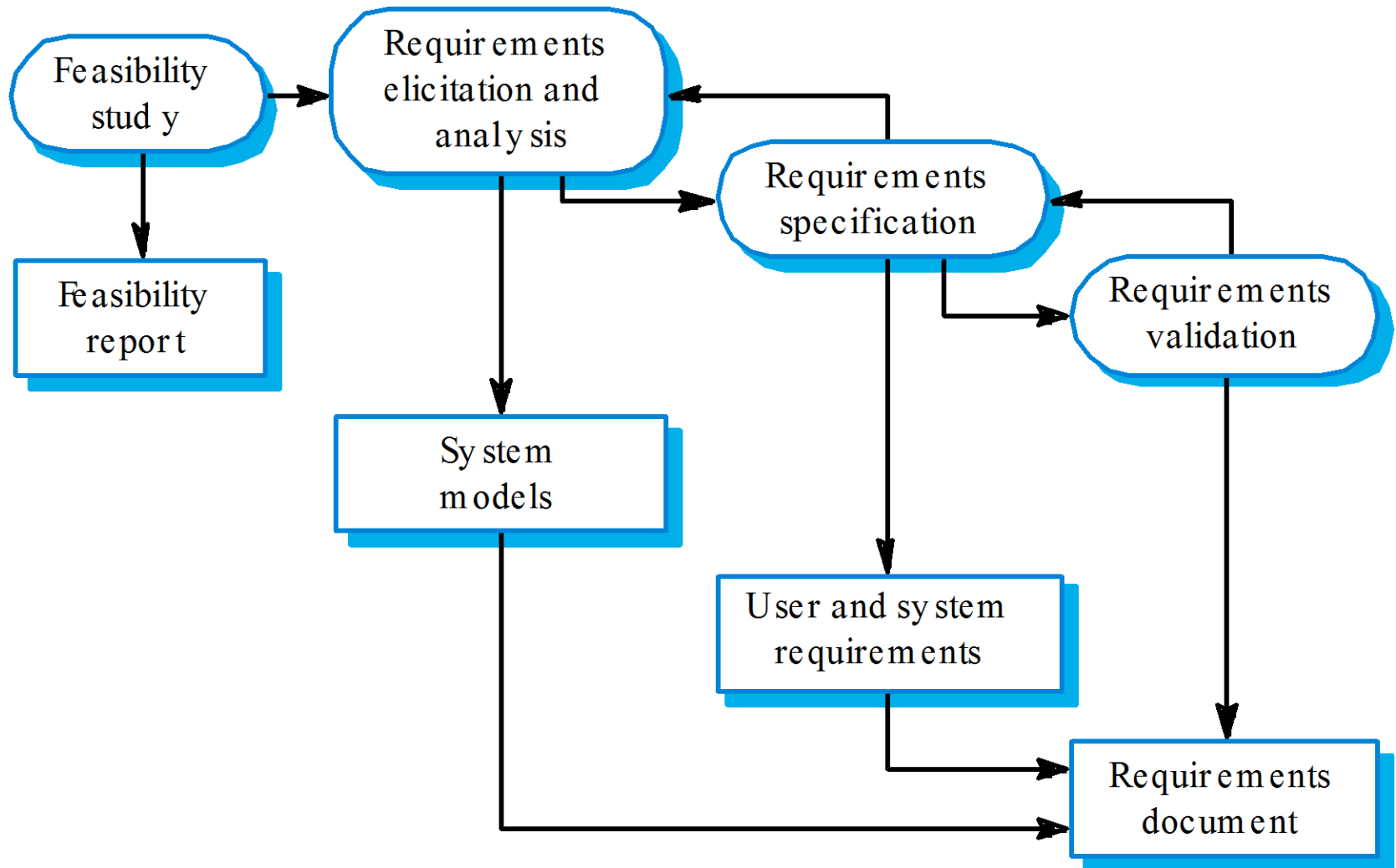


Специфициране на изискванията

Лекция 6

Процес на Инженеринг на изискванията



Съдържание

- Роля на метода на инженеринга на изискванията
- Документиране на изискванията на естествен език
- Специфициране на изискванията чрез аналитични модели:
 - Модели на функциите / процесите
 - Модели на поведението
 - Семантични модели на данните, Обектно ориентирани методи за анализ
 - Формални модели при ИИ

Методи за инженеринг на изискванията (ИИ)

- Процесът на инженеринг на изискванията обикновено се ръководи от *метод* на инженеринг на изискванията.
- **Методът на ИИ е систематичен подход за документиране и анализиране на изискванията на системата.**
- Нотацията, която осигурява средство за изразяване на изискванията, е свързана с метода.

Необходимите характеристики на метода

- Да подпомага формулирането на изисквания на всички етапи (запис, структуриране, анализ, решения, перспективи и връзки).
- *Точност на описанията* чрез избраната нотация с *възможност за проверка* за последователност и коректност.
- Подходящ запис на изискванията, удобен за *споразумение* с крайните потребители и клиенти: например, *интегриране на формални и неформални описания.*
- Възможност да се *дефинира* (моделира) *външния* свят.
- Възможност за *отразяване на промените* без да е необходимо да се преработи целия набор от изисквания.
- Възможности за *интегриране* на други (различни) методи (техники за моделиране) с цел пълнота на описанието.
- Да осигурява възможност за *комуникация* на хората (заради идеите и получаване на обратна връзка).
- Използване на *софтуерен инструмент* – начин да се управлява сложността.

- Методите използват различни *техники за моделиране*, за да формулират системните изисквания.
- Системните изисквания могат да бъдат обогатени чрез моделиране на *различни аспекти* с използването на *различни модели*.
- Примери за :
 - Модел на потока на данните (Data flow model)
 - Композиционен модел (Compositional model): *np.* (Entity relationship)
 - Модел на класовете (Classification model) Пр.: Object diagrams
 - Модел на поведението (Stimulus-response model)
 - Модел на процесите (Process model)

Специфициране на изискванията (Requirements specification)

Дефиниция: Процесът на записване на потребителските и системните изисквания в документ за изисквания.

- Изискванията на потребителите трябва да бъдат разбираеми от крайните потребители и клиентите, които нямат техническа подготовка.
- Системните изисквания са по-подробни изисквания и могат да включват повече техническа информация.
- Системните изисквания могат да бъдат част от договор за разработване на системата
 - Затова е важно те да са възможно най-пълни.

Аспекти при документирането - 1

- Независимо от използвания език и метод на работа **три аспекта** трябва да бъдат документирани:
- **Функционалност:** описва се с **трите перспективи** на системата
 - **Данни:** използване и структура
 - **Функции:** условия, обработка, резултати
 - **Поведение:** Динамиката на проявление
- *Нормалните случаи, както и изключенията* трябва да бъдат представени.

Аспекти при документирането - 2

- **Аспект:** качество на софтуерната система

- Начин на представяне на системата

- Обем на данните
- Време за реакция
- Скорост; производителност

...

Специфициране на стойности на метрики

- „-ilities“ като
 - Използваемост
 - Надеждност
 - Други

Аспекти при документирането - 3

- **Ограниченията**

- **Технически:** протоколи, интерфейси, среда за разработка ...
- **Правни:** закони, стандарти, нормативи
- **Културни, етични**
- **Околна среда:** физически закони
- **Решения / ограничения,** изисквани от ключови ЗЛ.

Как да документираме - начини

Представяне, ориентирано към решения (Documenting Solution-oriented Requirements):

Неформално

- Естествен език

Полуформално

- Структурни модели
- Модели на взаимодействието



Диаграми, обогатени с текст.

Формално

- Формални модели



Спецификация на естествения език

- Изискванията са написани като *естествени езикови изкази*
- Могат да бъдат допълнени с диаграми и таблици
- Изразителен, интуитивен и универсален начин
- Разбираем за клиентите

Указания за писане на изисквания

- *Стандартен* формат, използван за всички изисквания
- Изказ, представен по последователен начин:
 - за задължителни изисквания,
 - за желаните изисквания.
- *Маркиране* на текст, за да идентифицирате ключови части от изискването
- *Избягване* на използването на (компютърен) *жаргон*
- Включване на *обосновка* защо е необходимо изискването

Документиране на естествен език: запис на изисквания, насочени към софтуерно решение

Пример:

R15: If a glass break detector at the window detects that the pane has been damaged, the system shall inform the security service.

Трите перспективи:

(Структура на) данни: glass break detector, pane, information to security service

Функция: detects, inform

Поведение: If ...damaged, then inform ..

Проблеми при специфицирането на естествения език

- **Липса на яснота и на точност**
 - няколкократно проверки на документа
- **Смесване на изисквания**
 - Функционалните и нефункционалните изисквания са склонни да бъдат смесвани.
- **Сливане на изискванията**
 - Няколко различни изисквания могат да бъдат изразени заедно.
- **Справяне с проблемите – шаблон за писане ...->**

[Rupp et al. 2009
ISO/IEC/IEEE 29148:2011]

Phrase templates

Use **templates** for creating **well-formed** natural language requirements

Typical template:

[<Condition>] <Subject> <Action> <Objects> [<Restriction>]

Example:

When a valid card is sensed, the system shall send
the command 'unlock_for_a_single_turn' to the turnstile
within 100 ms.

Пример: A structured specification of a requirement for an insulin pump

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

Пример: A structured specification of a requirement for an insulin pump

Action

CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requirements

Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition

The insulin reservoir contains at least the maximum allowed single dose of insulin.

Post-condition r0 is replaced by r1 then r1 is replaced by r2.

Side effects None.

Таблична спецификация

- За допълване на описание на естествения език.
- Полза - при дефиниране на възможни алтернативни начини на действие.

Пример:

The insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Документиране с използване на модели на системата

- Data and object modelling
- Behaviour modelling
- Function and process modelling
- User interaction modelling
- Goal modelling
- UML



- Моделите се използват главно да представят *функционалните изисквания*, докато нефункционалните се специфицират на естествен език.

Структурно описание на системата: Модел

Какво е модел?

Има ли общоприета дефиниция на модел?

All models are wrong – some are useful.

George E. P. Box

Модел

Моделът е абстракция, опростено изображение на обекта на изследване във форма, различна от реалното му съществуване.

Моделът отразява или възпроизвежда обекта на изследването в *достатъчна степен*, за да позволи получаване на информация за неговото изучаване.

Модели на софтуерна системата: концептуални модели

Какво е модел?

Защо и кога се използват при ИИ?

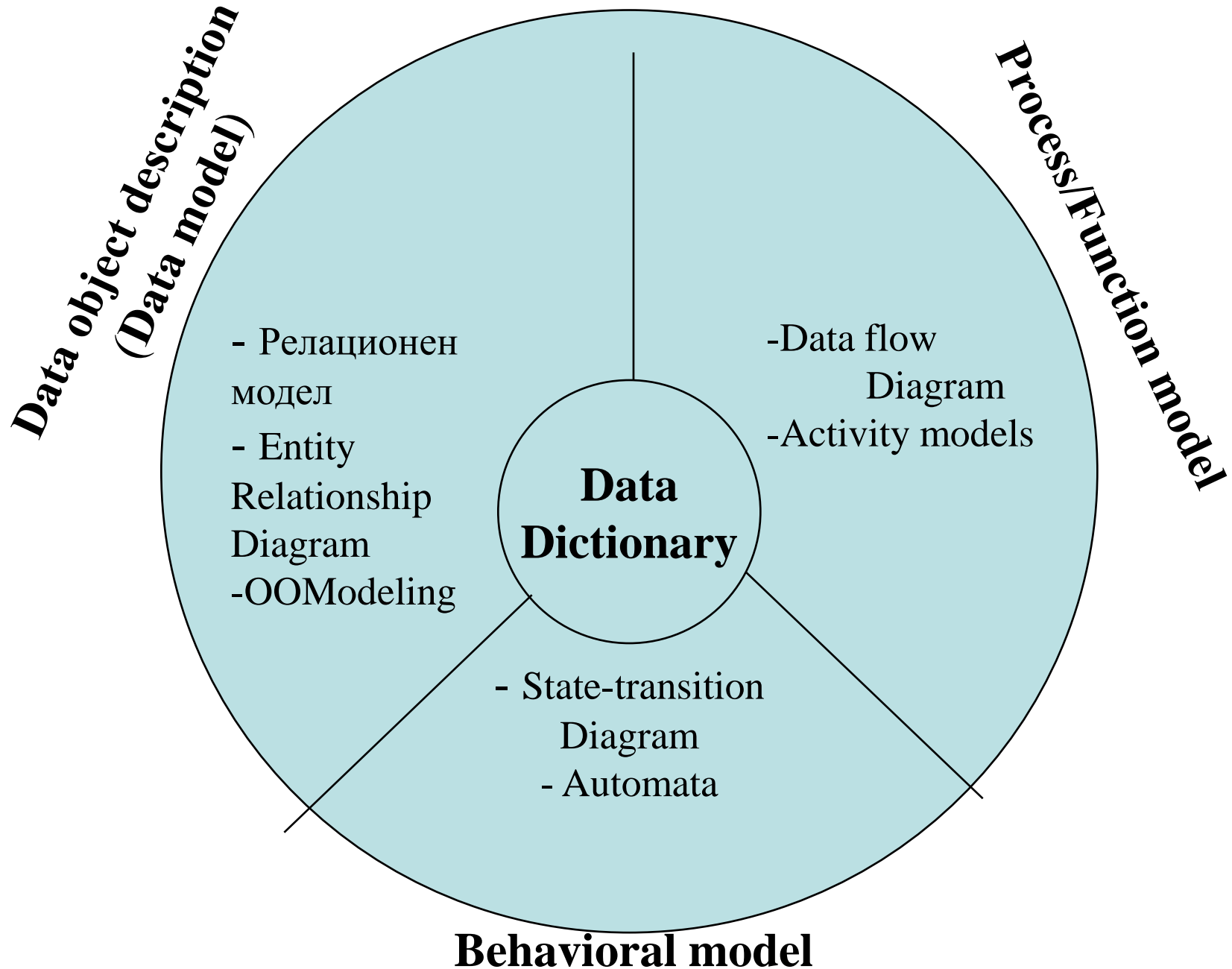
Моделите на системата са (важен) *мост* между анализа на дадена система и проектирането ѝ.

- Моделите са графични или математически представяния.
- Моделите на системата се основават на (*изчислителни*) *концепции* (*computational concepts*) като обекти и функции.

Перспективи на софтуерната система:

- **Перспективи на решението:**
 - **Функции**
 - **Поведение**
 - **Данни (статична структура)**
- **Връзка между трите перспективи**

Модели на различните перспективи



Модели според определена *перспектива* на системата

1. Модели на функциите, процесите - как се обработват/преработват данните в системата

Пример: **Модел на потока на данните (Data-flow models (DFM))**

2. Модели на поведението (Stimulus-response/behavioral models) - модели на състоянията на системата

Примери: **Диаграми на преходите** (state machine models, state transition diagrams); **Спецификация на управлението** (control specification)

3. Модели на данните - как някои същности (entities) на системата се изграждат от други; Как същностите имат общи/еднакви атрибути и имат общи характеристики

Пример: **Обектно описание и композиционни модели (Object description and composition models)**

Кои са недостатъците на методите на структурен анализ?

Какво друго може да се моделира? (виж Тема 4)

- **Взаимодействието на потребителя със системата**
 - Модели на контекста
 - Случаи на употреба
 - Диаграма на последователностите
- **Описание на целите**
 - Дърво на целите

Перспектива на функциите:

**Модели на функциите и на
Модели потока на действията**

Перспектива на функциите:

Модели на функциите и на потока на действията

1. Модел на потока на данните Data Flow Models (DFM)

- DFM представя потока на информация в рамките на системата, както и между системата и средата.
Той *не показва* последователност на поведение и на управляваща информация.
- Диаграми (модел) на потока на данните е графично представяне на:
 - външни за процеса елементи
 - процеси (функции)
 - поток на данните
 - хранилища за данни
- Връзките показват постепенното преобразуване на данните.

DFD елементи

External Entity (външни същности) - източник или получател на данни/информация. Те обикновено са извън обхвата на изучаваната област.

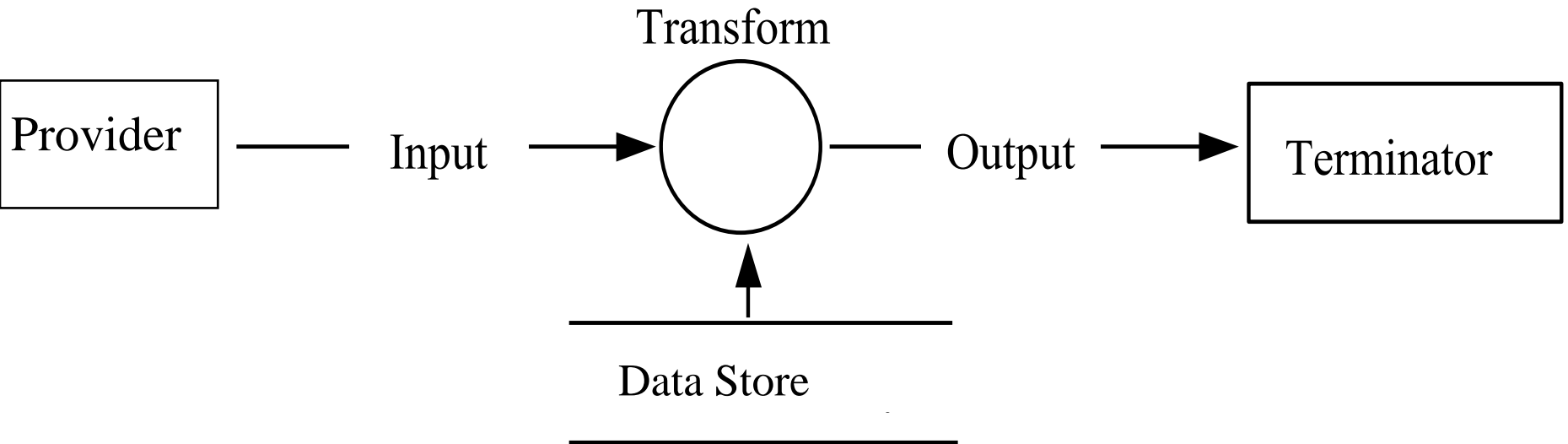
Process - трансформира или манипулира данни

Data flow - пренася данни между източник и процес, процес и получател, или между два процеса

Data Store - хранилище на данни

Connector symbols (конекторни символи), ако диаграмата се представя на няколко страници.

Data Flow Diagram (DFD): нотация



- *Прости и интуитивни*
- *Пример*

Документиране на изисквания - DFD

Пример:

R15: If a glass break detector at the window detects that the pane has been damaged, the system shall inform the security service.

Три перспективи:

Структура (данни): glass break detector at window, pane, security service

Функция: detects, inform

Поведение: If ...damaged, then inform ..

DFD йерархия

- DFD могат да се използват на различни нива на абстракция (*levels of abstraction*).
- Най-високото ниво на DFD е Контекстаната диаграма.
- Процесът може да се покаже подробно на по-ниско ниво на DFD.
 - В такъв случай, нетните входящи и изходящи потоци трябва да бъдат балансирани на съответните нива на DFD.
 - По-ниското ниво на DFD може да има складове за данни, които са локални за него и не са показани на по-високо ниво на DFD.

Най-ниското ниво в DFD

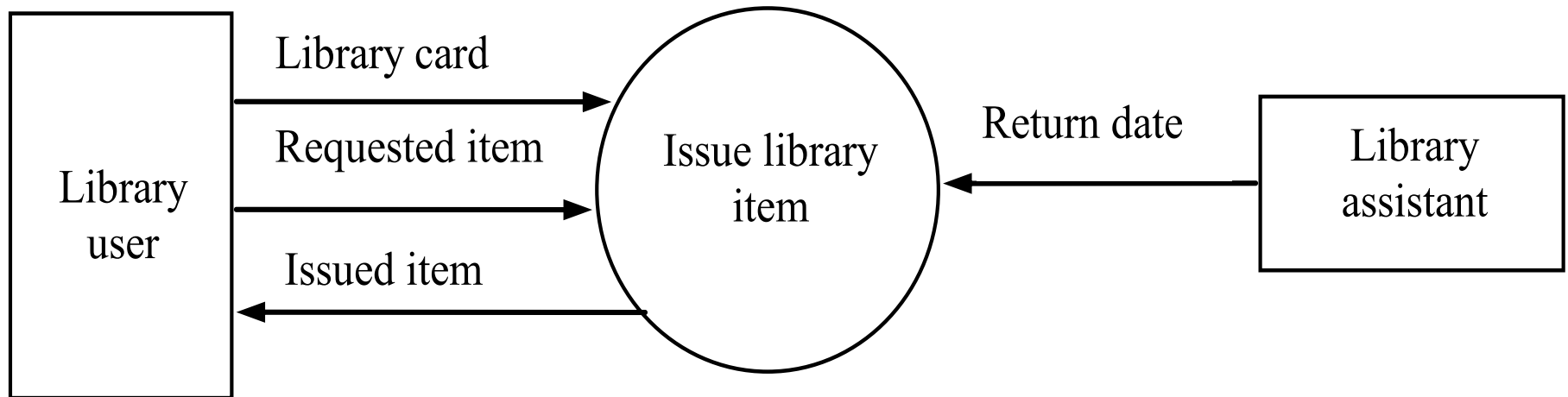
- Процес, който не може да бъде разгледан по-детайлно се свързва с мини спецификация.
- Мини спецификация:
 - Структурирано описание на естествен език
 - Диаграма на действия
 - Таблици (Decision Table, Decision Tree)

Пример: DFD на електронна библиотека

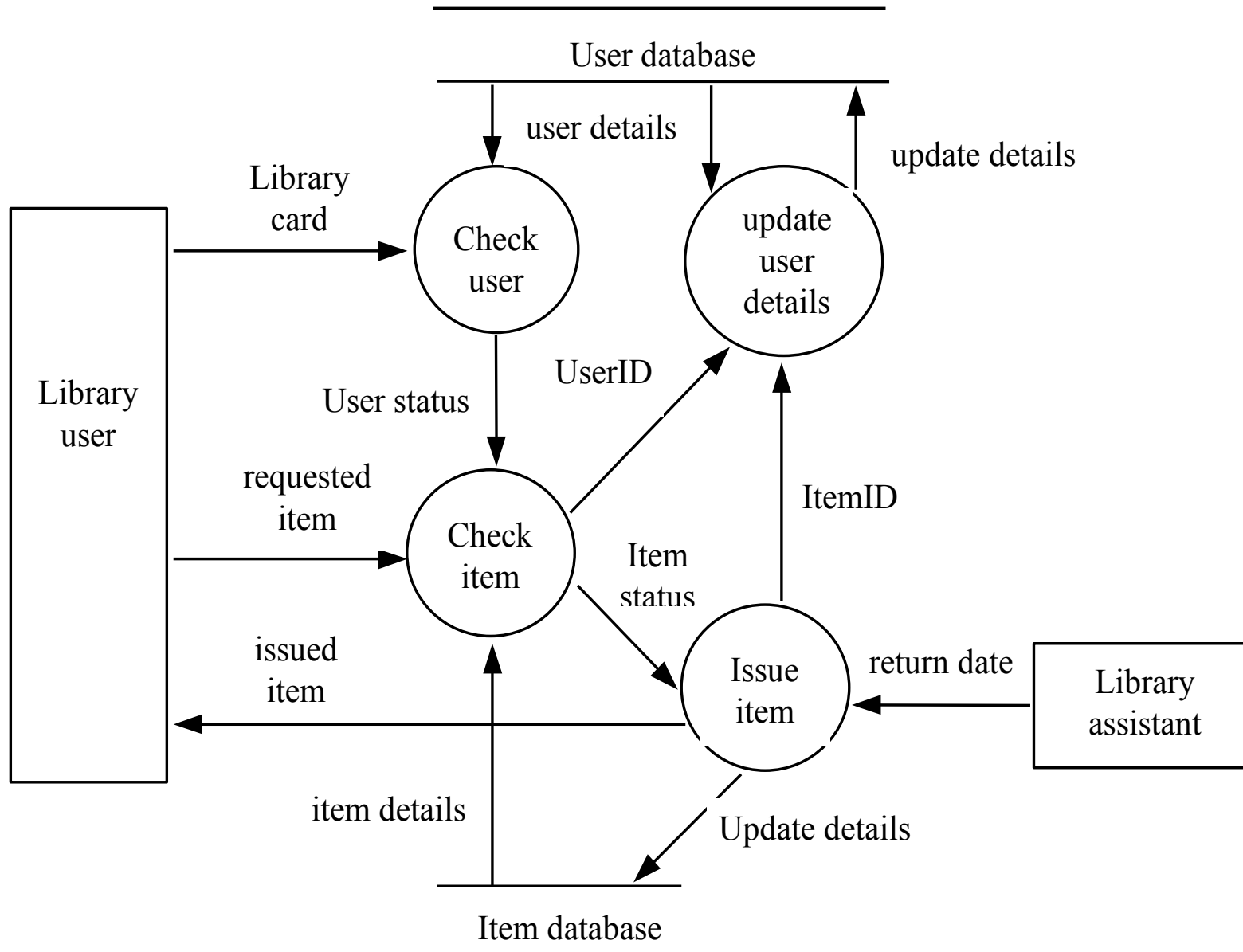
Библиотечна (елементарна) система за автоматизирано издаване (заемане) на библиотечни документи.

- **Първата диаграма** на потока на данни е контекстно ниво
- **Level 1** на DFD е конструирано чрез разлагане “балона” на библиотечната система в под-функции
- **Level 2** ...
- ...

Library example - Context level data flow diagram



Library example - Level 1 DFD



Означения - вариабилност

Няма единно означение на елементите на DFD

- Показаното означение е въведено от DeMarco
- Gane и Sarson използват *rounded rectangles* за процеси *shadowed rectangles* за източници и получатели, and *squared off C's* за хранилища на данни
- Orr използва *rectangles* за процеси, *ellipses* за източници и получатели и за хранилища на данни

Допълнение: DFD Guidelines

- Choose *meaningful* names for processes, flows, stores, and terminators
 - use active unambiguous verbs for process names such as Create, Calculate, Compute, Produce, etc.
- *Number* the processes: a DFD at any level, should have no more than *half* a dozen processes and related stores
- Make sure that DFD is internally ***consistent***
 - *avoid infinite* sinks
 - *avoid spontaneous* generation processes
 - *beware of unlabeled* flows and *unlabeled* processes
 - beware of *read-only* or *write-only* stores

Структурен анализ

- Подходът на потока на данни е типичен за метода на структурен анализ (Structured Analysis (SA))
- Две основни стратегии доминират структурния анализ

"Old" метод популяризиран от DeMarco 1978 г.

- Лесен за разбиране +
- Позволява декомпозиция +
- Описанието на данните е остаряло –

"Modern" подход от Yourdon (1984, 1990)

Подход на DeMarco

- Подход отгоре-надолу

Физическата система се описва с модел на логическия поток на данни чрез:

- Анализ на текущата физическа система
- Дефиниране на логически модел
- Разработване на логически модел
- Изграждане на нова система

Съвременен структурен анализ

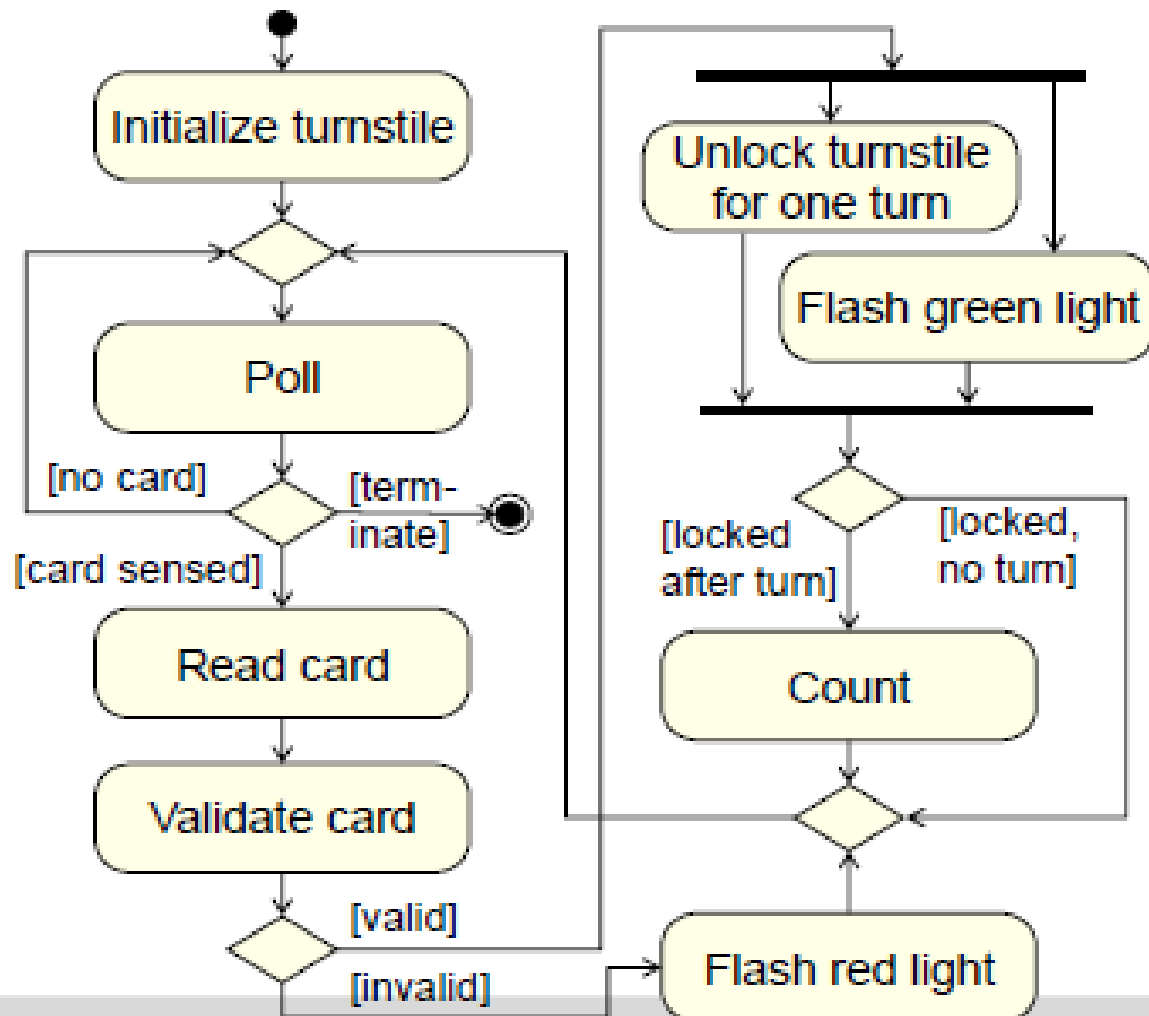
- Прави разлика между *реални* (съществени) нужди на потребителите и тези изисквания, които представляват външно поведение, което отговаря на тези нужди.
- Case tools (1990)
- Разширения за инженерни проблеми в *реално време* през 1985 г. и 1987 г. за описание на управление и поведение.
- Други подходи за структурен анализ:
 - Structured Analysis and Design Technique /Техника за структуриран анализ и дизайн (SADT) 1977
 - Структурирани системи за анализ и Методология на проектирането (SSADM), 1994

Недостатък! DFD не е ефикасен подход за сложни системи/интерфейси, защото се изисква отделна диаграма за всеки възможен вход.

2. Модел на дейностите:

Activity modelling: UML activity diagram

- Models process activities and control flow



3. Модел на процеси и работни потоци:

Process and workflow modelling

■ Elements

- Process steps / work steps
- Events influencing the flow
- Control flow
- Maybe data / information access and responsibilities

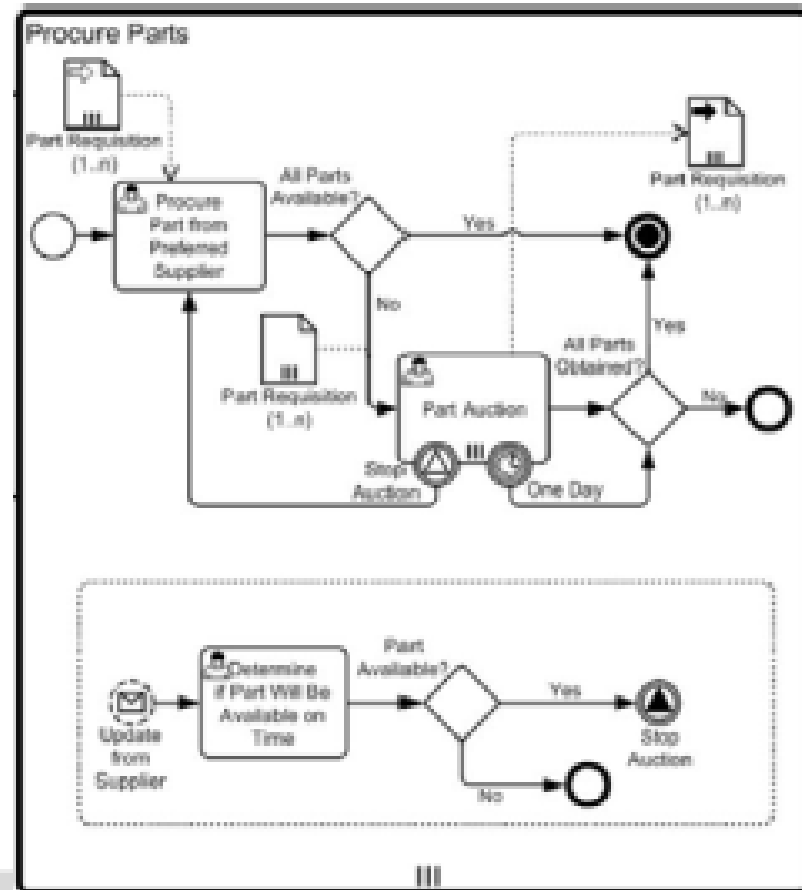
■ Typical languages

- UML activity diagrams
- BPMN
- Event-driven process chains

Process modelling: BPMN

- BPMN (Business Process Model and Notation)
- Rich language for describing business processes

[Object Management Group 2011a]



Документиране на изискванията от перспектива на поведението на системата

State Transition Diagram (STD)

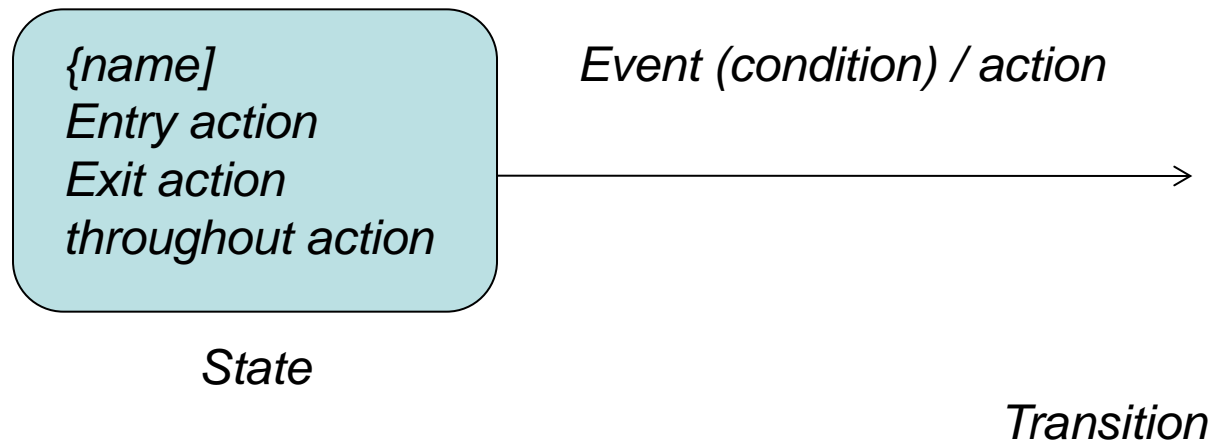
- Показва **динамиката в поведението** на една система:
 - Как системата реагира на последователност от външни събития/действия
 - Как независими компоненти координират работата
- Основни компоненти
 - Състояние (State)
 - Действие/събитие (Event):
 - Външно и/или вътрешно
 - Времево

Преходи между състоянията
- Полезно за описване на поведението на
 - системи в реално време (примери?)
 - интерактивни системи

Видове модели на поведението. Означения

Statecharts (Harel 1987)

- Улеснява дефиниране на действията, които системата ще изпълнява



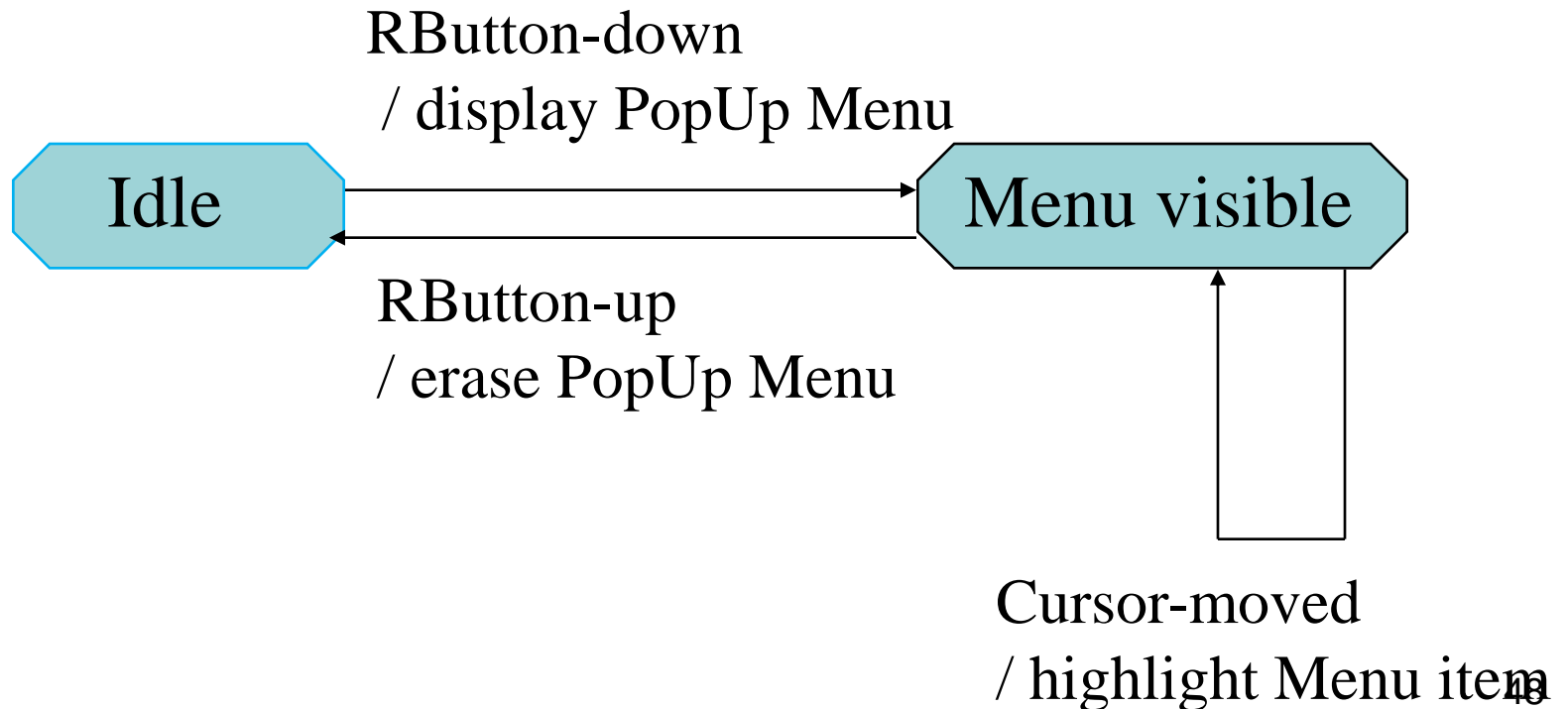
- Условни преходи
- Йерархични зависимости (за сложни системи)
- Паралелност

Забележка!: *Необходим е речник на означенията*

State Machine Diagram

The UML 2 SMD is based on Statecharts modeling language

Пример:



3. Модели на данните

- **Начин да представим структурата на данните**
- **Моделът на данните се ползва често заедно с DFD**
- **Видове модели на данните**

3.1. Релационен модел (Codd, 1979; Date, 1990)

- Задава данните като набор от таблици с общи ключове
- Недостатъци на релационния модел
 - Неявно описание на данните
 - Неадекватното/неудобно и непълно моделиране на релациите
- Подобрение - чрез включване на информация за семантиката на данните

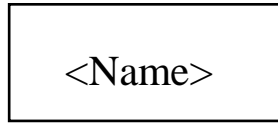
3.2 Семантичен модел

- Подходите на семантично моделиране на данни включват:
 - **Entity-relationship model ERM** (Chen, 1976) и вариантите му:
 - RM/ T (Codd, 1979)
 - SDM (Hammer and McLeod, 1989)
 - Hull and King, 1987
- Моделите представят: *субектите* (entities) в база данни, техните *атрибути* и техните *връзки*.
- Използва *графични нотации, софтуерни инструменти*
Пример: UML - същностите като обектен клас

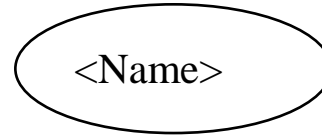
Модел същност-връзка (Entity Relationship Diagram (ERD) / Model)

- Представя логическия вид на данните и връзките между тях
- Основни компоненти:
 - *Същност (Entity)*: обект или дейност
 - *Атрибути (Attributes)*
 - *Връзки (Relationships)* - елементи:
 - кардиналност (cardinality): (1:1), (1:n), (n:1), (n:m)
 - опционалност: задължителна или незадължителна
 - зависимост:
 - а) рекурсивна връзка
 - б) под тип и над тип (supertype/subtype),
осъществени чрез наследствена връзка

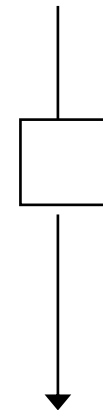
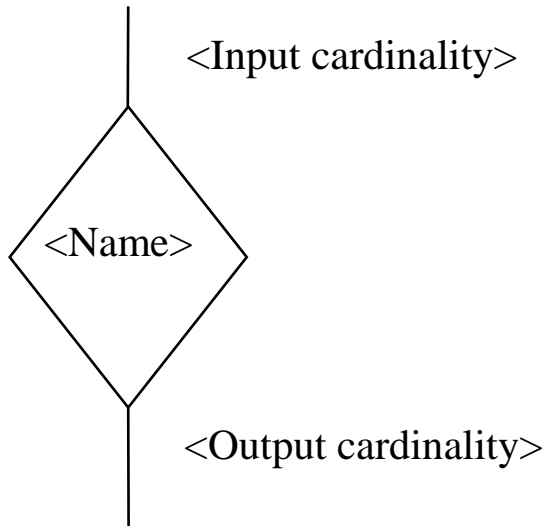
ERM notation



An Entity

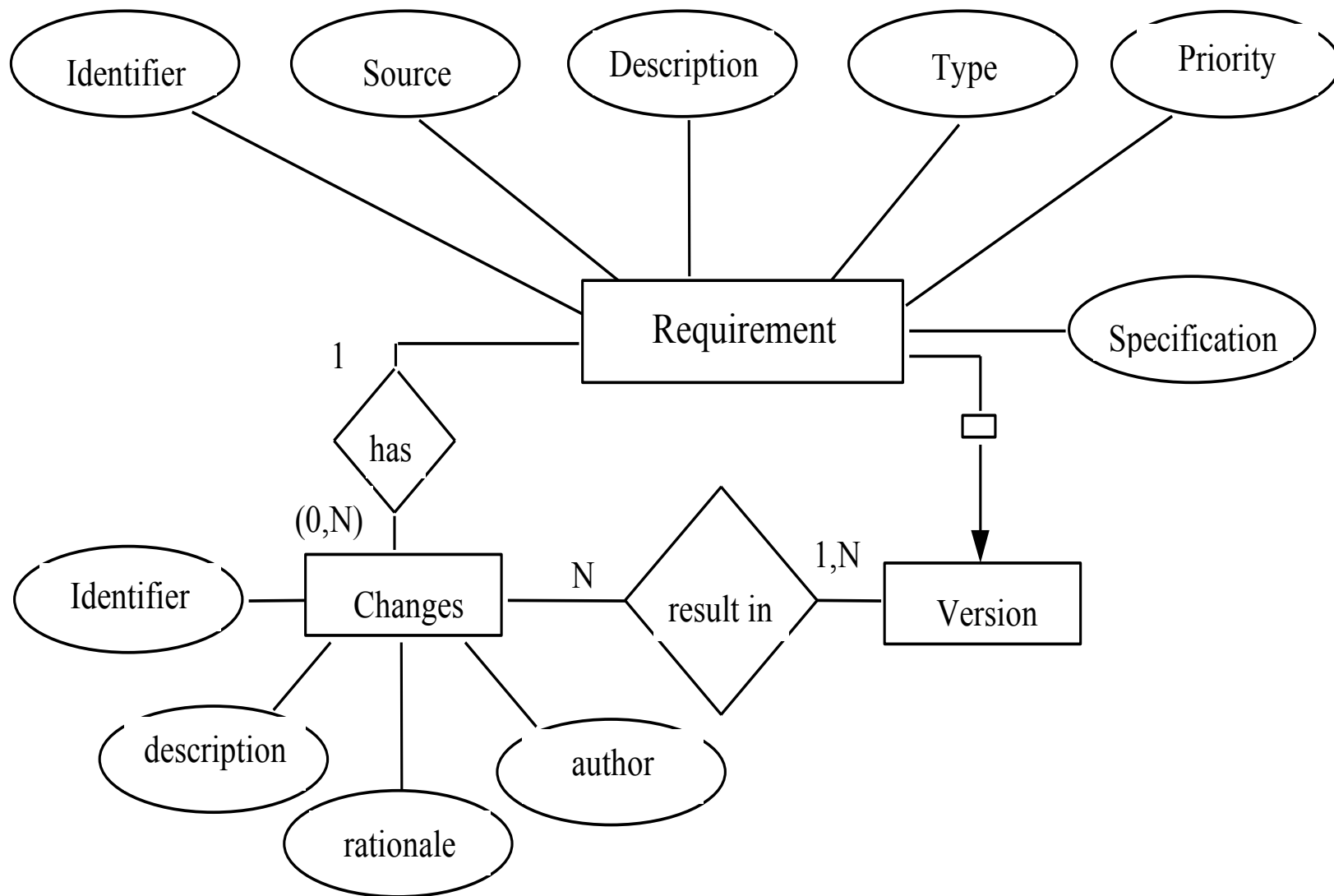


An entity/relation attribute



An inheritance relation

Пример: ERD на променящо се (evolving) изискване



3.3 Обектни модели (ОМ)

Обектно-ориентиран подход за моделиране

- ERM е най-близкият предшественик на **обектно-ориентирания модел**
- ОМ представят *едновременно данните и тяхна обработка*, както и *класификация и връзки* на субектите на системата
- Основната идея е идеята за **ОБЕКТ**
- Обектно-ориентираният анализ, проектиране и разработка е общоприет.

История:

- Shlaer and Mellor (1988) – first published material
- Colbert (1989)
- Coad and Yourdon (1990)
- Wirf-Brock (1990)
- **Rumbaugh** (1991)
- **Jacobson** (1992)
- Martin-Odell (1992)
- **Booch**, 1994

Забел.! Означенията за различните методи са семантично сходни.

- **UML (1999) е обобщаващ метод и ефективен стандарт за ООМ⁵⁴**

Основни концепции на обектно-ориентираното моделиране

- **Обекти и класове (Objects and classes)**
- **Капсулиране (Encapsulation) на данни за обект**
- **Наследство (Inheritance)**
- **Съобщения (Messages)**
- **Методи (Methods)**

Обектно-ориентираната парадигма се използва последователно от анализа до реализацията за целия процес на софтуерното инженерство.

Но! Понякога крайните потребители го намират за неразбираем и предпочитат и функционален модел като DFD.

Обектен модел: елементи

Обект:

Нещо реално или абстрактно, за което запазваме данни и описваме операции за манипулиране на тези данни.

- Обектите са концепции за *капсулиране*, *клас* и *наследство*
- Обектите поддържат абстракцията на компонент чрез разлагането на класовете на отделни обекти.

Обекти

- Типични обекти:
 - Физически места, сгради ...
 - Устройствата, с които системата взаимодейства
 - Системите, които си взаимодействат с проучваната система
 - Организационни единици
 - Нещо, което трябва да бъде запомнено с течение на времето
 - Специфични роли на хората, които използват системата.

примери: сензор, кола, акаунт, софтуерен дизайн, отдел, ...

- Обектите са *същности с атрибути на данните и операции* на обектния клас
- **Обектите може да бъдат композитни.**

(Обектен) Клас – основен елемент на ОМ

- Описание на множество от обекти, които споделят общи атрибути и операции (услуги)
 - Обектът е *инстанция* от класа
- Класът реализира обекти от даден тип
 - *Пример*: The object type *Bank Customer* refers to a **class of bank customers**.
If “John Smith” is a bank customer, then **bank customer** is the *class* and “John Smith” is an *instance* of the bank customer.
- Трудности при идентифицирането

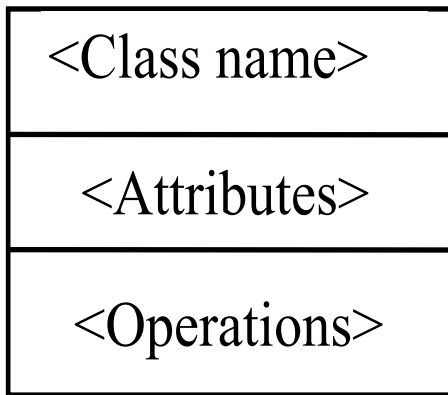
Операции и методи

- **Операция е действие за манипулиране данните на обект – четене, запис.**
 - Отнасят се *само* до структурите на данните от този тип обект.
 - За достъп до структурите за данни на друг обект, обектите трябва да изпращат *съобщения* до този обект.
- **Метод е реализацията на дадена операция – специфицира се пътя, по който операциите се кодират в софтуера.**

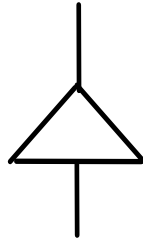
Капсулиране

- **Заедно пакетиране на данни и операции**
- **Предпазва** данните на обекта от **неоторизиран достъп**
- Детайли за това, как операциите се реализират остават **скрити** от клиентите.

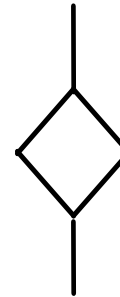
Object-oriented notation



(i) Class



(ii) Generalisation

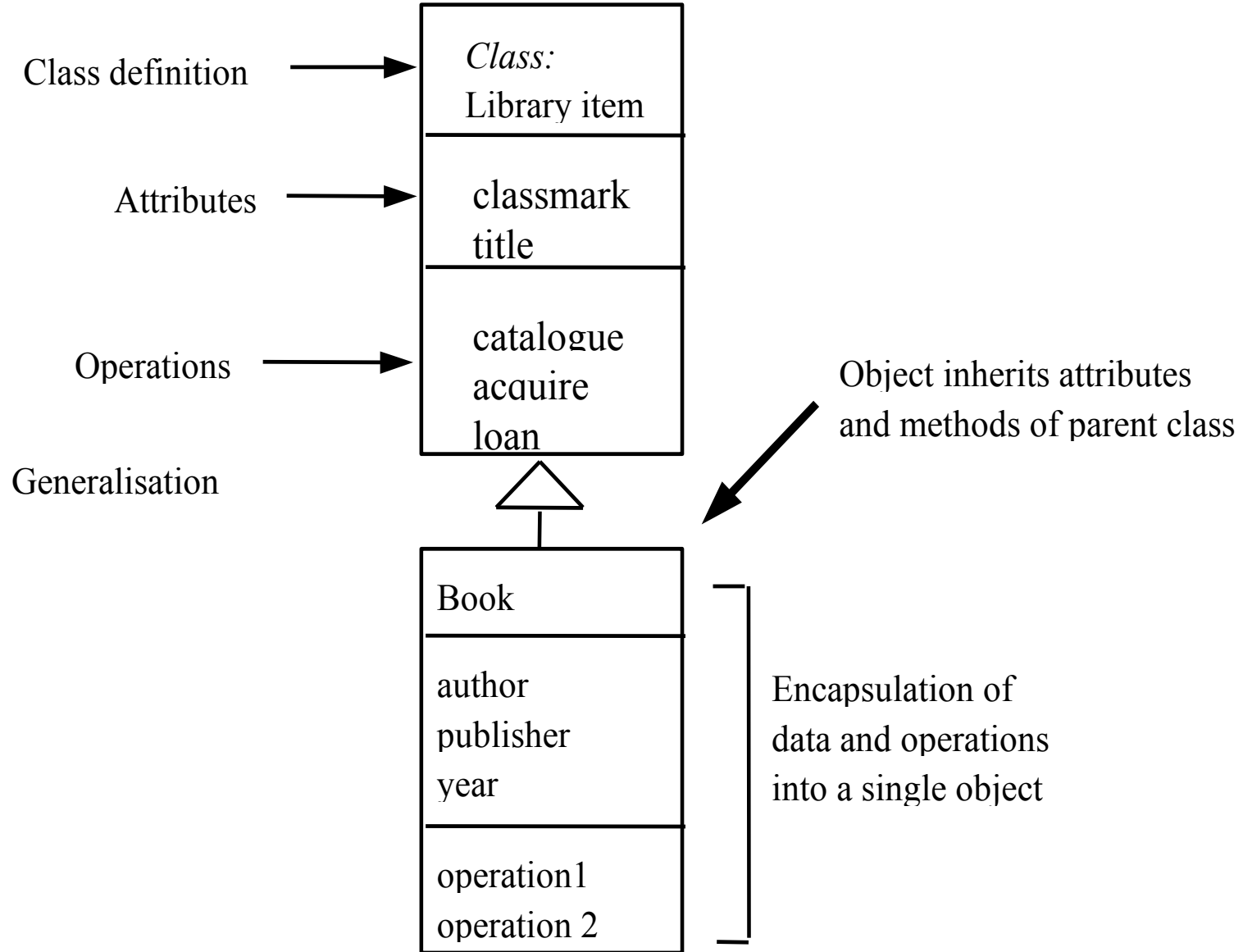


(iii) Aggregation

Наследство

- Таксономия на класовете
- Обекти на по-ниско ниво в **йерархията** на класа наследяват атрибутите и операциите на техните родители.
- Обектите са в състояние да включат данни и/или операции, които са специфични за тях.
- Наследяването на данни от повече от един родител се нарича **множествено наследяване**. Наследените атрибути и услуги са **конюнкция** на родителските.

Пример: Концепция за обектно-ориентиран подход –2



Агрегиране (Object aggregation)

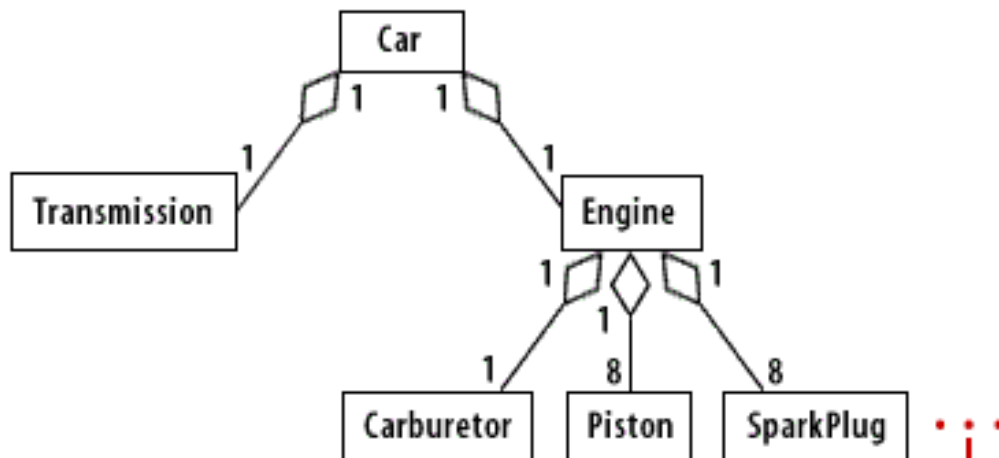
Обекти са съставени от други обекти

- Пример 1:

„A study pack (of a course) is composed of one or more assignments, slide packages, lection notes and video types.”

- Пример 2:

This example shows multiple levels of aggregation within the same object.

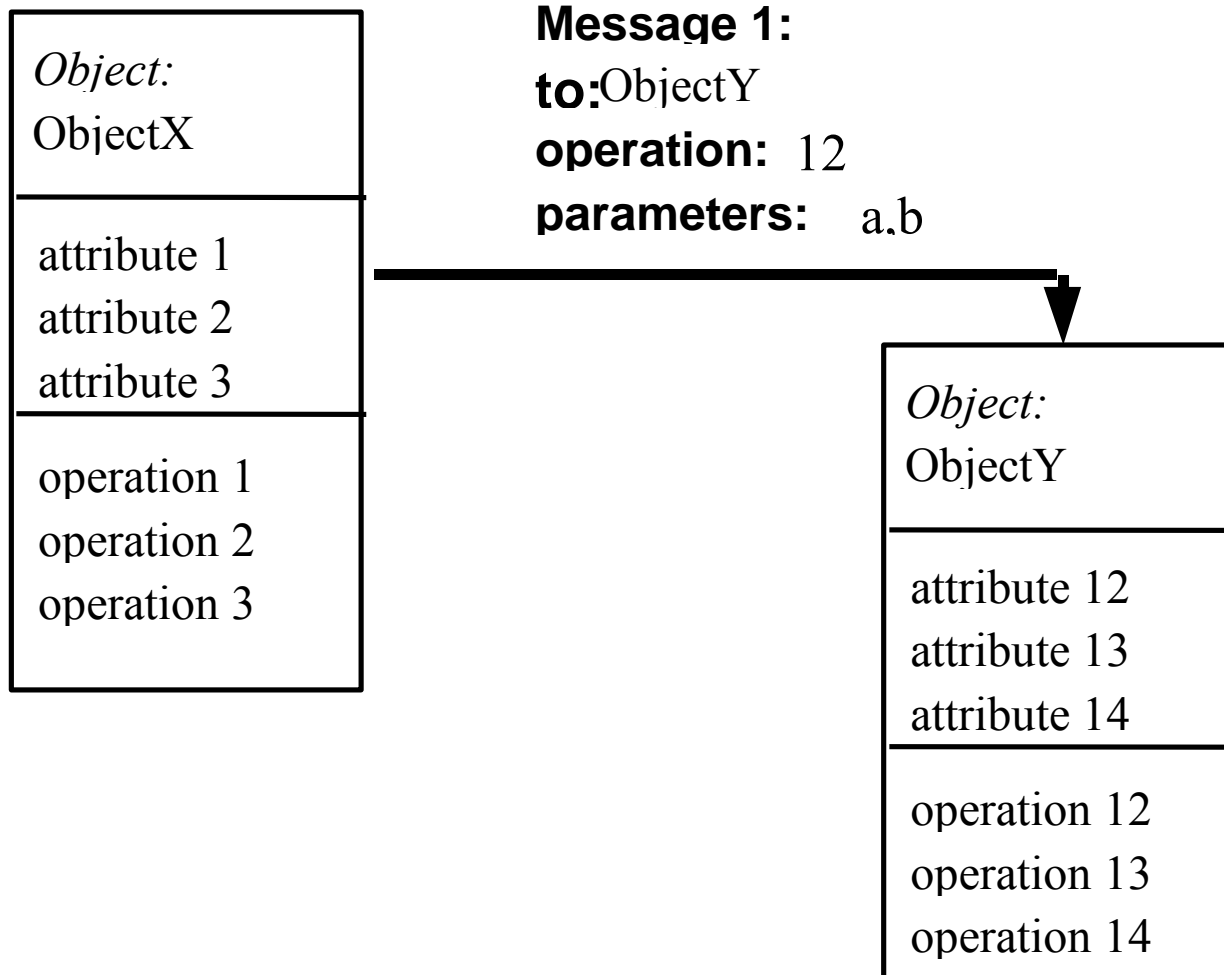


Indicates that there are other parts of the aggregate not shown in this diagram

Съобщения

- **Обектите комуникират чрез изпращане на съобщения**
- **Съобщенията обхващат:**
 - Името на обекта (класа) получател
 - Операцията, която трябва да се извика
 - Опционално - множество от параметри
- **Когато обект получи съобщение, то той извиква операция**
- **Операцията изпълнява подходящ метод**

Message passing



Стъпки в обектно-ориентирания метод за моделиране

Повечето методи, базирани на обектно-ориентиран модел споделят общи/еднакви стъпки на анализ:

- 1) Идентифициране на основните обекти
- 2) Конструирание на структури на обектите, определяйки асоциациите на обекти в класовете
- 3) Дефиниране на атрибутите, свързани с всеки обект,
- 4) Определяне на съответните операции за всеки обект
- 5) Дефиниране на съобщенията, които могат да бъдат предавани между обектите

Notes!: Object class identification is recognised as a difficult process requiring a deep understanding of the application domain.

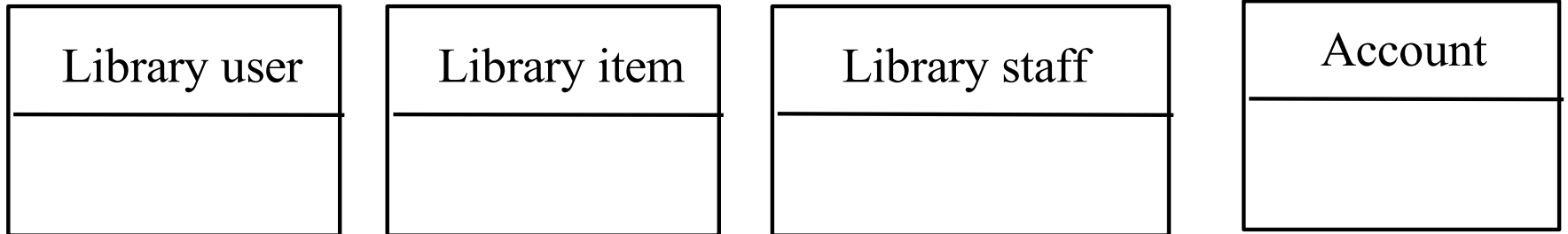
Object classes reflecting domain entities are reusable across systems.

Пример за обектно моделиране: Library example

Изисквания:

- A library system is intended to provide its **users** with the ability to automate the process of:
 - Acquiring library items
 - Cataloguing library items
 - Browsing library items
 - Loaning library items
- **Library items** comprise published and recorded material
- The system will be administered by a member of the **library staff**
- **Users** must *register* with the system administrator before they can borrow library items

Step 1 - Initial classes identified



Library example (cont.)

Изисквания:

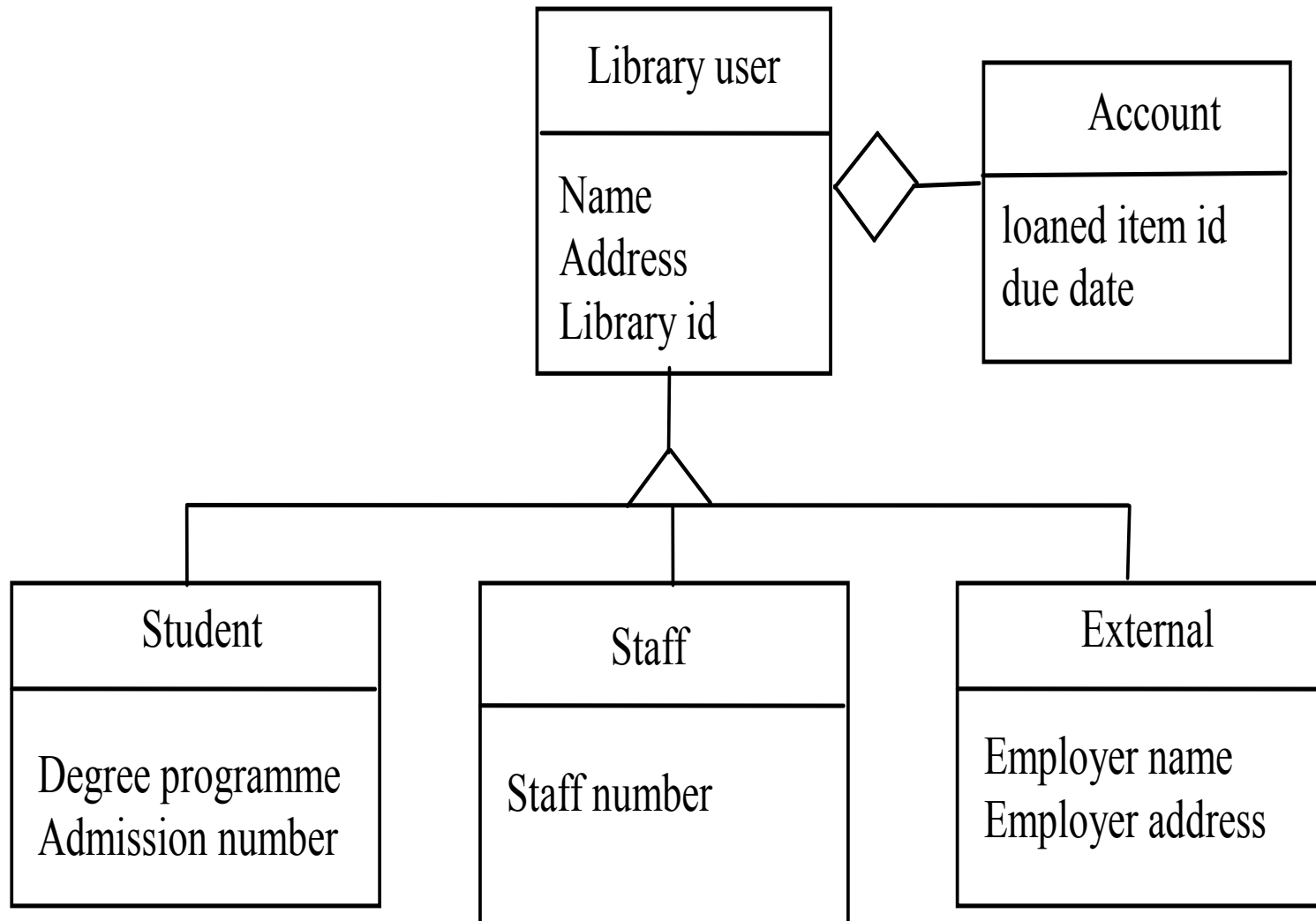
- **Library users** are drawn from three primary groups:
Students, Members of staff and External users
- **All library users** have as part of their registration:
*Name, Library number, Address, **Account***
- **In addition** the following information also required for registration:
Students - Degree programme and admission number
Staff - Staff number
External users - Employer details

Step 3 - Identifying the attributes

Attributes can be revealed by analysis of the system requirements

- For example, it is a requirement that **all library users must be registered** before they can use the library
 - This means that we need to keep registration data about library users
 - Library users may also be provided with an **account** to keep track of the items loaned to them
- Library item has the attributes; title, description and classmark
- The library user class has the attributes; name, address and library ID

Step 2, 3 - Inheritance for Library user



Step 4 - Relationships between classes

Изисквания:

We can identify the following relationships from the partial requirements:

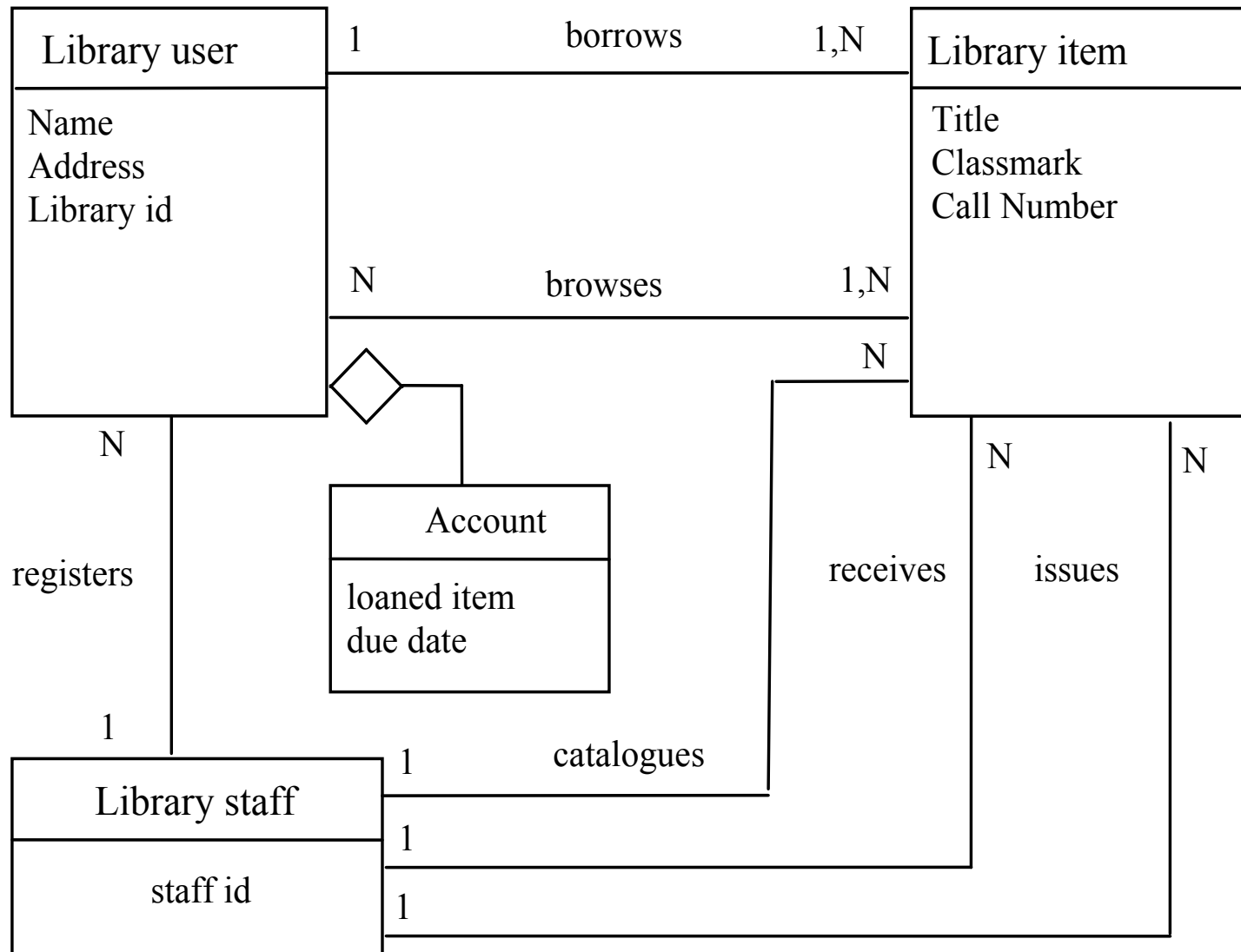
- (i) A library user **borrows** a library item
- (ii) A library item **is recorded** or **published**
- (iii) The system administrator **registers** the library user
- (iv) Library users **are** students, staff and external users
- (v) The system administrator **catalogues** the library items
- (vi) The library assistant **issues** the library items

....

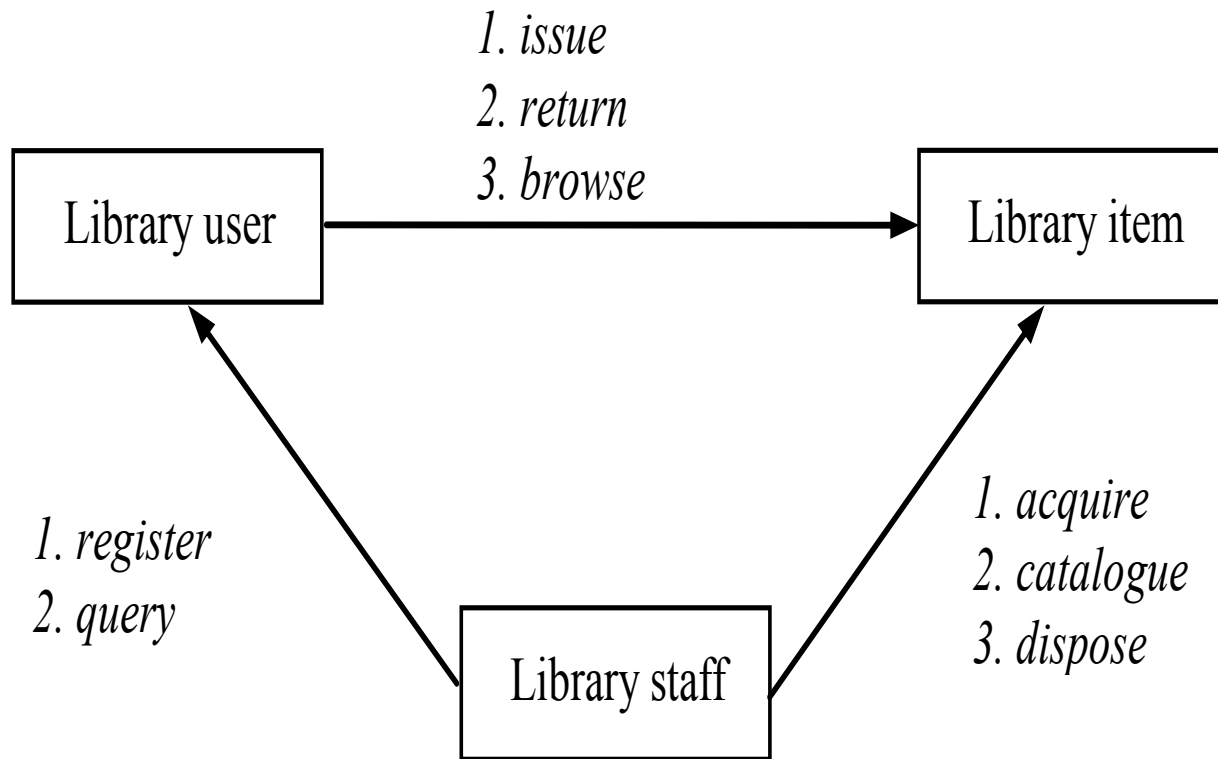
Step 4 - Object operations

- This step is intended to describe operations to be performed on the objects.
- Certain operations are **implicit from the object structure**
 - These include operations for accessing and modifying the attribute values within the class. These operations are assumed and we need not show them explicitly in the model.
- One way of identifying operations is by **modelling the messages** that may be passed between the objects.

Basic object operations



Step 5 - Messages between objects



Речник на данни

Речникът на данните е **организиран списък** на всички елементи, отнасящи се до системата с точни строги дефиниции, за да се има общо разбиране за всички елементи на системата.

(например) речникът описва:

- примитивни елементи на данните като *име, размер, обхват*
- *значение и обхват* на потоците от данни, на хранилища за данни
- подробности за *връзките*, които са дефинирани в моделите

Речник на данни: предимства

- Начин да осигурим *консистентност* на спецификацията.
- Механизъм за *управление/организиране на информация* чрез имена (на обекти, елементи, състояния, действия,...).
- Запазва *организационна информация*, която може да свързва анализа, проектирането, разработването, използването, поддържането и разширяването.

Речник на данни: примери

1. customer-name = title + first-name + (middle-name) + last-name

```
title = [Mr. | Miss | Ms. | Mrs.| Dr. | Prof. ]
```

first-name = {legal-character}

```
order = customer-name + shipping-addr
                        +1:10 {item}
                        * comments*
```

2. Детайлно описание на състоянията на системата

Формални методи - класификация

- Категоризация в спектъра на формалното описание (“**formality**” spectrum) (Pressman, 1992)
- **Полуформални методи (Semi-formal and informal methods)**
 - Use natural language, Diagrams, Tables and simple notation
 - Structured analysis; Object-oriented analysis
- **Формалните методи включват:**
 - Математически синтаксис и семантика
 - *Примери: Z, B, VDM, LOTOS, CSP ...*

ФМ за целите на разработването на софтуер

- Осигуряват високо ниво на точност на системата, която да покрие напълно спецификацията, но

Не могат да гарантират абсолютна точност!

 **Защо !**

- Не могат да решат всички въпроси и проблеми на анализа и разработката:

Разкриват само/част от функционалните изисквания.
Отнася се до всички модели - формални и полуформални.

 **Защо !**

Предимства на ФМ за целите на разработването на софтуер

Формалните методи са ниво на абстракция, която позволява да се създаде качествен (удобен, разбираем, надежден) софтуер.

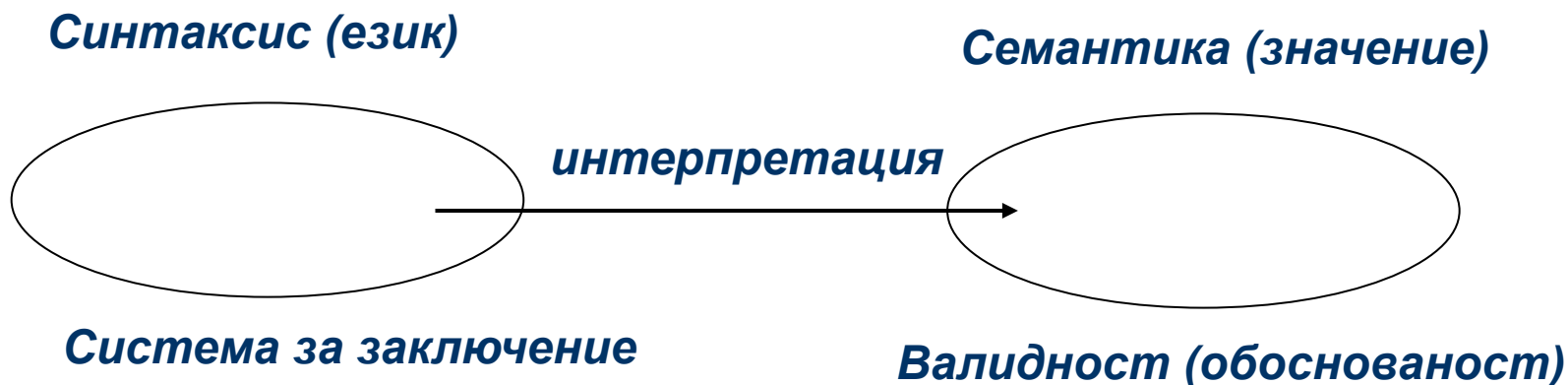
- Намаляват неточността
- Осигуряват строгост в описанията на ранни етапи от разработката
- Могат да проверят/гарантират коректността и точността, непълнотата и неконсистентността на спецификацията.

Разпространение на ФМ

- ФМ не са широко разпространени
(FM are slowly emerging from academic research and now being applied in industrial software developments where *safety or security is critical*.)
- **Причини:**
 - Трудности да бъде разбрана спецификацията
 - Трудна формализация на отделни аспекти на системата
 - Трудна възвръщаемост на инвестицията +
+ (консерватизъм и нежелание).

Компоненти на формалната спецификация

- **Синтаксис (Syntax):** *специфична нотация за представяне на спецификацията.* Базира се на синтаксиса на теорията на множествата и предикатните изчисления.
- **Семантика (Semantics):** *дефинира съвкупността от обекти, които ще се използват, за да се опише системата.*
- **Връзки (Relations):** *дефинират правилата, които показват кои обекти правилно удовлетворяват спецификацията.*



Z нотация: въведение

Една формална спецификация трябва да може да представи голямо количество проза. Тя съпоставя математическите обекти към особеностите на проектираната система: състояния на системата, структури от данни, техни свойства и операции с тях.

Как се постига ясна и точна спецификация?

Съществуват два елемента, които изграждат формалната спецификация на **Z нотацията**:

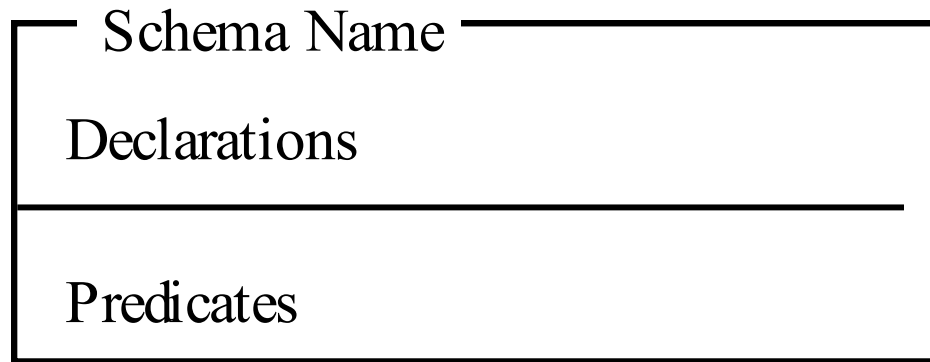
- **математическият**: теорията на множествата и математическата логика;
- **езикът на схемите** се използва за *структурни* и *композиционни* описания: събиране на части от информацията, формулиране на общи описания и доказателства, необходими при следващо приложение.

Z нотация: Схеми

- използва **Теория на множествата и логиката**
 - множества, релации, функции
 - предикатна логика от първи ред
- **Схемите** са именувани записи, които съдържат:
 - име
 - декларация
 - предикат
- **Изчисления със схеми (Schema calculus)**
 - Математически оператори, чрез които се изграждат по-големи схеми чрез по-малки
 - **Z описанието се представя като колекция от схеми**
 - Синтактични събирания (Syntactic conventions)
 - за да се направят описанията лесни за възприемане;
 - за да улесним програмирането.

Z schema

Z схемата обхваща 3 части: Име, Декларация и Предикат

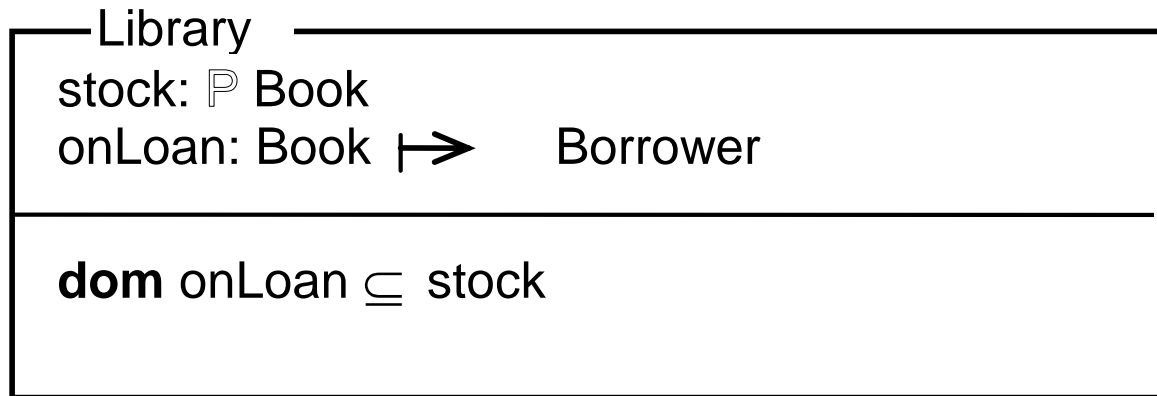


Декларациите на схемата съдържат имената на обектите и техния тип:
идентификатор: тип

Предикатите на схемата дефинират връзките между обектите в декларацията, за да представят специфични свойства.

Пример на описание със Z нотация: Library example

- **Basic sets** for this system are [Book, Borrower]
- The **state space** of the lending library can be defined using the following schema:



Schema for borrow operation

Borrow

Δ Library

book?: Book

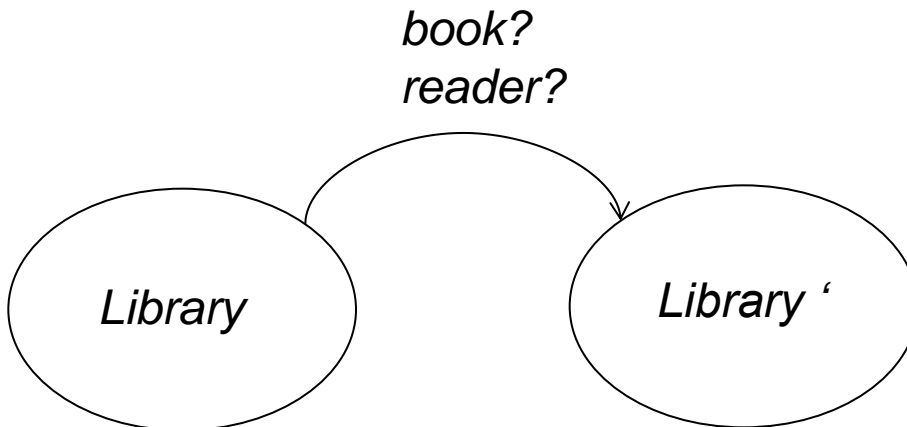
reader?: Borrower

book? \in stock

book? \notin dom onLoan

onLoan' = onLoan \cup {(book?, reader?)}

stock' = stock



Key points

- No ideal requirements method.
- System models can be considerably enriched by combining different techniques.
- Data-flow model is based on the notion that systems can be modelled as a set of *interacting functions*.
- The object-oriented approach is based on the notion that systems can be modelled as a set of *interacting objects*.
- Formal methods are based on mathematical principles and are intended to achieve a high degree of confidence that a system will conform to its specifications.