Ontology-Based Generation of Data Platform Assets

Vincenzo De Leo, Gianni Fenu
Department of Math and Computer Science University of Cagliari
Cagliari, Italy

Enrico Motta
Knowledge Media Institute Open University
Milton Keynes, UK

Andrea Giovanni Nuzzolese
Institute of Cognitive Sciences and Technologies National Council of Research
Bologna, Italy

Diego Reforgiato Recupero
Department of Math and Computer Science University of Cagliari
Cagliari, Italy

Francesco Osborne
Knowledge Media Institute Open University
Milton Keynes, UK

David Greco, Nicolò Bidotti, Paolo Platter
Big Data Laboratory AgileLab s.r.l. Rome, Italy

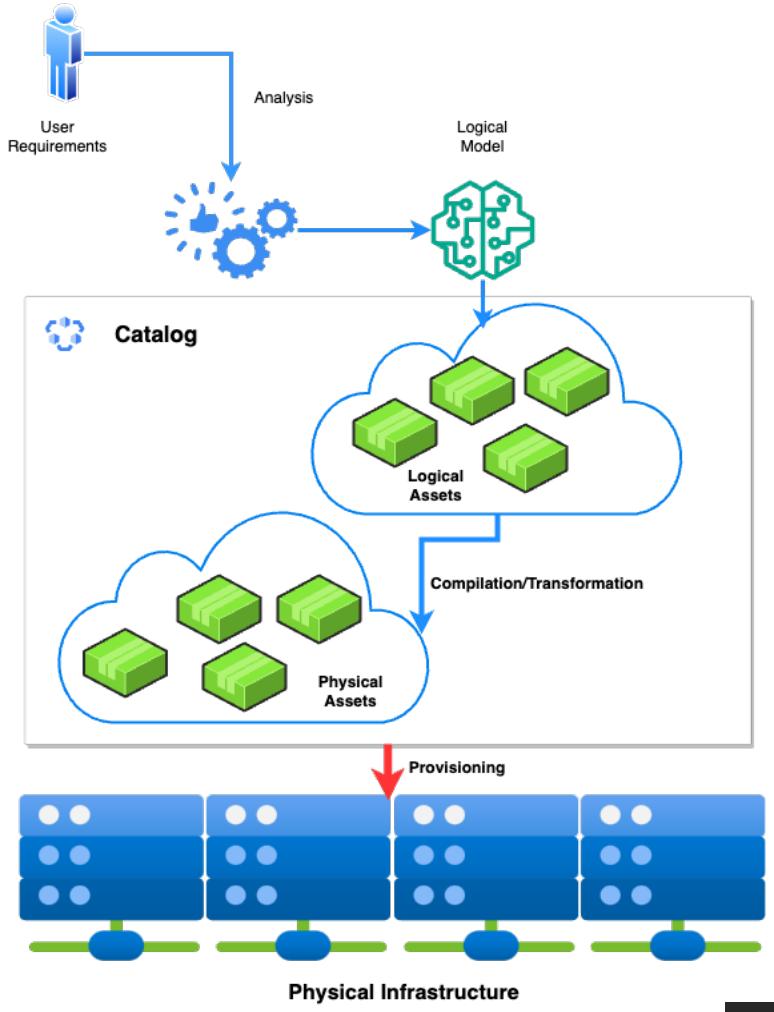
The Problem

- Designing a modern data platform is a very complex integration task.
- Many technologies must be integrated: storage technologies, computational platforms, and data catalogs.
- All those technologies must be integrated to provide a smooth user experience, avoid unauthorized data access, and ensure strict auditability of data access.
- Most of the time, companies build their data platform starting from the available technology without defining a logical model to keep separating the user requirements from the specific technological concerns.
- This lack of abstraction tends to produce very rigid data platforms locked to very specific technologies and extremely difficult to adapt to the evolution of technology.
- Designing a data platform conceived to last ten years is almost impossible.

The Solution: back to the fundamentals

- We propose a new approach that adopts a correct software engineering process, starting with user requirements.
- The user requirements in the context of a data platform are about how the data needs to be organized, kept secure, and usable by the users.
- The data platform user requirements tend to describe the data containers more than the data itself.
- The user requirements analysis includes a list of data platform assets like data collections, contracts, and access rules. All these assets must be defined, described, and stored in a flexible catalog.

The Solution: enabling the process through a flexible catalog



- An asset catalog is fundamental to keeping track of all the assets and their relationships.
- Any application that governs a complex data platform has an asset catalog as one of its fundamental components.
- A simple process:
  1. User requirements gathering.
  2. Analysis and logical model defined.
  3. Definition of the logical assets inside the catalog.
  4. The compilation/transformation rules define the corresponding physical assets inside the catalog.
  5. Definition of the provisioning tasks for deploying the physical assets on the target technologies.

The Solution: a flexible catalog

- To support our process, we need a flexible catalog able to:
  1. Allow users to define types together with a schema.
  2. Mark a user-defined type with “traits” to describe its kind and behavior.
  3. Strict separation between the shape/form of a type and the relationships among each other. In a data platform, the managed assets are mainly characterized by their relationships.

The Solution: a flexible catalog

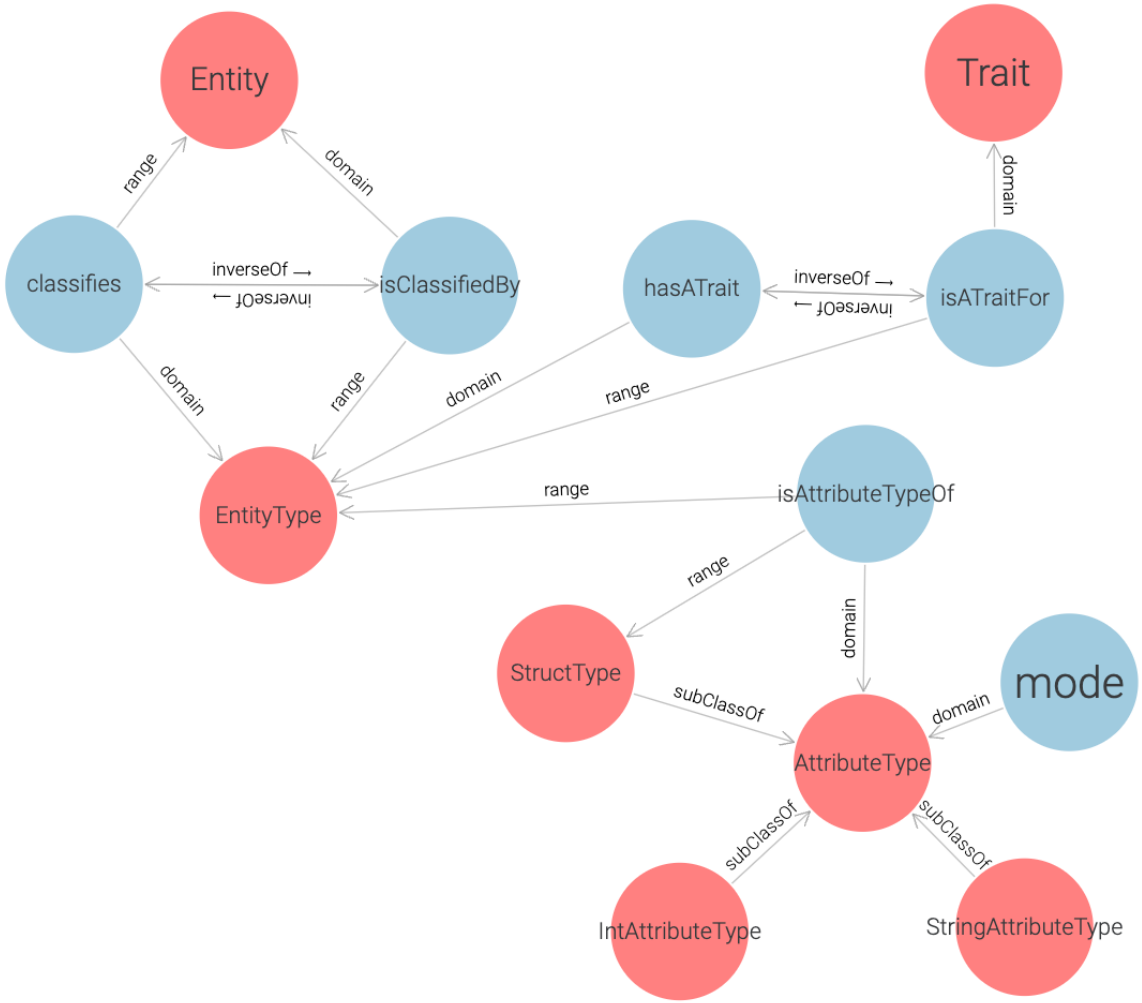
- We designed our catalog system based on an ontology structured into four layers (L0, L1, L2, L3).
  - L0 provides the mechanism for creating user-defined types.
  - All the types will be defined in L2, which is used as a container for all the possible user-defined types.
  - Each user-defined type is associated with a name and a schema
    - A schema is a list of pair names/types.
    - A type can also be structured like a list, structure, or option (optional attribute). It supports an arbitrary level of nesting.
  - L1 describes all the traits corresponding to the asset kinds.
    - A trait can be used to represent the behavior of a user-defined type.
    - L1 defines all the possible structural relationships among traits.
    - These will be automatically inherited by all the instances implementing those traits.
  - L3 contains all the user-defined type instances.

The Solution: a flexible catalog

- We built a prototype and put it under an Apache 2 open-source license into a GitHub repository:
  https://github.com/agile-lab-dev/data-platform-shaper
  - We used an RDF-based knowledge graph, and on top of it, we built a Scala3-based library and a microservice exposing REST APIs.
  - The knowledge graph is initially fed with the L0 and L1 ontologies. All the mechanisms for defining traits, types, type instances, trait relationships, etc., are exposed by the library and by the REST APIs.

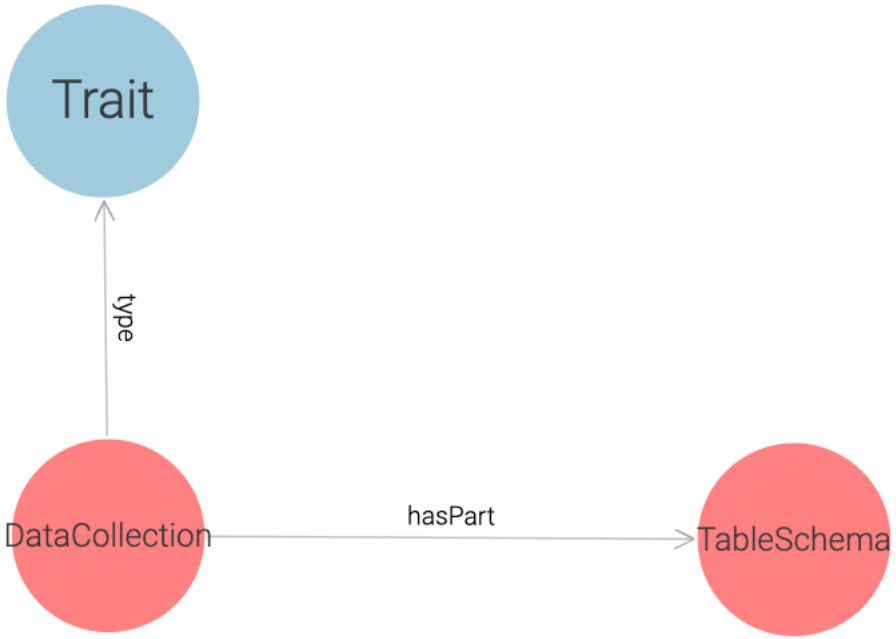
The Solution: an example

- L0: the basic machinery



The Solution: an example

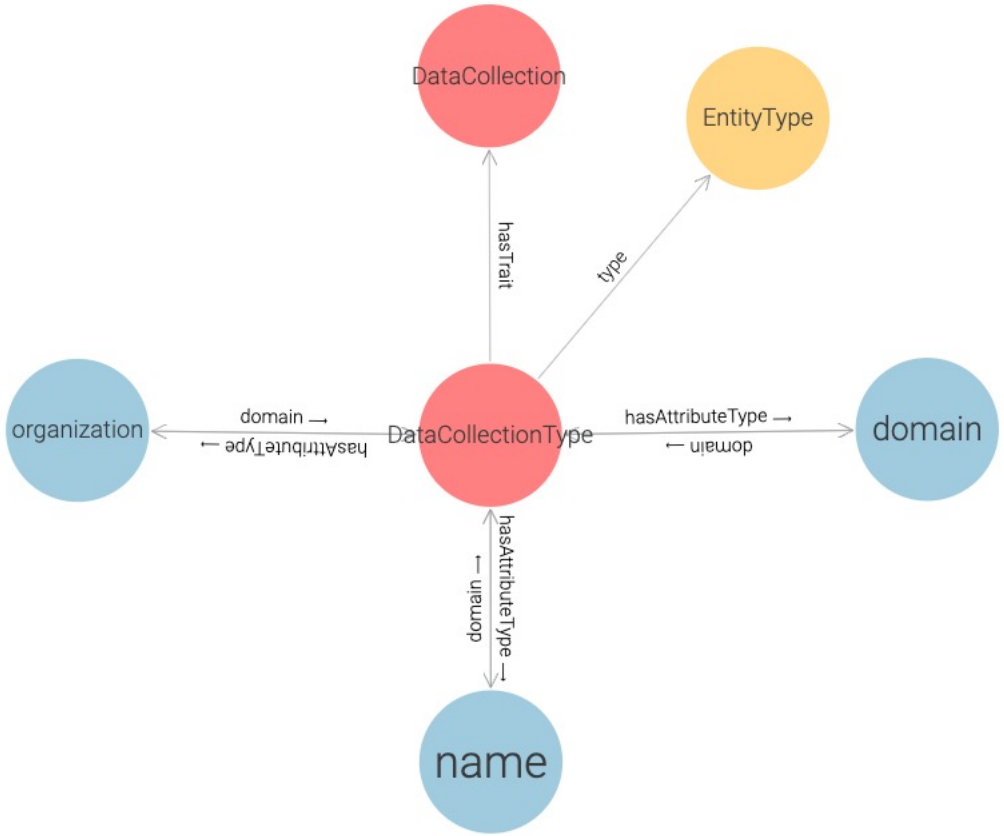
- L1: Traits & Relatioships



The Solution: an example

- L2: User-defined Types

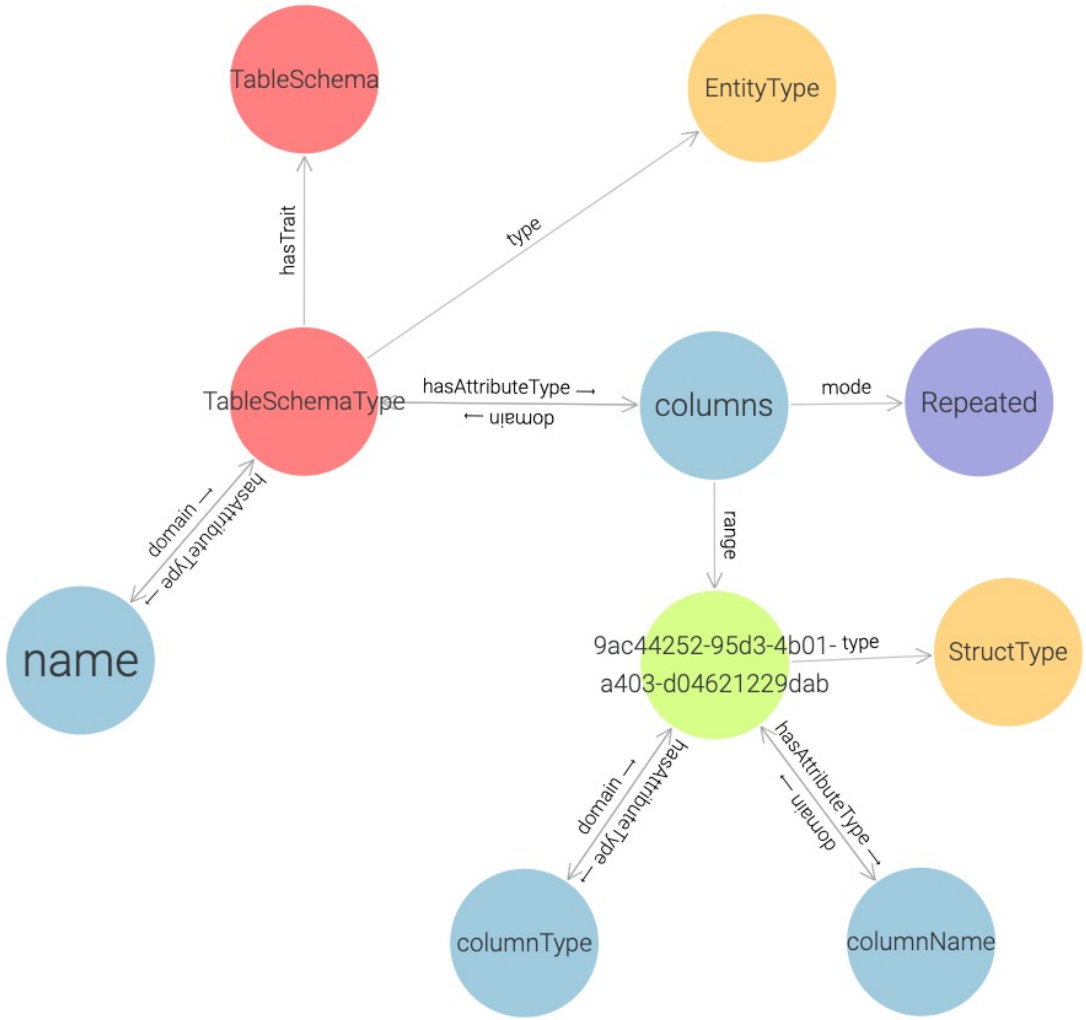
name: DataCollectionType
traits:
- DataCollection
schema:
- name: name
  typeName: String
  mode: Required
- name: organization
  typeName: String
  mode: Required
- name: domain
  typeName: String
  mode: Required



The Solution: an example

- L2: User-defined Types

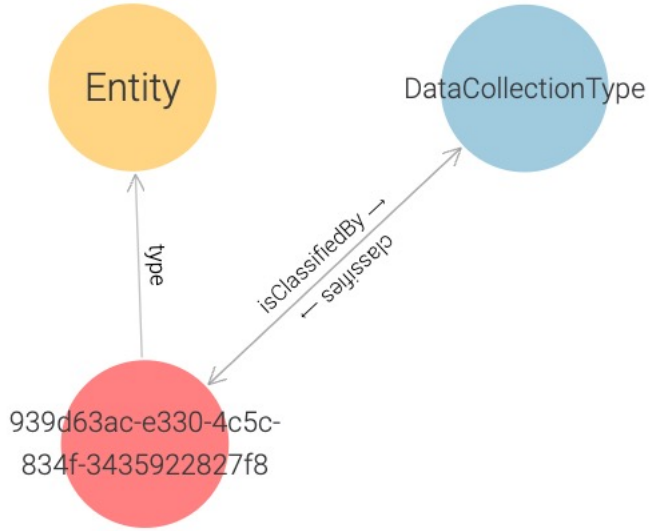
name: TableSchemaType
traits:
- TableSchema
schema:
- name: tableName
  typeName: String
  mode: Required
- name: columns
  typeName: Struct
  mode: Repeated
  attributeTypes:
    - name: columnName
      typeName: String
      mode: Required
    - name: columnType
      typeName: String
      mode: Required



The Solution: an example

- L3: User-defined Types Instances

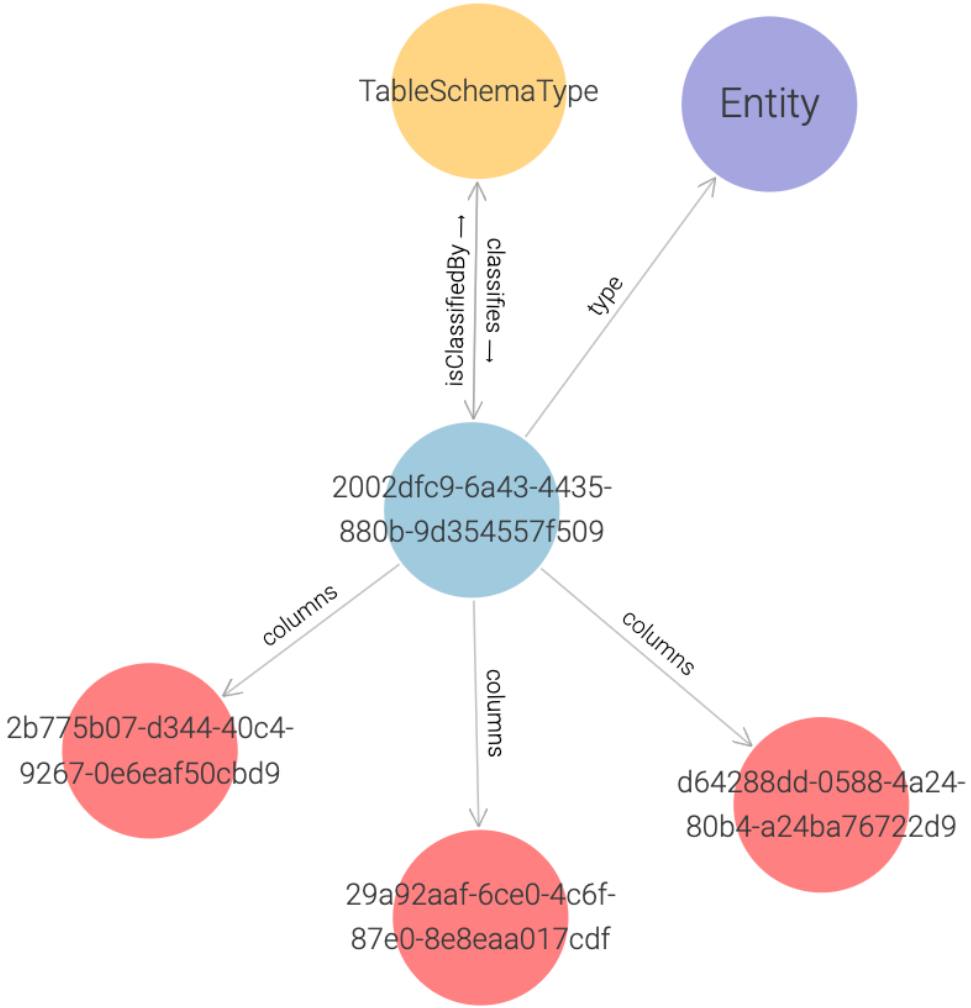
entityTypeName: DataCollectionType
values:
  name: Person
  domain: Registrations
  organization: HR



The Solution: an example

- L3: User-defined Types Instances

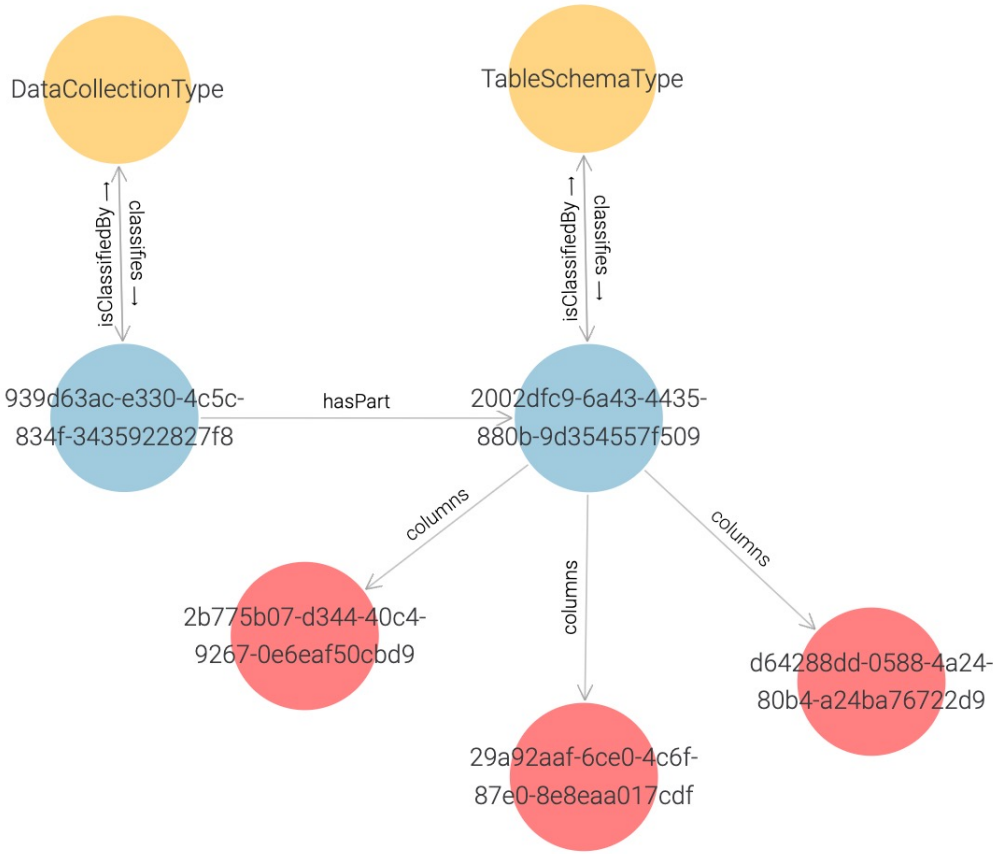
entityTypeName: TableSchemaType
values:
  name: Person
  columns:
  - columnName: firstName
    columnType: String
  - columnName: familyName
    columnType: String
  - columnName: age
    columnType: Int

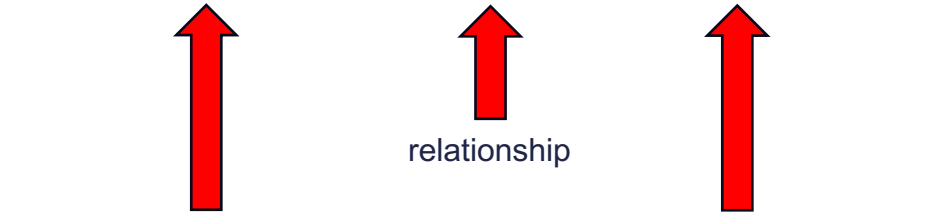


The Solution: an example

- L3: Linking instances

```
curl -X 'POST' \
'http://127.0.0.1:8093/dataplatform.shaper.uservice/0.0/ontology/entity/link/939d63ac-e330-4c5c-834f-3435922827f8/hasPart/2002dfc9-6a43-4435-880b-9d354557f509'
```





The Solution: next

- Improving the overall system expressiveness.
- Adding search functions.
- Instance validation.

Thank you!