Data and Process Modelling

3. Object-Role Modeling - CSDP Step 3

Marco Montali

KRDB Research Centre for Knowledge and Data
Faculty of Computer Science
Free University of Bozen-Bolzano

A.Y. 2014/2015

Trimming and Finding Derivations

CSDP Step 3

Check for entity types that should be combined; note any arithmetic derivations.

Refine the conceptual schema diagram answering to:

1. Can the same entity belong to two entity types?
2. Can entries of two different types be meaningfully compared? Do they have the same unit/dimension?
3. Is the same kind of information recorded for different entity types, and will you ever need to list the entities together for this information?
4. Is a fact type arithmetically derivable from others?

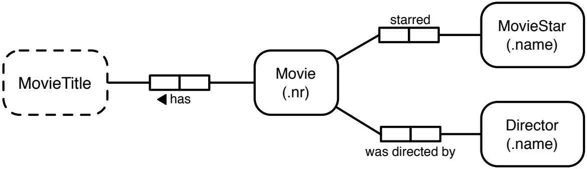
Two possible actions:

- Combination of entities type in a unique type;
- Derivation rule to connect different (related) fact types.

Partitioning of the UoD

- UoD is partitioned into exclusive and exhaustive slices:
  - Values;
  - Entities.
- Entities are again partitioned into primitive entity types: top-level entities never overlap.
- Top-level entity: not subtype of another entity.
- Subtyping: classification of objects into a more specific type.
  - Will be discussed later on in the course.
  - If object type A is subtype of object type B, then every instance of A is also instance of B (set inclusion).
  - Represented by a solid arrow in ORM notation.
  - Subtypes could overlap!
- Top-level values could overlap.
  - 'Indiana' is the name of a US state and the first name of a fictional character.
- Subtyping of values is rarely used.

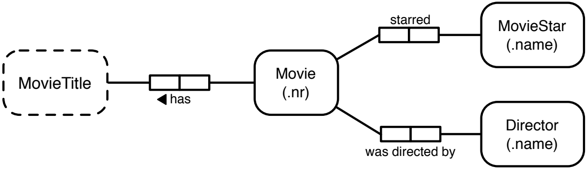
Analysis of Separate Entities: Overlapping Instances



Can the same entity belong to two entity types?

- MovieStar and Director: top-level object types → non-overlapping → no Director can be a MovieStar.
- Is this reasonable?

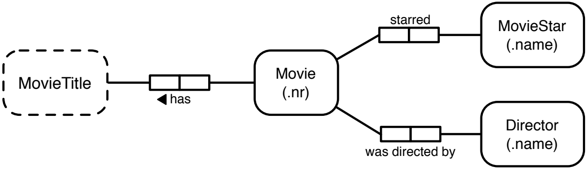
Analysis of Separate Entities: Overlapping Instances



Can the same entity belong to two entity types?

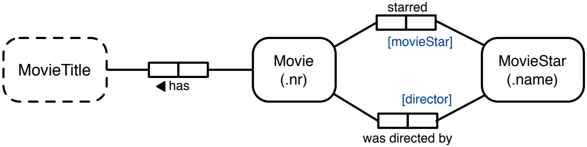
- MovieStar and Director: top-level object types → non-overlapping → no Director can be a MovieStar.
- Is this reasonable?
- Consider now the case of Alfred Hitchcock

Analysis of Separate Entities: Overlapping Instances



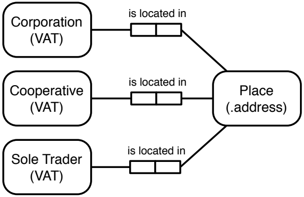
Can the same entity belong to two entity types?

- MovieStar and Director: top-level object types → non-overlapping → no Director can be a MovieStar.
- Is this reasonable?
- Consider now the case of Alfred Hitchcock → there is an overlap → combination of the object type.



Analysis of Separate Entities: Queries

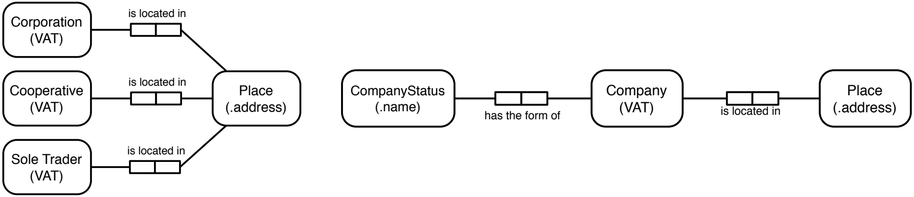
- Is the same kind of information recorded for different entity types?
- Do we need to list the entities together for this information?



List all companies and their location.
List all companies located in '. . . '.

Analysis of Separate Entities: Queries

- Is the same kind of information recorded for different entity types?
- Do we need to list the entities together for this information?

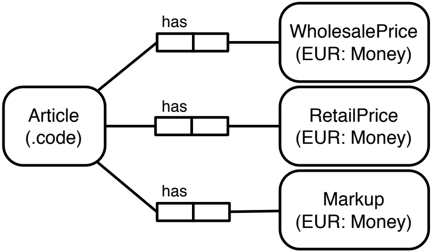


List all companies and their location.

List all companies located in '. . . '.

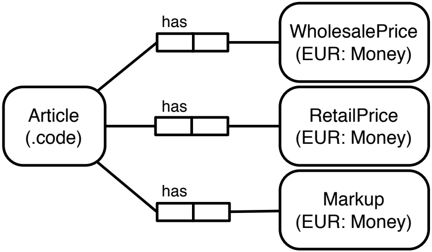
Analysis of Separate Entities: Units

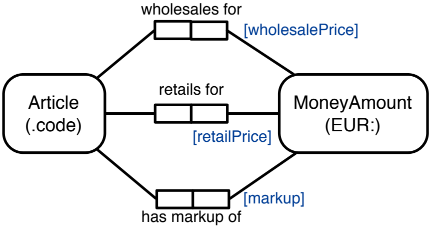
- Can entries of two different types be meaningfully compared? Do they have the same unit/dimension?
- Entities with same **unit-based** reference mode can be meaningfully compared and combined.



Analysis of Separate Entities: Units

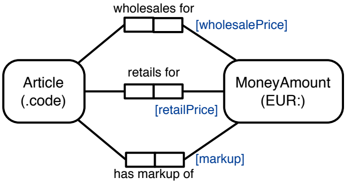
- Can entries of two different types be meaningfully compared? Do they have the same unit/dimension?
- Entities with same **unit-based** reference mode can be meaningfully compared and combined.





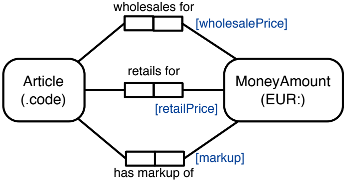
Discovering Arithmetic Constraints

Is a fact type arithmetically derivable from others? (making arithmetic constraints explicit)



Discovering Arithmetic Constraints

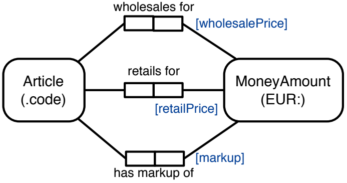
Is a fact type arithmetically derivable from others? (making arithmetic constraints explicit)



$$markup = retailPrice - wholesalePrice$$

Discovering Arithmetic Constraints

Is a fact type arithmetically derivable from others? (making arithmetic constraints explicit)

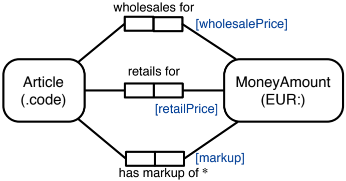


$$markup = retailPrice - wholesalePrice$$

- Derived type: type completely determined by other types. They must obey to a *constraint*.
- May be conceptually relevant to keep derived types in the conceptual diagram.
- Two decoration symbols for derived fact types:
  1. derived ($*$) vs semi-derived ($+$);
  2. derived-on-query vs derived-on-update ($*$).
- Controlled textual annotation to represent the derivation constraint.

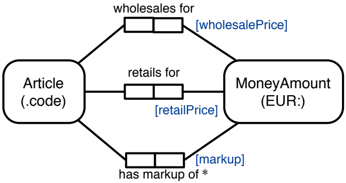
Derivation vs Semi-Derivation

- Derivation: a commitment is taken on how to interpret the constraint.
  - Fixed inputs.
  - Fixed output (derived type).
  - Typical case.



Derivation vs Semi-Derivation

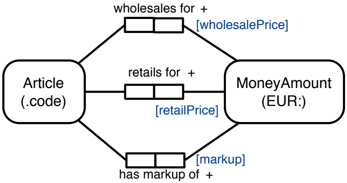
- Derivation: a commitment is taken on how to interpret the constraint.
  - Fixed inputs.
  - Fixed output (derived type).
  - Typical case.



- Constraints can be interpreted in different ways.

$$markup = retailPrice - wholesalePrice$$
$$retailPrice = markup + wholesalePrice$$
$$wholesalePrice = retailPrice - markup$$

- What about keeping different possible derivation policies?
  - Semi-derivation: many uses of the same constraint to derive multiple types from each other.



Derivation Rule

Constraint telling how a fact type is derived from other fact types.

- Context of the constraint.
  - Globally identified in the constraint (e.g., `Article`).
  - Locally identified: dot notation (e.g., `Article.markup`) vs of-notation (e.g., `markup of Article`).
- Attribute style: uses role names.
- Relational style: uses predicate readings.

Derivation Rule

Constraint telling how a fact type is derived from other fact types.

- Context of the constraint.
  - Globally identified in the constraint (e.g., `Article`).
  - Locally identified: dot notation (e.g., `Article.markup`) vs of-notation (e.g., `markup of Article`).
- Attribute style: uses role names.

Example (attribute style)

```
for each Article, markup = retailPrice - wholesalePrice
```

- Relational style: uses predicate readings.

Derivation Rule

Constraint telling how a fact type is derived from other fact types.

- Context of the constraint.
  - Globally identified in the constraint (e.g., `Article`).
  - Locally identified: dot notation (e.g., `Article.markup`) vs of-notation (e.g., `markup of Article`).
- Attribute style: uses role names.

Example (attribute style)

```
for each Article, markup = retailPrice - wholesalePrice
```

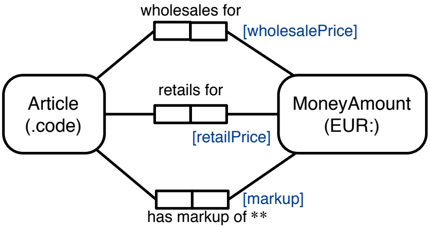
- Relational style: uses predicate readings.

Example (relational style)

```
Article has markup of MoneyAmount iff
   Article retails for MoneyAmount$_1$ and
   Article wholesales for MoneyAmount$_2$ and
   MoneyAmount = MoneyAmount$_1$ - MoneyAmount$_2$
```

Storage of Derived Facts

- Derived-on-query (lazy evaluation): derived information is computed on request.
  - Typically part of a *view* of the conceptual model.
- Derived-on-update (eager evaluation): derived information is stored.
  - Another $*$ added.
  - Every time one of the primitive facts is updated, the derived fact must be updated too.
  - In databases: trigger or computed column.


wholesales for
[wholesalePrice]
Article
(.code)
retails for
[retailPrice]
MoneyAmount
(EUR:)
[markup]
has markup of **