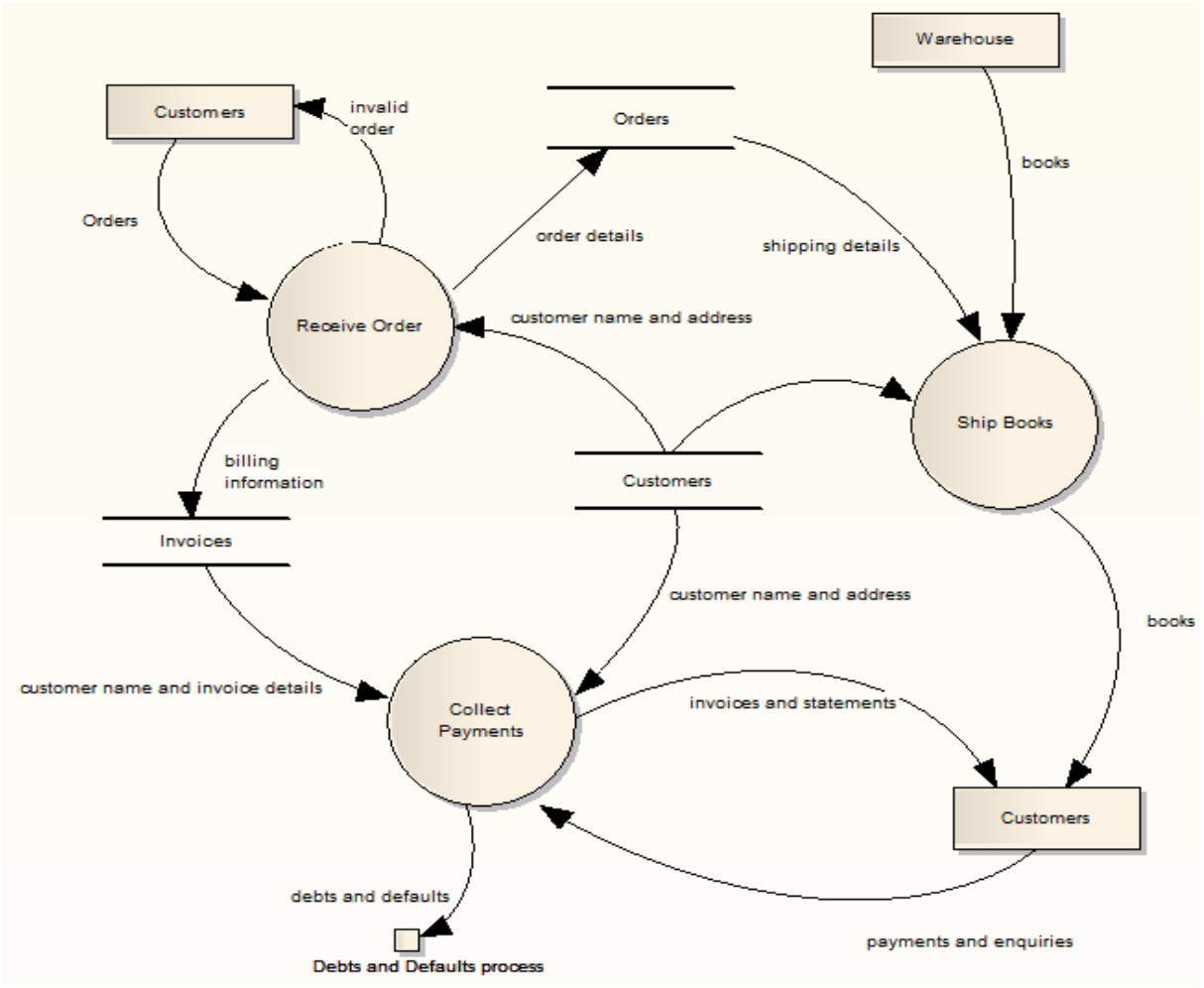
U DFD diagramů existuje několik forem záznamu, nejčastěji se používá notace dle DeMarca nebo Yourdana. Vycházejí z teorie grafů.

DFD se skládá z prvků „**proces**", „**datový sklad**", „**terminátor**". Datovým skladem v tomto pojetí rozumíme jakékoli úložiště dat.

DFD diagramy mají hierarchickou úroveň, procesy se dají postupně zjemňovat. Na nejvyšší úrovni stojí **kontextový diagram** – jedná se o speciální případ DFD diagramu, kdy je systém zobrazen jako jediný proces a naznačuje vztah systému s okolím. Na dalších úrovnícj jsou popisovány jednotlivé procesy a datové nebo řídicí toky.

**Dekompozice DFD** na nižší úrovně až na úroveň základních elementárních funkcí:

- Nesmí existovat proces, který nemá žádné vstupy a přesto produkuje datové toky
- Nesmí existovat proces, který pouze spotřebovává data a nemá výstupy




Obr. 1.1 Příklad DFD diagramu

## FSD (Function State Diagram)

Účelem FSD je zobrazit dekompozici systému na funkční celky (subsystémy), zdokumentovat funkční hierarchii systému a poskytnout pohled na vyvíjený systém se zaměřením na jeho hierarchickou strukturu.

FSD diagram se skládá z

- funkce (proces, systém)
- vazba

Funkce dělíme na:

- procesní
- dialogové
- řídicí

Rozdíl mezi DFD a FSD je následující:

- FSD je zaměřen na hierarchickou strukturu subsystémů, jde o statický pohled
- DFD je zaměřen na datové toky a procesy transformující vstup na výstup, jde o dynamický pohled na systém

## STD (State Transition Diagram)

Stavový diagram (STD) obsahuje sled stavů, v jakém se systém (nebo jeho část) může nacházet a za jakých podmínek může dojít ke změně stavu. Modeluje časově závislé chování systému.

- Důležitý z hlediska pochopení logiky systému
- Stavy jsou statické, změna stavu je většinou důsledek nějaké události.
- Vyjádření – např. vývojové diagramy
- Může být hierarchický.

STD diagram musí mít jeden počáteční bod a jeden nebo více koncových bodů. Samotný stav je vnímán jako statický. Diagram eviduje podmínku, při které systém přejde z jednoho stavu do druhého.

## Data Dictionary (datový slovník)

Datový slovník slouží k formalizovanému popisu dat systému z pohledu uživatele.

Metadata (data o datech):

- Spravuje databázový stroj
- Přístup jen ke čtení

Vytvoříme-li datový slovník, který obsahuje všechny položky databáze, dostaneme informaci, zda se některé položky nevyskytují v databázi vícenásobně. Tím pádem můžeme upravit datový model a snížit redundanci.

## Structure chart

Slouží pro znázornění hierarchie programových modulů systému, ve tvaru grafu (stromu). Kořenem je hlavní programový modul, uzly znázorňují dílčí volané moduly.

## Flow chart (vývojový diagram)

Zachycuje algoritmus. Alternativní vyjádření algoritmu, vhodnější pro strukturované programování je strukturogram. Pro modelování systému má Flow Chart nejmenší důležitost, lze použít pro popisy funkcí.

## Návrh informačního systému

Výše uvedené diagramy slouží pro návrh informačního systému. Návrh může být dále doplněn **případy užití**, modelem spolupráce nebo funkčním modelem. Důležitá je také volba architektury, dnes převažuje architektura **třívrstvá** (prezentační, funkční a datová vrstva).

## Metodiky

- Logické modelování Gane-Sarson
- Datově orientovaný přístup (Warnier/Orr)
- Entitně relační model – Chen
- Relační datový model - Codd
- Yourdanova strukturovaná analýza
- SSADM

## Yourdonova moderní strukturovaná analýza (YMSA)

Edward Yourdon vytvořil konceptuální popis systému, skládající se ze tří částí:

a) Datový model
b) Model chování
c) Model řízení

Yourdon doporučuje pro modelování nástroj následujících vlastností:

a) Musí být grafický
b) Podpora TOP-DOWN přístupu
c) Minimalizace redundance – řešení pomocí Data Dictionary
d) Snadná čitelnost
e) Předvídat chování systému

Jádro metodiky (1989) spočívá v nalezení **esenciálního modelu**, který vyjadřuje podstatu systému, je dlouhodobě stabilní a je nezávislý na použité technologii a implementaci. Z esenciálního modelu je následně odvozen implementační model.

Esenciální model se skládá ze dvou částí:

**a) Model okolí**
**b) Model chování**
**c) Implementační model**

Model okolí se skládá z:

a) Dokument o účelu systému
b) Kontextový diagram
c) Seznam událostí

V kontextovém diagramu je celý systém znázorněn jako jeden proces. Cílem je zachytit výměnu informací s okolím, tedy hlavně s uživateli systému.

Model okolí a zejména dokument o účelu systému je určen pro zákazníka a management. Model chování naproti tomu je vytvářen pro návrháře a obsahuje popis chování uvnitř systému. Pro model chování se používají DFD, ERD a DD, vytváří se hierarchická struktura DDF, která se následně vyvažuje. V posledním kroku se doplní o STD diagramy a **minispecifikaci** a dokončení datového slovníku.

Po dokončení modelu chování může být vytvořen implementační model. Zde se určí, které procesy budou automatizovány a které budou manuální (v modelu vystupují jako terminátory a v reálu reprezentují uživatele, který procesy manuálně provádí).

## SSADM – Structured System Analysis and Design Method

Vyvinuta firmou LBMS, velmi rozšířená ve Velké Británii, kde se stala standardem. Vývoj software je rozdělen do šesti následujících etap:

- analýza stávajícího systému,
- specifikace požadavků,
- výběr technických možností,
- návrh logických dat,
- návrh logických procesů,
- fyzický návrh.

Hlavními nástroji pro modelování systému jsou diagramy datových toků, logické datové struktury (LDS) a životní cykly entit (ELH). Pro DFD používá odlišnou notaci.

## Příklady aplikací pro podporu strukturální analýzy

- Microsoft Visio
- Dia - http://live.gnome.org/Dia
- CASE Studio Toad Modeler - http://www.casestudio.com/enu/default.aspx
- Visual Paradigm - http://www.visual-paradigm.com/
- Edge Diagrammer - http://www.pacestar.com/

# Objektový přístup

Objektově orientovaný přístup je založen na objektech.

Objekt je struktura, která má definované

- **Vlastnosti** (tomu odpovídají atributy objektu)
- **Chování** (tomu odpovídají funkce, pro které se používá termín „metody“)

Vlastnosti a chování je zapouzdřené v jednotlivých objektech. Každý objekt je schopen reagovat na události.

Informační systém z pohledu objektově orientovaného přístupu je chápán jako množina spolupracujících objektů.

Objektový přístup lépe odpovídá chování reálného světa.

V oblasti programování je hlavní ideou objektového přístupu **znovupoužitelnost.**

## Základní myšlenky objektového přístupu

- **Zapouzdření (encapsulation) -** objekt je pro nás „černou skříňkou – zajímá nás, co dělá a ne z čeho se skládá.
- **Dědičnost** - možnost vytvářet nové instance objektů s možností přidat nové prvky. Opakem dědičnosti je **generalizace** – z konkrétních dílčích částí sestavujeme obecnou společnou část
- **Polymorfismus (vícetvarost)** – různé chování objektů na stejný podnět – metoda stejného jména může mít trochu jinou funkcionalitu, přestože je tato funkcionalita pojmově blízká. Analogie je pojem „otevřít“ a rozdíl mezi funkcí „otevřít dveře“ a „otevřít láhev“. Příkladem polymorfismu je například funkce Read()/Write() v operačním systému – podle situace lze uplatnit na soubor nebo na nějaké zařízení.
- **Genericita** – možnost vytvářet parametrizovatelné programové moduly

## Srovnání relačního a objektového modelu

**Relační model** – prvky reálného světa se snažíme zobrazit do pevných předem připravených struktur

**Objektově orientovaný model** – pro prvky reálného světa vytváříme objekty, které se jim podobají

## Shrnutí objektového přístupu

Chceme-li vyvinout návrh systému od koncepce k podrobnému objektově orientovanému návrhu, musíme provést tyto kroky:

1. Pochopit a definovat kontext a externí interakce se systémem
2. Navrhnout systémovou architekturu
3. Identifikovat základní objekty v systému

4. Vyvinout modely návrhu
5. Specifikovat rozhraní

## UML

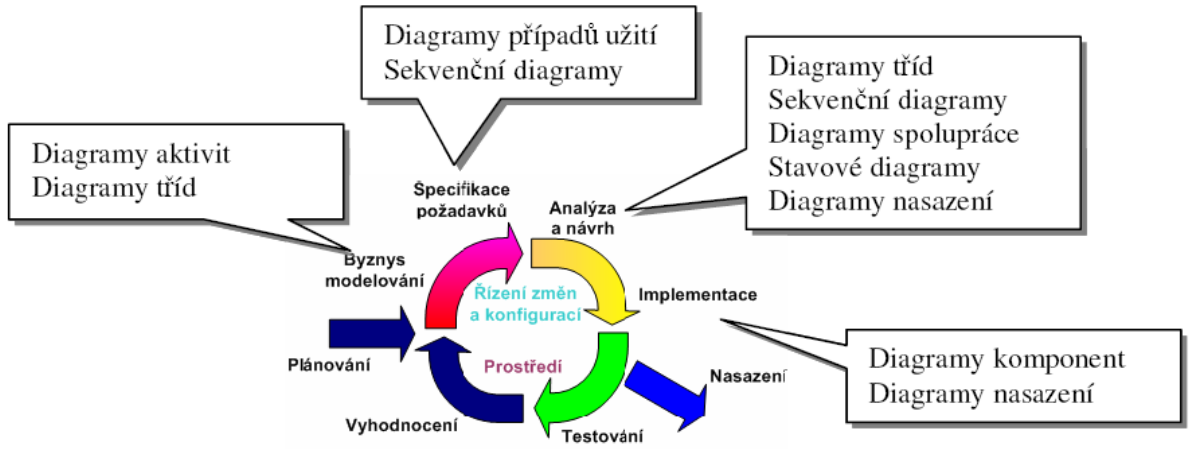
UML neboli **Unified Modeling Language** je nástroj pro modelování informačních systémů založený na objektově orientovaném přístupu.

Je přijat sdružením OMG (Object Management Group) jako standard pro tvorbu informačních systémů.

Jedná se o nejrozšířenější objektovou notaci pro modelování systémů, která je podporována všemi CASE nástroji.

Jednou z důležitých charakteristik UML je jeho nezávislost na metodologiích.

Použití UML je široké – od prostředku pro obecný popis systému po detailní návrh, který lze využit pro generování kódu v objektově orientovaném programovacím jazyce.




UML nabízí k popisu systému mnoho typů diagramů, rozdělených do tří skupin. První skupina diagramů popisuje statickou strukturu aplikace. Druhá skupina popisuje různé aspekty dynamického chování. Třetí slouží k organizaci a správě aplikačních modulů.

| **Statická struktura** | **Dynamické chování** | **Správa modulů** |
|---|---|---|
| diagram tříd (Class Diagram) | Use case diagram | balíčky (Packages) |
| objektový diagram (Object Diagram) | sekvenční diagram (Sequence Digram) | subsystémy (Subsystems) |
| komponentový diagram (Component Diagram) | diagram činností (aktivit) (Activity Diagram) | modely (Models) |
| diagram nasazení (Deployment Diagram) | diagram spolupráce (Collaboration Diagram) | |
| | stavový diagram (Statechart Diagram) | |

Diagram komponent a diagram nasazení reprezentují **implementační model**.

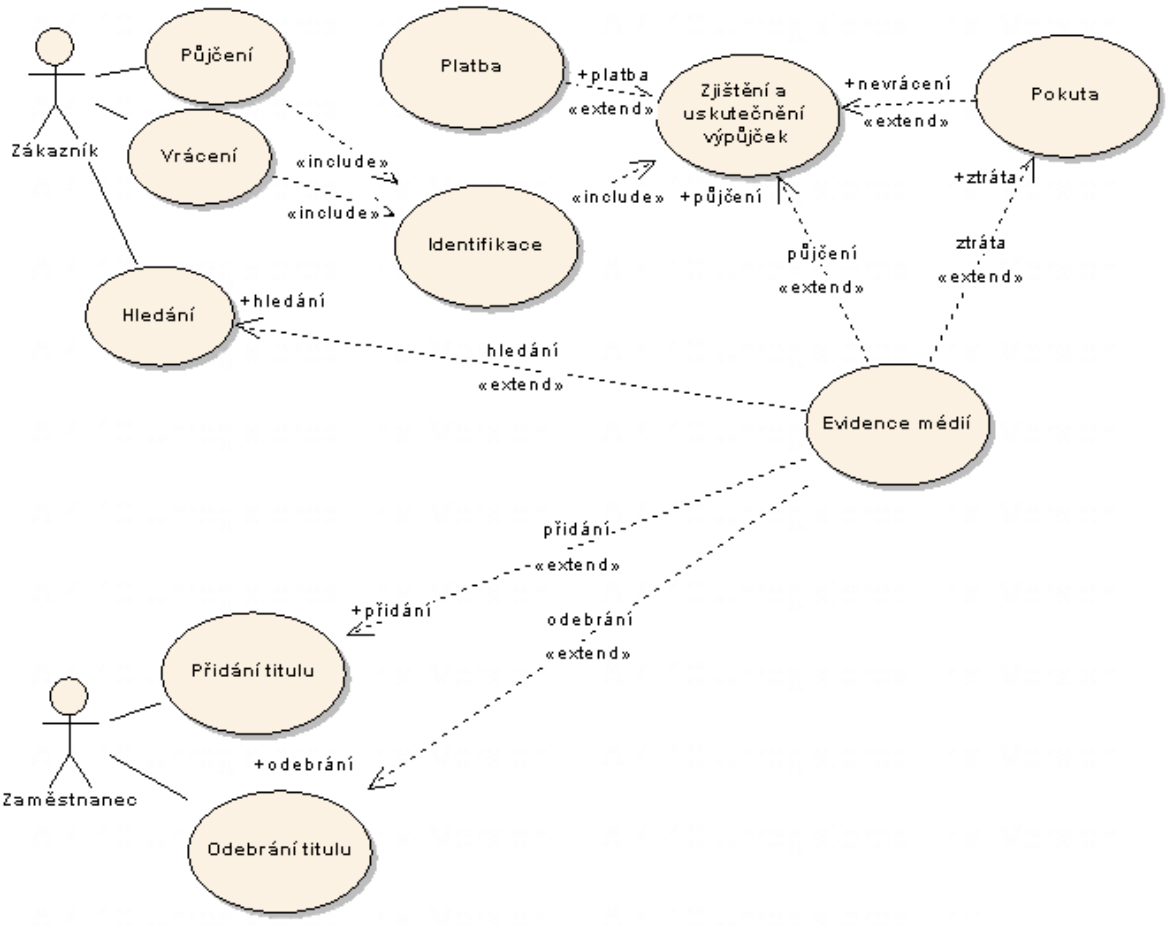
*UML **nezahrnuje DFD diagramy** (Data Flow Diagramy), které slouží v strukturované analýze k popisu chování systému. Datové toky a jiné typy diagramů, které nebyly do UML zahrnuty, nezapadají čistě do konsistentního objektově orientovaného paradigmatu. Diagramy aktivit a diagramy spolupráce splňují mnoho z toho, co lidé chtějí od DFD. Diagramy aktivit jsou zároveň vhodné pro modelování workflow.*

## Use Case

Diagram, který popisuje užití systému, jeho funkce.

Definice požadavků na funkcionalitu.

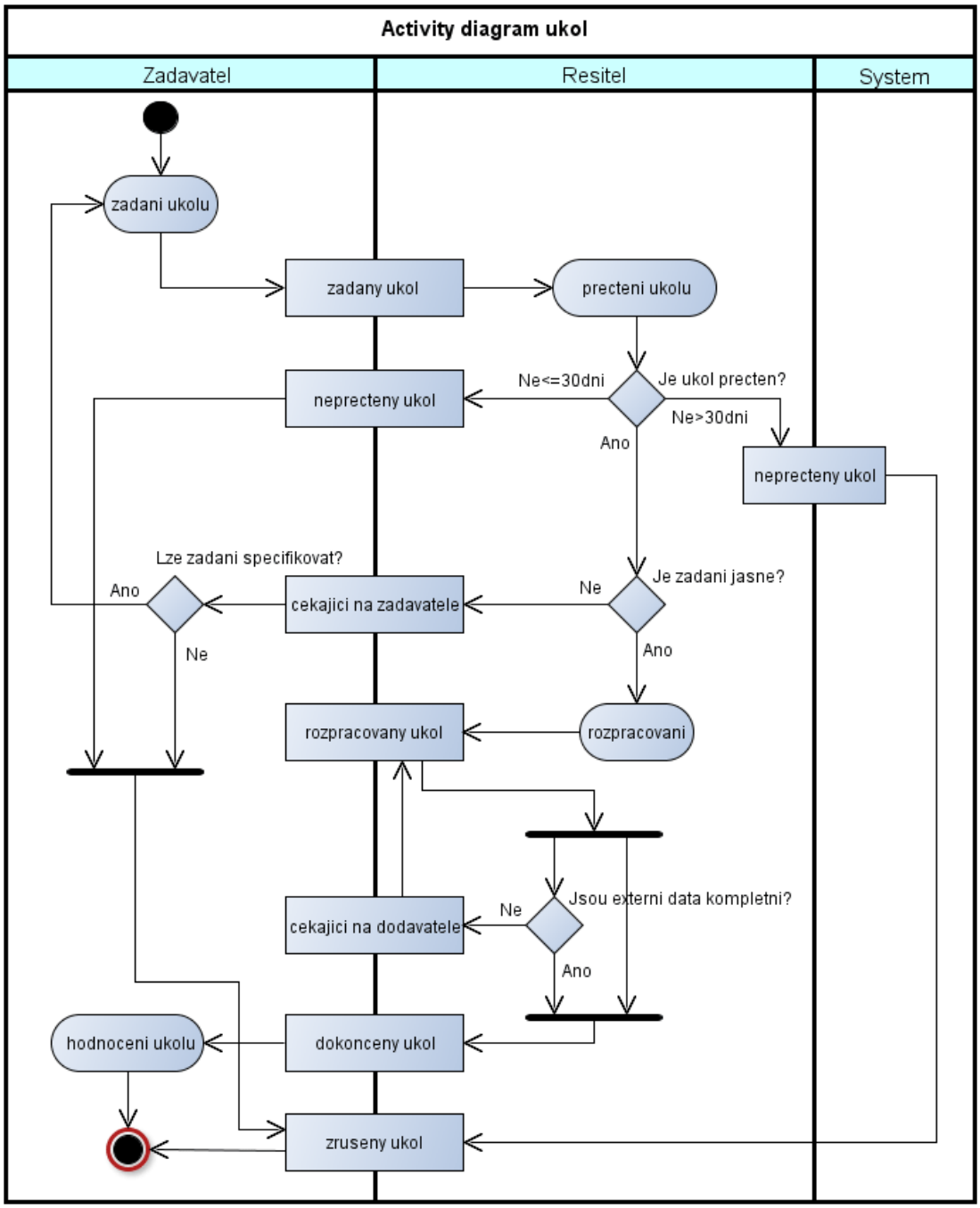
- Vysvětlení činnosti **aktorů** (účastníků systému – osoba, jiný IS, HW komponenta…)
- Vymezení hranic systému
- Dynamický pohled na vyvíjený systém z pohledu zákazníka (uživatele)




Obr. 1.3 Ukázka Use case diagramu

## Diagram aktivit (Activity Diagram)

- Odvozen z FlowCharts nebo WorkFlow
- Modelování business procesů, pracovních postupů, procedurální logiky
- Objektově orientovaný vývojový diagram
- Zachycuje posloupnost činností objektů - sekvenčně nebo paralelně




Obr. 1.4 Diagram aktivit

Prvky aktivity diagramu:

- Akce - obdélník s oblými rohy + název
- Zahájení a ukončení
- Návaznost akcí – šipkou od zdrojové k cílové
- Rozhodnutí a sloučení (Decision and merge)
- Větvení a spojení (fork and join)
- Plavecké dráhy (oddíly aktivit; swimlines)

## Nedostatky UML

Za nedostatek UML můžeme považovat neschopnost UML poskytovat prostředky pro návrh uživatelského rozhraní a datových modelů. Například Activity diagram v UML neumožňuje zachytit místo pro uložení informací, podobně jako to umí diagram DFD ve strukturovaném návrhu.

Nedostatečností UML v oblasti datového modelování je to, že UML neposkytuje vhodné prostředky pro datové modelování, a proto se navrhuje rozšíření notace UML pomocí vhodných *stereotypů*, což jsou prostředky jazyka UML pro jeho vlastní rozšiřování. Soubor určitých *stereotypů*, které rozšiřují UML pro určitou oblast užití, se nazývá *UML extensions*.