**Notion Theory**

**Michel Böhms (TNO), 03-06-2024**

Nowadays we see a lot of data specifications like schemas and ontologies/object type libraries that should define the meaning of (‘individual’/’instance’) data on real world phenomena. This, on its turn, should improve the data exchange and/or data sharing between software and ultimately the people using that software.

These data specifications typically contain types for concrete or abstract things (denoted as concepts, entities, element types, classes, …), their attributes and interrelationships. They often also contain constraints that are rules that should be respected for the data to be valid. These rules can come from ontological definitions, user requirements, regulations and/or recommendations.

Since typically we specify the link between individuals and a concepts ourselves (in linked data you would use the ‘rdf:type’ relationship for that) these types can be anything. Often they are fuzzy and ill-defined. The fuzziness is sometimes even misused by modellers (bridges become special kinds of buildings in older bSI IFC Open BIM models like IFC 2x3 that do not yet have a bridge concept). That causes a lot of transformation trouble between multiple views/specifications especially in exchange scenarios where data is lost that way.

Note that properties and relations have a more formal nature. Typically the value/reference ranges are clear and the only fuzzy thing might be the handling of unknown data (resulting in CWA or OWA interpretations). One way to make properties (especially quantities) more formal is the indication of their underlying quantity kind (basic like ‘length’ or ‘mass’ or as algebraic combination of basic quantities kinds represented by ‘dimensions’).

So what can we do about it? The solutions lies in the idea of “necessary and sufficient” properties. That is: each defined (“asserted”) class should be equivalent to a class that is 100% defined by a set of discriminators, or better: discriminating constraints (as valued/constraint properties). Equivalence means “being subclass in two directions” so: any individual of the asserted class is an instance of the equivalent class (necessary) and the other way round (sufficient).

The sufficient part tells you: if I meet a phenomena and it complies to all those constraints I can automatically also classify it as the original class. You can call it a “derived class” now. The motto now becomes: for the best modelling result only allow “derived classes”!

In other words, instead of randomly introducing classes you like, you have to think first and really try to get to the essence of the concept you want to introduce. That is often tough (it is much more easy to just copy the classes people already use in some domain) but the rewards are high: you obtain a deep semantic specification which can be easily mapped to other (equally deep semantic) specifications. Actually the mapping between views comes down to mapping of properties and relations which are way more concrete as discussed before. And there is a second, quite important, advantage: once you model the essence of something the results are typically more generic. Instead of introducing one or some random classes you implicitly introduce a whole set of potentially relevant (now or for a future extension) classes.

In short: we go from concept-first to property-first (and concept-derived) modelling.

Personally I think the above feels quite obvious but at the same time almost nobody is doing it or even aware of the issue. The first and so far only account that really made this approach explicit is “Notion Theory” by Wim Gielingh. A ‘notion’ is the (valued) observation made in the context of a perceptive frame (“view”). The property without value is referred to as the Notion Frame. The default interpretation is OWA here, if you ‘don’t know’, there is per definition ‘no notion’. Note that “valued” not necessarily means “having only one specific value”; it can be a more flex constraint indicating a set or range of allowed values.

Related to that, Peter Willems has implemented prototype software tooling supporting these ideas in a JSON/GraphQL context.

A typical example:

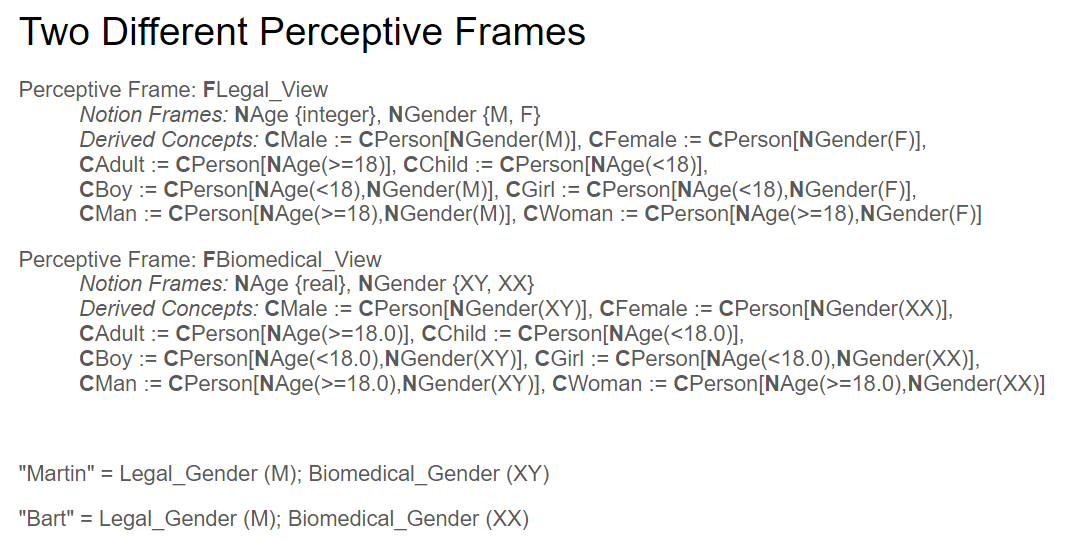


Figure 1: Formal specification

Would result in the GraphQL implementation:

enum GenderByDNA {

XX

XY

}

enum LegalGender {

M

F

}

enum DerivedConcept {

LEGAL\_ADULT

LEGAL\_BOY

LEGAL\_CHILD

LEGAL\_FEMALE

LEGAL\_GIRL

LEGAL\_MALE

LEGAL\_MAN

LEGAL\_WOMAN

BIOMEDICAL\_ADULT

BIOMEDICAL\_BOY

BIOMEDICAL\_CHILD

BIOMEDICAL\_FEMALE

BIOMEDICAL\_GIRL

BIOMEDICAL\_MALE

BIOMEDICAL\_MAN

BIOMEDICAL\_WOMAN

PERSON

}

input LegalViewInput {

legalAge: Int!

legalGender: LegalGender!

}

input BiomedicalViewInput {

biomedicalAge: Float!

biomedicalGender: GenderByDNA!

}

interface View {

name: String!

}

type LegalView implements View {

name: String!

legalAge: Int!

legalGender: LegalGender!

}

type BiomedicalView implements View {

name: String!

biomedicalAge: Float!

biomedicalGender: GenderByDNA!

}

type Person {

id: ID!

views: [View!]!

}

type PersonMutationResponse {

"Similar to HTTP status code, represents the status of the mutation"

code: Int!

"Indicates whether the mutation was successful"

success: Boolean!

"Human-readable message for the UI"

message: String!

"Creation result"

person: Person!

}

type Query {

concepts(personId: ID!): [DerivedConcept!]!

isConcept(personId: ID!, concept: DerivedConcept!): Boolean!

}

type Mutation {

createPerson(personId: ID!): PersonMutationResponse!

addLegalView(personId: ID!, view: LegalViewInput!): PersonMutationResponse!

addBiomedicalView(personId: ID!, view: BiomedicalViewInput): PersonMutationResponse!

}

**Open Issues**

* Relation modelling in notion theory?
* More complex constraints (other than =, <, >)
* Derived attributes?
* Subproperties relevant?
* Formal property modelling? (always indicate quantity kinds?)
* Taxonomy (not relevant anymore, just set of ‘bootstrap’ root classes + notions)
* Meronomy? (aka ‘typical decomposition’relations, still relevant?)
  + Related to functional/technical (configuration) aspects
* Views currently not in modelling, not better explicit? (like separate/linked specs)
  + Do we have some theoretical counterpart? Notion: used for specialization, x?, used for decomposition (compare or versus and logic)
* **How to combine with current practice (here bSI+LD)**