Rigorous Development of Service-driven Applications

Dr. Nasreddine Aoumeur
FIN, ITI, DB group
aoumeur@iti.cs.uni-magdeburg.de

Course Web-Site:
wwwiti.cs.uni-magdeburg.de/~aoumeur
wwwiti.cs.uni-magdeburg.de/iti_db/lehre/soa
Information Systems: Working definition

- *reactive* systems (i.e. in continuous interaction with their environment), with

- large amount of *immutable and non-immutable data* (i.e. fixed and changing) and, with

- *processes and activities* for exhibiting behaviors on these (state-less and -full) data.
Information Systems: Graphical show

Environment

Potential
Users
(e.g. Private / Corporate)

Organizations

Society

Information system

State-less and -full DATA

Processes and Rules

Managerial / organizational / human / Business concerns
Information Systems: Illustrations

Environment

Users

Banks

Actions (public/private)

Banking system

State-less and -full DATA

Account:
- number
- balance
- limit
- status
- ...

Customer:
- name
- holder-account
- credit
- ...

Mortgage:
- name
- ...
- status
- ...

rule1: how to withdraw
Rule2: how to proceed credit-card ...
Rule3: ...

process1: mortgage ...
Process2: insert → transact --> ...
Process3: ...

Processes and Rules

withdraw

deposit

transfert

action
**IS Model-driven life-cycle**

**Domain level**
- Goals
- Stakeholders
- Resources
- General functioning
- Functional requirements
- Non-Functional requirements
- Interfaces (to the outside)

**Organization**

**IS Conceptual Modelling Level**
- Abstract description of (non-) functional and business requirements: *Conceptual Model (CM)*
- Validation / Verification of the CM (symbolic animation / formal verification)
- Transformation rules to implementation

**State-less and -full DATA**

**Processes and Rules**

**IS Deployment level**

```java
aspect activity1 {
    pointcut chg1 = call (* .. Within..}
    before : …
    After : …
}

aspect activity2 {.....
    around : call(...) }

Public static main {
    activity1(...); activity2(…)
    If …while switch (activity_)
    -…..
```
**Requirements on Good IS Conceptual Model**

**Complete ( ... as possible)**
- Cope with **Structural** and **behavioral** features
- Endowed with rich and intuitive **abstraction mechanisms**
- Deal with integrity/safety constraints

**Flexible**
- Supports **modularity** and compositionality
- Supports modifiability, **adaptivity**

**Precise**
- Governed by **mathematical** semantics
- Allows validation / verification

**(symbolically) Executable**
- Supported by **operational** / symbolic computational model
- Supported by graphical animation tools and translators

**Methodologically Supported**
- Bottom-Up or Top-Down and better mixed development process.
Different generations of CMs: “Entity first”

**IS Conceptual Model**
- Entity-Relationship [1973..]
  - Intuitive / Simple
  - Revolutionary for IS
  - UoD: Entities+Relations
  - Mathematically sound
  - Rich abstraction mechanisms
  - Different variants
    (NIAM / SADT / MERISE /..)

**Entity-less and -ful DATA**

**Processes and Rules**

**Strengths and advantages**

**Process-centric Formalisms**
- CCS, CSP, DFD, Petri Nets, ...
- Synchronous / asynchronous
- Mathematically sound
- Executable / operational
Different generations of CMs: “Entity first”

IS Conceptual Model

State-less and -ful DATA

Conflict / contradiction

Entity-Relationship
- Only structural aspects
- Not so modular / compositional
- No Adaptivity at all

Rapidly outdated

Weakeness and limitations

Unrelevant

Process-centric Formalisms
- No reasoning on data
- Not quite specific to IS
- Not compositional / evolving

Processes and Rules
Different generations of CMs: “Object first”

Object Paradigm [1980..]
- Really Intuitive / highly accepted
- Object (Id, structure / dynamic)
- UoD: Interacting objects community
- Rich abstraction mechanisms (e.g. Class, inheritance, role, aggregation, ..)
- Different formal interpretations

Strengths and advantages

Behavior-centric OO Formalisms
- OO Petri Nets, Mondel, JSD..  
- Mathematically sound
- Executable / operational / concurrent
- Cope with data and process around objects
Different generations of CMs: “Object first”

IS Conceptual Model

State-less and -ful DATA

UML
- Class-object-diagrams
- OCL constraints

Non-conformance, Redundancy, rigidity

Object Paradigm [UML]
- Not evolving / adaptable
- Not rigoros (UML)
- No coherent global unique view
- No explicit business rules and behavioral connectors
- Not service-driven

Weakness and limitations

Processes and Rules

UML
- Activity-diagrams
- Sequence diagrams
- Statechart diagrams

Rigoros and Adaptive Service-driven Applications
Current generation of CMs: “Interaction first”

**IS Conceptual Model**

**Aspect-/connector-/meta-level**

All What is:

- **Evolving** (business rules, market laws, cross-organizational policies, …)
- **Cross-Cutting** (security, management, …)
- **Non-functional** (availability, performance, …)

**Aspect/architectural Paradigm**

- Explicit separation between **Coordination** and computation
- Separation between what is evolving from what is stable
- Dynamic shifting-down / up of changing behavior
- Suitable for services and mobility

**Stable and invariant entities**

**Entity Interfaces**

Dynamically weaving
The Entity/Relationship CM: Overview

State-based entity

Book

Reference: Math-CM12 [string]
Title: Petri nets [string]
Author: W. Reisig [string]
Copies: 4 [nat]
State: borrowed [list-string]

Fixed data-types
The Entity relationship Model

• The E-R (entity-relationship) data model views the real world as a
  set of basic entities and relationships among these entities.

• It is intended primarily for the DB (data part of an IS) design process
  - by allowing the specification of an enterprise scheme.
Entities and Entity Sets

• **An entity** is any object that exists and is distinguishable from other ones.

  **Example:** *the Petri.net-book-of-reisig-1985 is an entity*, as it can be **uniquely identified** as one particular book in the Library universe.

• An entity may be **concrete** (a person or a book, for example) or **abstract** (like a holiday or a concept).

• An **entity set** is a set of entities of the same type.

  **Example:** Book, Account.

• Entity sets need **not be disjoint**. For example, the entity set *employee* (all employees of a bank) and the entity set *customer* (all customers of the bank) may have members in common.
Entities and Entity Sets

- An entity is represented by a set of *attributes*.
  
  **Example:** name, security number, street, city for ```customer``` entity.

  The *domain* of the attribute is the set of permitted values (e.g. the telephone number must be *seven* positive integers).

- Formally, an *attribute is a function which maps an entity set into a domain.*
  - Every entity is described by a set of (attribute, data value) pairs.
  - There is one pair for each attribute of the entity set.

  **Example:** a particular *customer* entity is described by the set
  
  `{(name, Harris), (S.I.N., 890-123-456), (street, North), (city, Georgetown)}`.
Entities and Entity Sets: Illustrations

**Customer**: the set of all people having an account at the bank. Attributes could be:

**Employee**: with attributes
- `employee-name`, `phone-number`, `function`, `status`.

**Account**: the set of all accounts created and maintained in the bank. Attributes may be:
- `account-number`, `balance`, `limit`, `history`.

**Transaction**: the set of all account transactions executed in the bank. Attributes may be
- `transaction-number`, `date` and `amount`
Relationships & Relationship Sets

- A **relationship** is an association between several entities.
- A **relationship set** is a set of relationships of the same type.
- Formally it is a mathematical relation on \((n \geq 2)\) (possibly non-distinct) sets.
- If \(E_1, E_2, ..., E_n\) are entity sets, then a relationship set \(R\) is a subset of \(\{(e_1, e_2, ..., e_n) / e_i \in E_i\}\).
- **Example:** Consider the two entity sets **customer** and **account**.
  - We define the relationship **CustAcct** to denote the association between customers and their accounts. This is a **binary** relationship set.
The relationship set $\text{CustAcct}$ is a subset of all the possible customer and account pairings.

A relationship may also have descriptive attributes.

Example: $\text{date}$ (last date of account access) could be an attribute of the $\text{CustAcct}$ relationship set.
Attribute OR Entity?: Domain Question

It is possible to define a set of entities and the relationships among them in a number of different ways. The main difference is in how we deal with attributes.

- Consider the entity set *employee* with attributes *employee-name* and *phone-number*.
- We could argue that the phone be treated as an entity itself, with attributes *phone-number* and *location*.
- Then we have two entity sets, and the relationship set *EmpPhn* defining the association between employees and their phones.
- This new definition allows employees to have several (or zero) phones.

The question of what constitutes an entity and what constitutes an attribute depends mainly on the real world situation being modeled, and the semantics associated with the attribute in question.
Mapping Constraints and Cardinalities

An E-R scheme may define certain constraints to real world situations

- **Mapping Cardinalities**: express the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality could be:

1. **One-to-one**: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

2. **One-to-many**: An entity in A is associated with any number in B. An entity in B is associated with at most one entity in A.
Mapping Constraints and Cardinalities

3. **Many-to-one**: An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.

4. **Many-to-many**: Entities in A and B are associated with any number from each other.

The appropriate mapping cardinality for a particular relationship set depends on the real world being modeled. (Think about the *CustAcct* relationship...)

- **Existence Dependencies**: if the existence of entity X depends on the existence of entity Y, then X is said to be existence-dependent on Y. (Or we say that Y is the dominant entity and X is the subordinate entity.)
Example:

- Consider `account` and `transaction` entity sets, and a relationship `log` between them.
  - This is one-to-many from account to transaction.
- If an `account` entity is deleted, its associated `transaction` entities must also be deleted.

Thus `account` is dominant and `transaction` is subordinate.
Entity Keys

• Differences between entities must be expressed in terms of attributes.

• A superkey is a set of one or more attributes which, taken collectively, allow us to identify uniquely an entity in the entity set.

  Example: in the entity set customer
  - customer-name and S.I.N represent a superkey.
  - two customers could have the same name!

• A superkey may contain extra attributes, and we are often interested in the smallest superkey.

• A superkey for which no subset is a superkey is called a candidate key.
Entity Keys

• In the example above, S.I.N. is a candidate key, as it is minimal, and uniquely identifies a customer entity.

• A primary key is a candidate key (there may be more than one) chosen by the IS designer to identify entities in an entity set.

• An entity set that does not possess sufficient attributes to form a primary key is called a weak entity set. One that does have a primary key is called a strong entity set.

Example:

• The entity set transaction has attributes transaction-number, date and amount.

• Different transactions on different accounts could share the same number.
Entity Keys

• These are not sufficient to form a primary key (uniquely identify a transaction).
• Thus *transaction* is a weak entity set.
• For a weak entity set to be meaningful, it must be part of a one-to-many relationship set. This relationship set should have no descriptive attributes.
• The idea of strong and weak entity sets is related to the existence dependencies seen earlier.
• Member of a strong entity set is a dominant entity.
• Member of a weak entity set is a subordinate entity.
• A weak entity set does not have a primary key, but we need a means of distinguishing among the entities.
Entity Keys

- The **discriminator** of a weak entity set is a set of attributes that allows this distinction to be made.
- The primary key of a weak entity set is formed by taking the primary key of the strong entity set on which its existence depends (see Mapping Constraints) plus its discriminator.

**Example:**
- *transaction* is a weak entity. It is existence-dependent on *account*.
- The primary key of *account* is *account-number*.
- *transaction-number* distinguishes transaction entities within the same account (and is thus the discriminator).
- So the primary key for *transaction* would be (*account-number, transaction-number*).
Primary Keys for Relationship Sets

• The key of a relationship set are the attributes that comprise the primary keys of the entity sets involved in the relationship set.

Example:

1. S.I.N. is the primary key of customer, and
2. account-number is the primary key of account.
3. The attributes of the relationship set custacct are then (account-number, S.I.N.).
4. This is enough information to enable us to relate an account to a person.
Primary Keys for Relationship Sets

• If the relationship has descriptive attributes, those are also included in its attribute set. For example, we might add the attribute *date* to the above relationship set, signifying the date of last access to an account by a particular customer.

• Note that this attribute cannot instead be placed in either entity set as it relates to both a customer and an account, and the relationship is many-to-many.

• The primary key of a relationship set depends on the mapping cardinality and the presence of descriptive attributes.
The E/R Diagrams

We can express the overall (data-oriented) model of an information system **graphically with an E-R diagram.**

Its components are:

- *rectangles* representing entity sets.
- *ellipses* representing attributes.
- *diamonds* representing relationship sets.
- *lines linking* attributes to entity sets and entity sets to relationship sets.
The Entity Relationship Diagram

Account

- balance
- Account number

log

Transaction

- Trans.number
- date
- amount

Transaction log diagram showing relationships between Account, log, and transaction entities.
Extended E-R Diagrams

Some of the variations are:

• **Diamonds being omitted (in the UML class-diagram) - a link between entities indicates a relationship.**
  1. Less symbols, clearer picture.
  2. What happens with descriptive attributes?
  3. In this case, we have to create an intersection entity to possess the attributes.

• **Numbers instead of arrowheads indicating cardinality.**
  1. Symbols, 1, n and m used.
  2. E.g. 1 to 1, 1 to n, n to m.
  3. Easier to understand than arrowheads.
Extended E-R Diagrams

*Multivalued* attributes may be indicated in some manner.
- Means attribute can have more than one value.
- E.g. hobbies.

Extended E-R diagrams allowing more details/constraints in the real world to be recorded.
- Composite attributes.
- Derived attributes.
- Subclasses and superclasses.
- Generalization and specialization.

• Roles in E-R Diagrams
Extended E-R Diagrams

• The function that an entity plays in a relationship is called its role. Roles are normally explicit and not specified.
• They are useful when the meaning of a relationship set needs clarification.

Example: The entity sets of a relationship may not be distinct. The relationship works-for might be ordered pairs of employees (first is manager, second is worker).
• In the E-R diagram, this can be shown by labelling the lines connecting entities (rectangles) to relationships (diamonds).
Extended E-R Diagrams

Roles in E-R Diagrams

Employee

function

name

manager

worker

Works for
Extended E-R Diagrams

- Weak entity sets in E-R diagrams
Nonbinary relationships

Customer

posses

Account

branch

name

Social security

Account-number

balance

Branch-name
Generalization

**Example:** Consider extending the entity set *account* by classifying accounts as being either
- *savings-account* or *checking-account*.

- Each of these is described by the attributes of *account* plus additional attributes. (*savings* has *interest-rate* and *chequing* has *overdraft-amount*.)
- We can express the similarities between the entity sets by generalization.

- In E-R diagrams, generalization is shown by a *triangle*.
- Generalisation hides similarity and emphasizes similarities.
  - Distinction is made through attribute inheritance.
  - Attributes of higher-level entity are inherited by lower-level entities.
Generalization

Account

IS-A

Saving-account

Checking-account

balance

Account number

Interest-rate

Overdraft-amount
Aggregation in E-R

- Worker
  - name
  - Social security
- Project
  - hours
  - number
- Machine
  - branch
  - Id

- works
Aggregation in E-R

Worker -- works --> Project

- name
- Social security
- hours
- number

Machine

- branch

Id