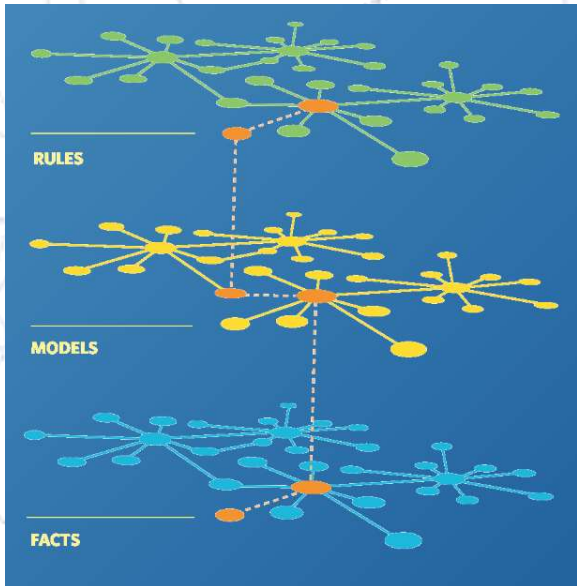


Getting Started with Knowledge Graphs: Main Challenges and How to Overcome Them



FOUNDATION

- TopQuadrant was founded in 2001
- Strong commitment to standards-based approaches to data semantics

MISSION

- Empower people and drive results — by making enterprise information meaningful



FOCUS

- Provide comprehensive data governance solutions using knowledge graph technologies

CUSTOMERS

- Over 120 active customer organizations

Today's Agenda

✓ *Knowledge Graphs are growing in importance and adoption*

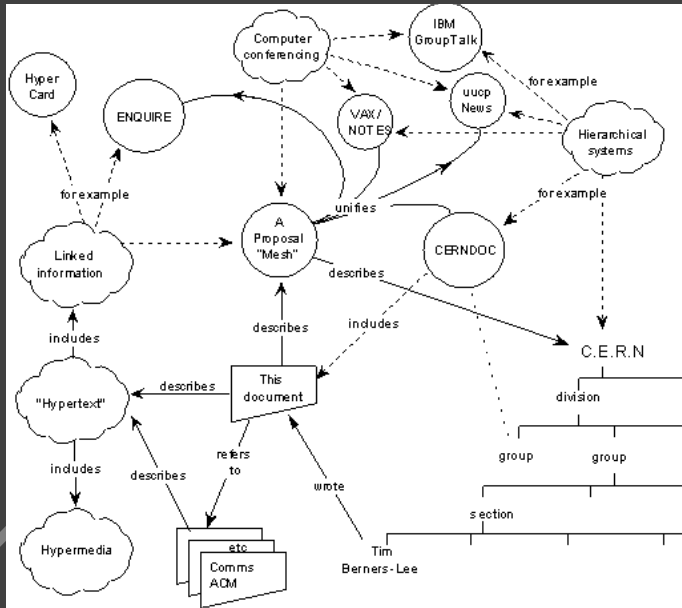
What are some key challenges and best practices to using them?

1. Define URIs – URI design
2. Create, define, use ontology models
3. Ensure that developers appreciate / like to use RDF
4. Capture statements about statements
5. Take advantage of effective ways of doing reasoning or inferencing
6. Use task appropriate visualizations



Irene Polikoff

2001
Tim Berners-Lee



2012
Google

2019-2020

Knowledge Graphs in the news

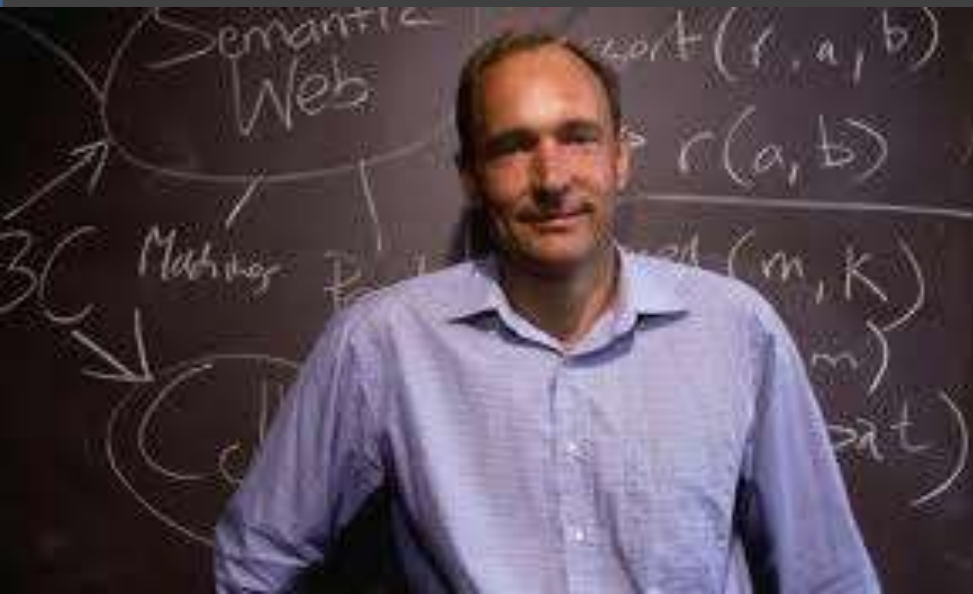
> Google on AI and knowledge graphs in ZDNET

> Data governance 2.0 on Dataversity

> Conferences & workshops

> Technology trends 2019

> Forbes article by Kurt Cagle



Enter Knowledge Graphs

- A Knowledge Graph represents a knowledge domain
- It represents knowledge as a graph
 - A network of nodes and links
 - Not tables of rows and columns
- It represents facts (data) and models (metadata) in the same way
 - Rich rules and inferencing
- It is based on open standards, from top to bottom
 - Readily connects to knowledge in private and public clouds



RULES: If both of a person's parents have blue eyes, they will also have blue eyes



MODELS: A person has eye color. A person has two parents. A person's father is also a person and he is male.

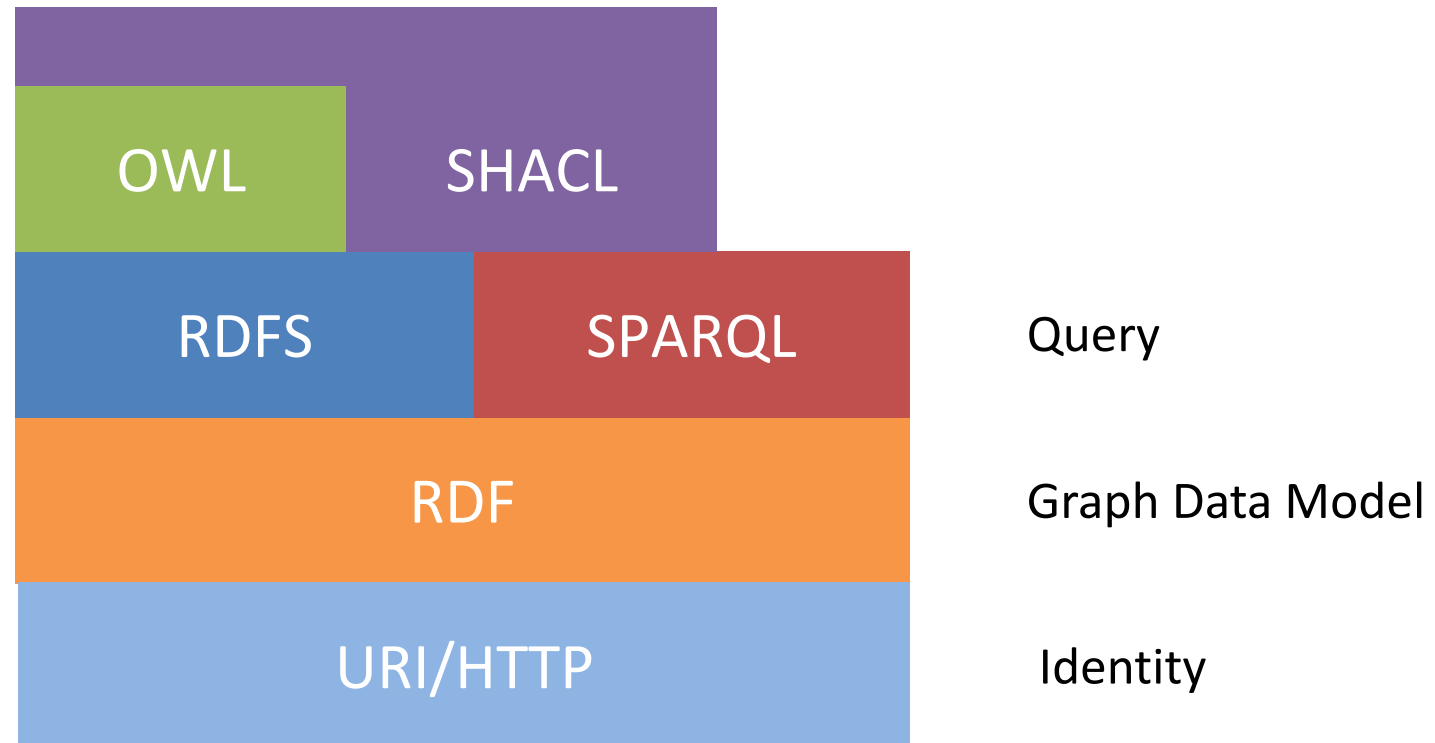


FACTS: James has blue eyes. James' father is Andrew. James is a person.

There can be different types and instances of Knowledge Graphs ...

Standards-Based

Domain Specific
Models and Rules
aka Ontologies



Reflecting on the Past

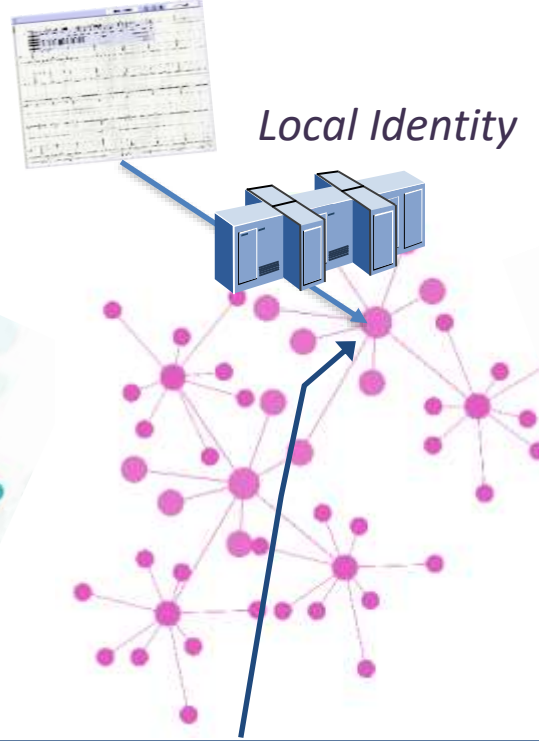
- Some RDF-based standards and technologies have been available for over 20 years now
 - RDF, RDFS, SPARQL
- During this period, the standards evolved and matured
 - as did best practices
 - as did our understanding of what works and what does not
- This presentation is informed by working with many industries and customers over the last 20 years
 - as they learned about ways to model with **fitness-for-purpose** and applied knowledge graph technologies in **hybrid technology stacks**

Challenge 1: Why URIs are Valuable

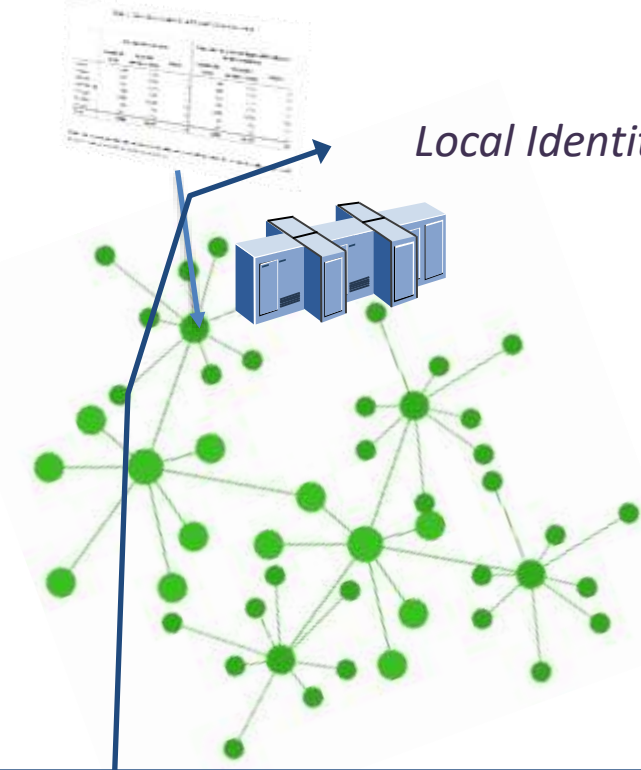
Local Identity



Local Identity



Local Identity



“Identity Matters”

For information assets as well as classes, attributes, relationships and rules that describe them

RDF – the “escape route” for Assets locked in silo repositories



Assets in the enterprise intranet

Challenge 1: Defining URIs

- URI Practices Converged On:
 - Use Labels
 - Most suitable for models – classes, properties
 - Can be suitable for controlled vocabularies
 - Use a System Generated UUID
 - Best for data in general
 - Numeric UUID are often required for integration
 - Use a “Primary Key” – a property with unique values
 - Best for reference data
 - Ensures integrity

TopBraid EDG Enterprise Data Governance + Data Graphs Global Lookup Hello, Irene.Polikoff

Create New Data Graph

Data graphs are used to represent arbitrary instance data.

This creates a new Data Graph with yourself as the manager.

Label:

Default Namespace:

Description:

Include this asset collection in the index for Search the EDG: Enable

URI Class Prefix:

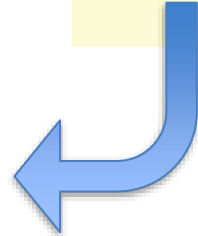
URI Construct Method:

- Default
- label
- uuid
- counter
- custom

User Cannot Modify URI:

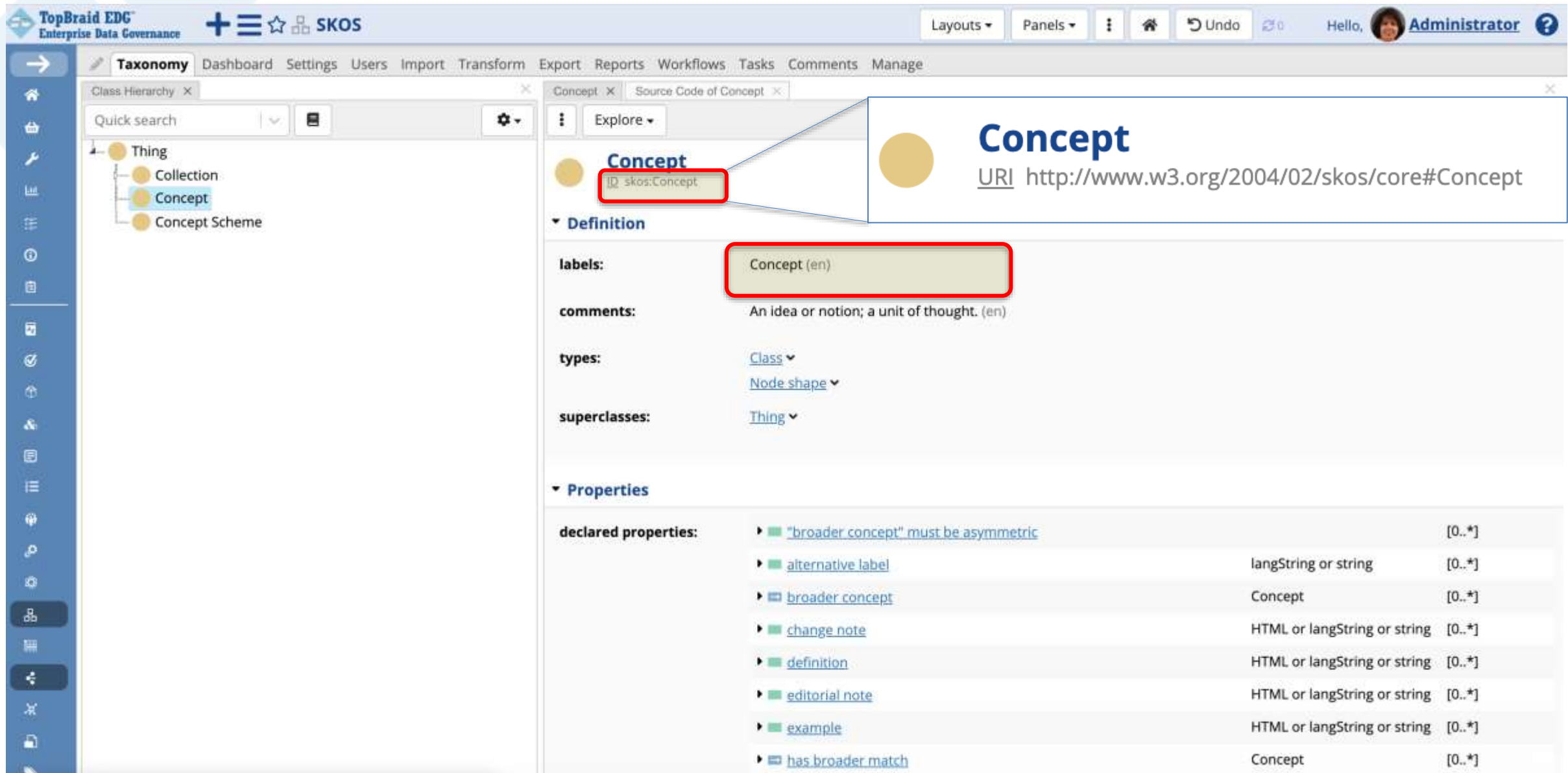
Includes:

- Use Labels
- Use a System Generated UUID
- Use a “Primary Key”- a property which values provide uniqueness (declared in an ontology)



Note: Construct methods that use Counter/UUID are not used here. (Users that use these methods should be identified in the Questionnaire below)

SKOS: Example of using labels to generate URIs



The screenshot shows the TopBraid EDC interface for SKOS. On the left, a 'Class Hierarchy' panel shows a tree structure: Thing (parent), Collection (child), Concept (child), and Concept Scheme (child). The 'Concept' node is selected. The main panel displays the details for the 'Concept' class. A red box highlights the 'ID' field, which contains 'skos:Concept'. A callout box points to this ID, showing a yellow circle icon, the label 'Concept', and the URI 'http://www.w3.org/2004/02/skos/core#Concept'. Below the ID, the 'labels' section shows 'Concept (en)' highlighted with a red box. Other sections include 'comments' (An idea or notion; a unit of thought. (en)), 'types' (Class, Node shape), 'superclasses' (Thing), and 'Properties' (declared properties).

TopBraid EDC Enterprise Data Governance SKOS

Layouts Panels Undo Hello, Administrator

Taxonomy Dashboard Settings Users Import Transform Export Reports Workflows Tasks Comments Manage

Class Hierarchy x Quick search

- Thing
 - Collection
 - Concept
 - Concept Scheme

Concept x Source Code of Concept x Explore

Concept
URI <http://www.w3.org/2004/02/skos/core#Concept>

Definition

labels: Concept (en)

comments: An idea or notion; a unit of thought. (en)

types: [Class](#) [Node shape](#)

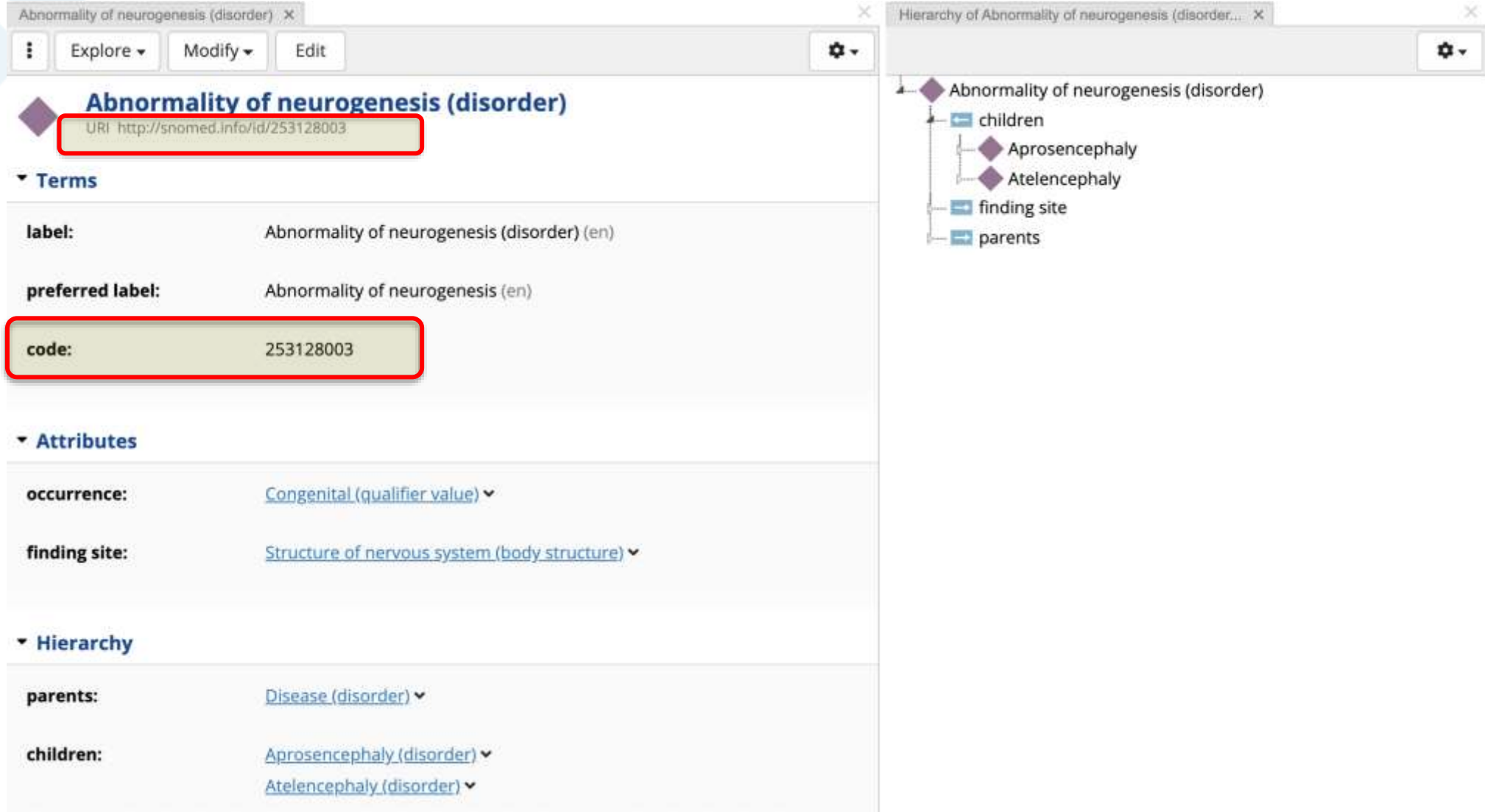
superclasses: [Thing](#)

Properties

declared properties:

Property	Domain	Cardinality
"broader concept" must be asymmetric		[0..*]
alternative label	langString or string	[0..*]
broader concept	Concept	[0..*]
change note	HTML or langString or string	[0..*]
definition	HTML or langString or string	[0..*]
editorial note	HTML or langString or string	[0..*]
example	HTML or langString or string	[0..*]
has broader match	Concept	[0..*]

SNOMED: Example of using values of “code” property to generate URIs



The screenshot displays two panels from the SNOMED CT interface. The left panel shows the details for the concept 'Abnormality of neurogenesis (disorder)'. The right panel shows a hierarchy view of the same concept.

Abnormality of neurogenesis (disorder)
URI <http://snomed.info/id/253128003>

Terms

label:	Abnormality of neurogenesis (disorder) (en)
preferred label:	Abnormality of neurogenesis (en)
code:	253128003

Attributes

occurrence:	Congenital (qualifier value)
finding site:	Structure of nervous system (body structure)

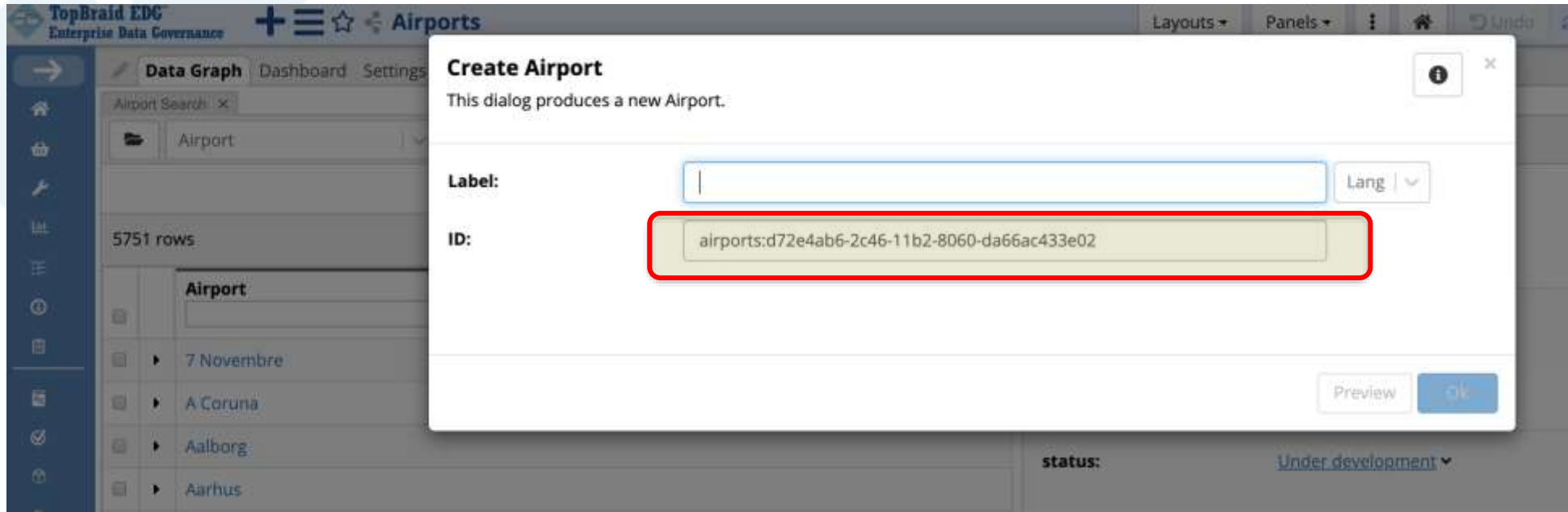
Hierarchy

parents:	Disease (disorder)
children:	Aprosencephaly (disorder) Atelencephaly (disorder)

Hierarchy of Abnormality of neurogenesis (disorder...)

- Abnormality of neurogenesis (disorder)
 - children
 - Aprosencephaly
 - Atelencephaly
 - finding site
 - parents

URI Construction Rules and UUIDs



URI Construction Rules

Determines the default construction rules for URIs for this collection.

URI Class Prefix:

Default

URI Construct Method:

uuid

User Cannot Modify URI:

Default

Challenge 2: Creating Ontology Models

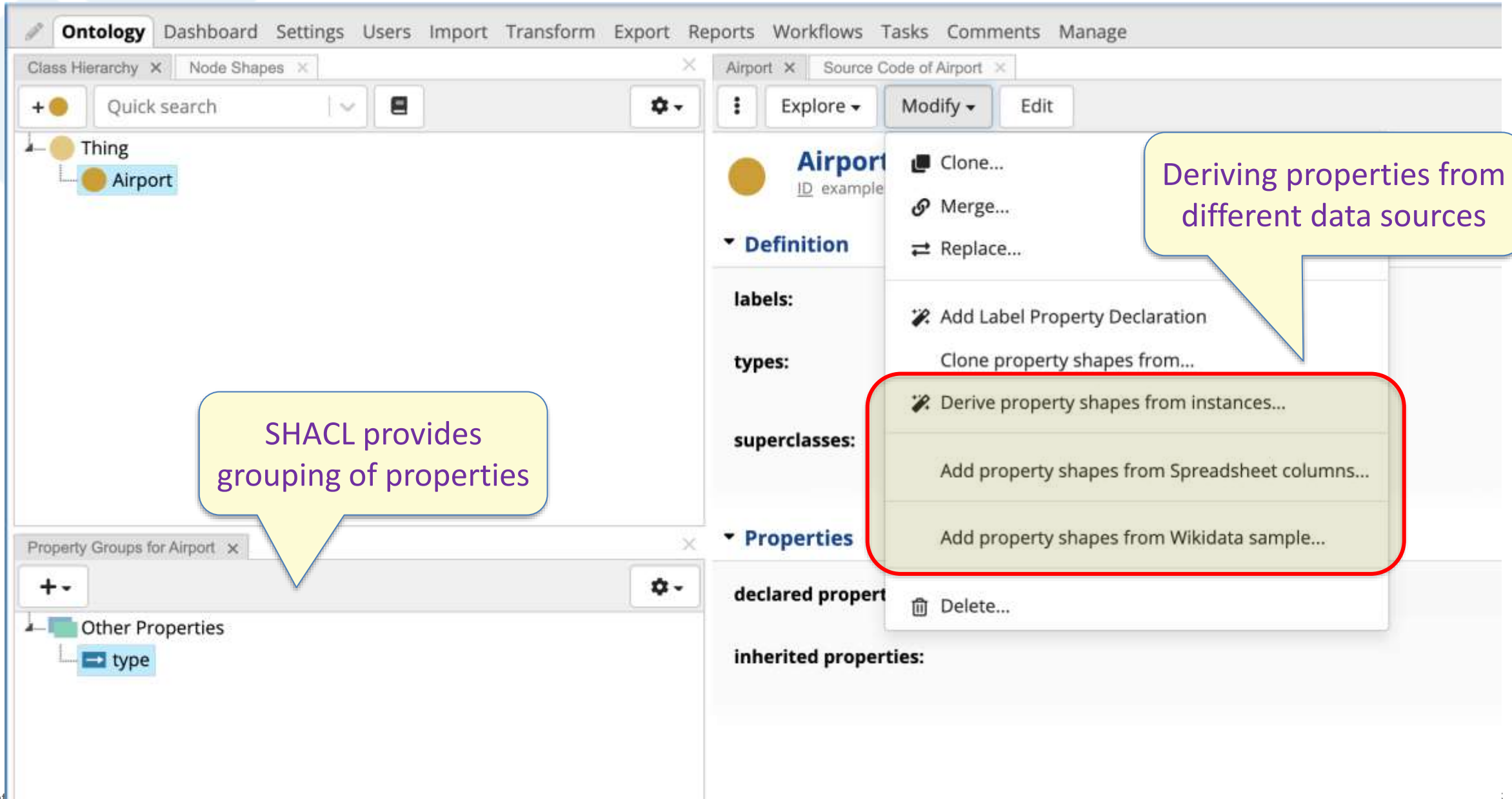
- **Not Effective:**

- Modeling in abstract, separate from data and how it will be used
- Once created, users have serious issues with using these models

- **Best Practice**

- Understand purposes of the ontology – they drive the design
- Use your Data to test the salience and quality of the models
- Harmonize, iterate and align as necessary
- Develop approaches for reuse of models

Example of Lifting a Model from Data



The screenshot shows the TopQuadrant ontology editor interface. The main window is titled "Ontology" and contains several panes:

- Class Hierarchy:** Shows a tree structure with "Thing" as the root and "Airport" as a child class.
- Property Groups for Airport:** Shows a tree structure with "Other Properties" as the root and "type" as a child property.
- Airport Details:** Shows the "Definition" and "Properties" sections for the "Airport" class. The "Properties" section is expanded, showing a list of options for adding property shapes.

A context menu is open over the "Properties" section, listing several options:

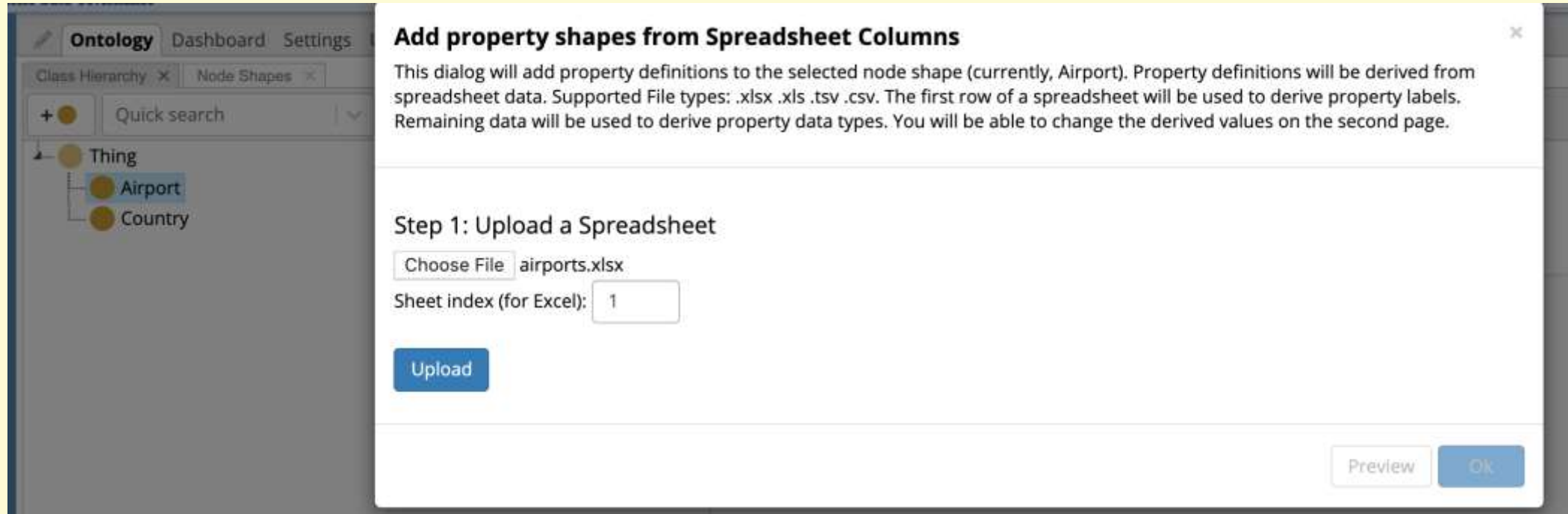
- Clone...
- Merge...
- Replace...
- Add Label Property Declaration
- Clone property shapes from...
- Derive property shapes from instances...** (highlighted with a red border)
- Add property shapes from Spreadsheet columns...
- Add property shapes from Wikidata sample...
- Delete...

Two callout boxes provide additional context:

- A yellow callout box points to the "Derive property shapes from instances..." option, containing the text: "Deriving properties from different data sources".
- A yellow callout box points to the "Property Groups for Airport" pane, containing the text: "SHACL provides grouping of properties".

1

1. Select a spreadsheet



Add property shapes from Spreadsheet Columns

This dialog will add property definitions to the selected node shape (currently, Airport). Property definitions will be derived from spreadsheet data. Supported File types: .xlsx .xls .tsv .csv. The first row of a spreadsheet will be used to derive property labels. Remaining data will be used to derive property data types. You will be able to change the derived values on the second page.

Step 1: Upload a Spreadsheet

Choose File airports.xlsx

Sheet index (for Excel): 1

Upload

Preview OK

2. Select, modify, map as needed

3. Airport's properties are added

2

1. Select a spreadsheet
2. Select, modify, map as needed

Add property shapes from Spreadsheet Columns

This dialog will add property definitions to the selected node shape (currently, Airport). Property definitions will be derived from spreadsheet data. Supported File types: .xlsx .xls .tsv .csv. The first row of a spreadsheet will be used to derive property labels. Remaining data will be used to derive property data types. You will be able to change the derived values on the second page.

Step 2: Convert Columns to Property Definitions

Select "Label" from the Property Type dropdown to identify the property to be used as a label (name) for resources. This is not needed if your class already defines a label property, e.g. through inheritance.

Select "Relationship" if a property is a relationship. You can then select the target class.

Property Label	Property ID	Property Type
<input checked="" type="checkbox"/> Airport Name	example:AirportName	String
<input checked="" type="checkbox"/> City	example:City	String
<input checked="" type="checkbox"/> Country	example:Country	String
<input checked="" type="checkbox"/> Country Code	example:CountryCode	String
<input checked="" type="checkbox"/> IATA Code	example:IATACode	String
<input checked="" type="checkbox"/> Latitude	example:Latitude	Double
<input checked="" type="checkbox"/> Longitude	example:Longitude	Double

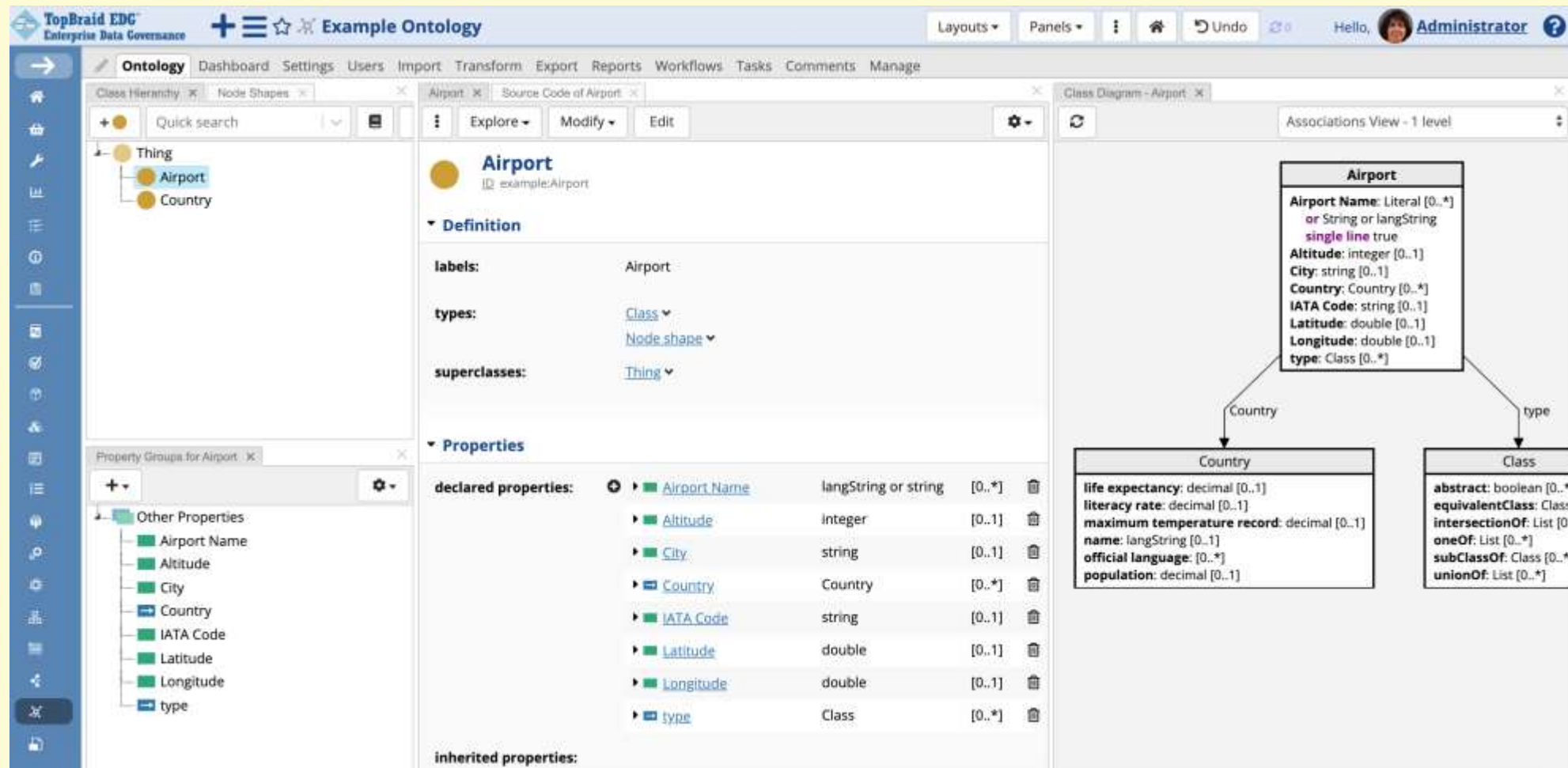
- Boolean
- Date
- Date Time
- Double
- Integer
- Label
- Relationship
- String
- Time
- URI (literal)

Preview

Ok

3

1. Select a spreadsheet
2. Select, modify, map as needed
3. Airport's properties are added



The screenshot displays the TopBraid EDG interface for an 'Example Ontology'. The main view shows the 'Airport' class definition with the following details:

- Definition:**
 - labels: Airport
 - types: Class, Node shape
 - superclasses: Thing
- Properties:**
 - declared properties:**

Property Name	Domain	Cardinality
Airport Name	langString or string	[0..*]
Altitude	integer	[0..1]
City	string	[0..1]
Country	Country	[0..*]
IATA Code	string	[0..1]
Latitude	double	[0..1]
Longitude	double	[0..1]
type	Class	[0..*]
 - inherited properties:** (None listed)

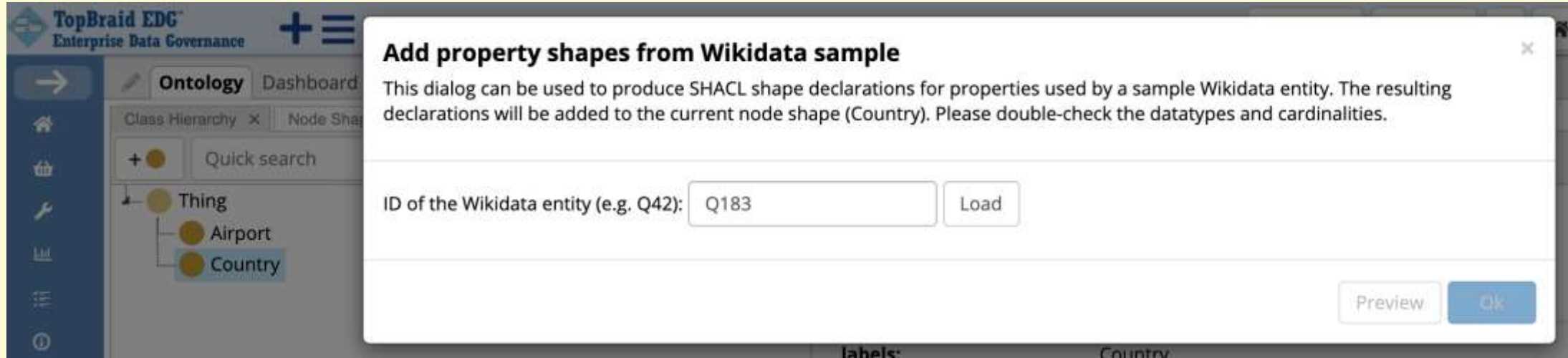
The right-hand pane shows a 'Class Diagram - Airport' with the following structure:

- Airport Class:**
 - Airport Name: Literal [0..*] or String or langString, single line true
 - Altitude: integer [0..1]
 - City: string [0..1]
 - Country: Country [0..*]
 - IATA Code: string [0..1]
 - Latitude: double [0..1]
 - Longitude: double [0..1]
 - type: Class [0..*]
- Country Class:**
 - life expectancy: decimal [0..1]
 - literacy rate: decimal [0..1]
 - maximum temperature record: decimal [0..1]
 - name: langString [0..1]
 - official language: [0..*]
 - population: decimal [0..1]
- Class Class:**
 - abstract: boolean [0..*]
 - equivalentClass: Class
 - intersectionOf: List [0..*]
 - oneOf: List [0..*]
 - subClassOf: Class [0..*]
 - unionOf: List [0..*]

The diagram shows 'Airport' as a subclass of 'Country' and 'Class'.

TopBraid EDG Derives Schema from External Knowledge Graphs - 1

1. Map a type (e.g., Country class) to an external Knowledge Graph (e.g., Wikidata)



2. Select properties to be derived from the external graph (e.g., population)
3. Country's properties are added

TopBraid EDG Derives Schema from External Knowledge Graphs - 2

2. 1. Map a type (e.g., Country class) to an external Knowledge Graph (e.g., Wikidata)
2. 2. Select properties to be derived from the external graph (e.g., image, population, etc)

Add property shapes from Wikidata sample

This dialog can be used to produce SHACL shape declarations for properties used by a sample Wikidata entity. The resulting declarations will be added to the current node shape (Country). Please double-check the datatypes and cardinalities.

- ▶ life expectancy (wdt:P2250) 0..1 decimal
- ▶ literacy rate (wdt:P6897) 0..1 decimal
- ▶ LoC and MARC vocabularies ID (wdt:P4801) 0..1 string
- ▶ located in or next to body of water (wdt:P206) 0..*
- ▶ located in time zone (wdt:P421) 0..*
- ▶ located on terrain feature (wdt:P706) 0..*
- ▶ location map (wdt:P1943) 0..*
- ▶ locator map image (wdt:P242) 0..*
- ▶ lowest point (wdt:P1589) 0..*
- ▶ mains voltage (wdt:P2884) 0..1 decimal
- ▶ maintained by WikiProject (wdt:P6104) 0..*
- ▶ Marine Regions Geographic ID (wdt:P3006) 0..1 string
- ▶ maritime identification digits (wdt:P2979) 0..1 string
- ▶ marriageable age (wdt:P3000) 0..1 decimal
- ▶ maximum temperature record (wdt:P6591) 0..1 decimal
- ▶ median income (wdt:P3529) 0..1 decimal

Preview

Ok

Germany (Q183)

sovereign state in central-western Europe
FRG | BRD | Bundesrepublik Deutschland | Federal Republic of Germany | de | Deutschland | GER

In more languages

Configure

Language	Label	Description	Also known as
English	Germany	sovereign state in central-western Europe	FRG BRD Bundesrepublik Deutschland Federal Republic of Germany de Deutschland GER

wikidata.org/wiki/Q183

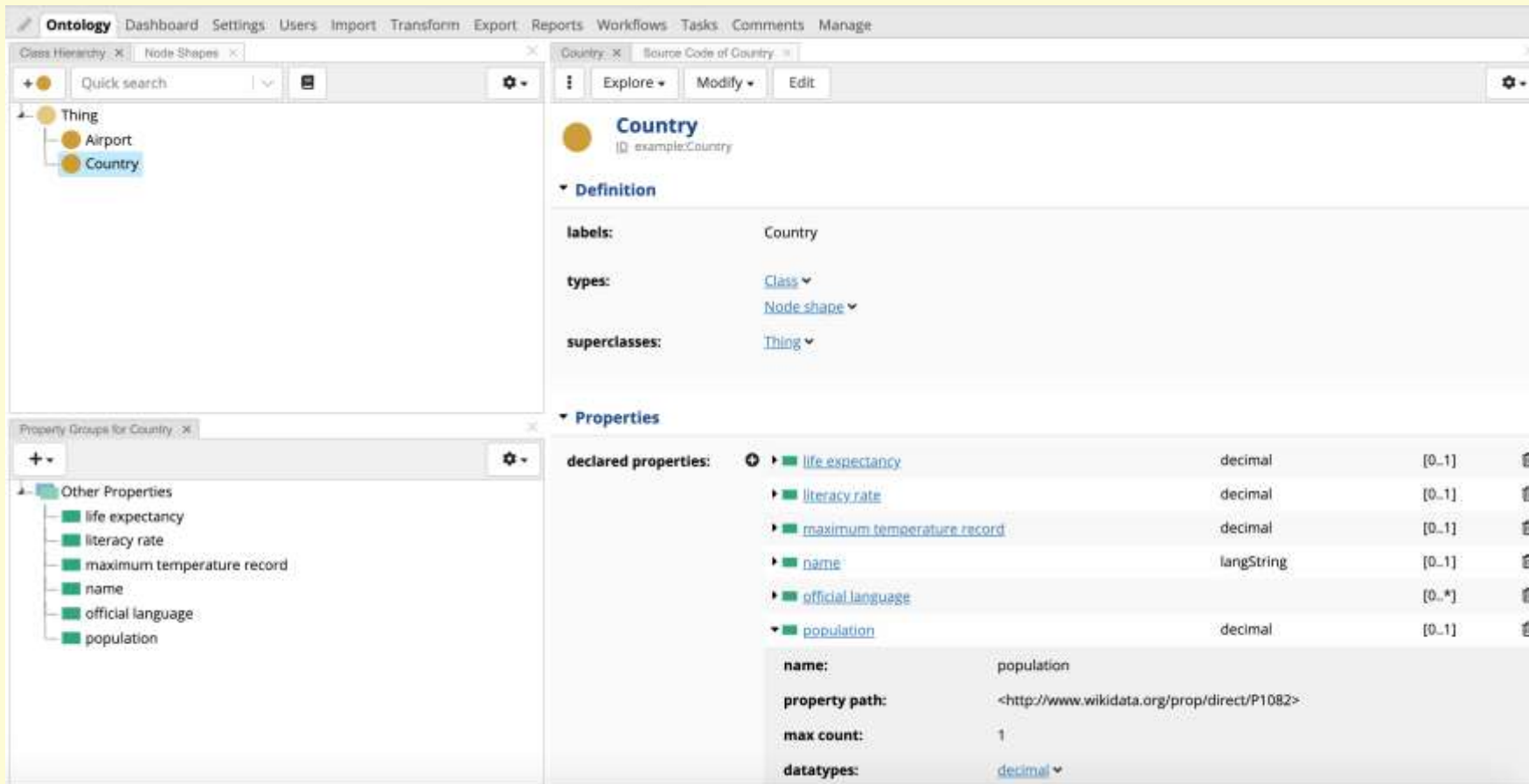
- instance of
 - + 1 reference
 - administrative territorial entity
 - 0 references
 - countries bordering the Baltic Sea
 - 0 references
- drainage basin
 - Danube basin
 - 0 references
- life expectancy
 - 50.8 year
 - point in time
 - applies to part
 - 2017
 - both genders
 - + 1 reference



TopBraid EDG Derive Schema from External Knowledge Graphs - 3

3

1. Map a type (e.g., Country class) to an external Knowledge Graph (e.g., Wikidata)
2. Select properties to be derived from the external graph (e.g., life expectancy, population, etc)
3. Country's properties are added

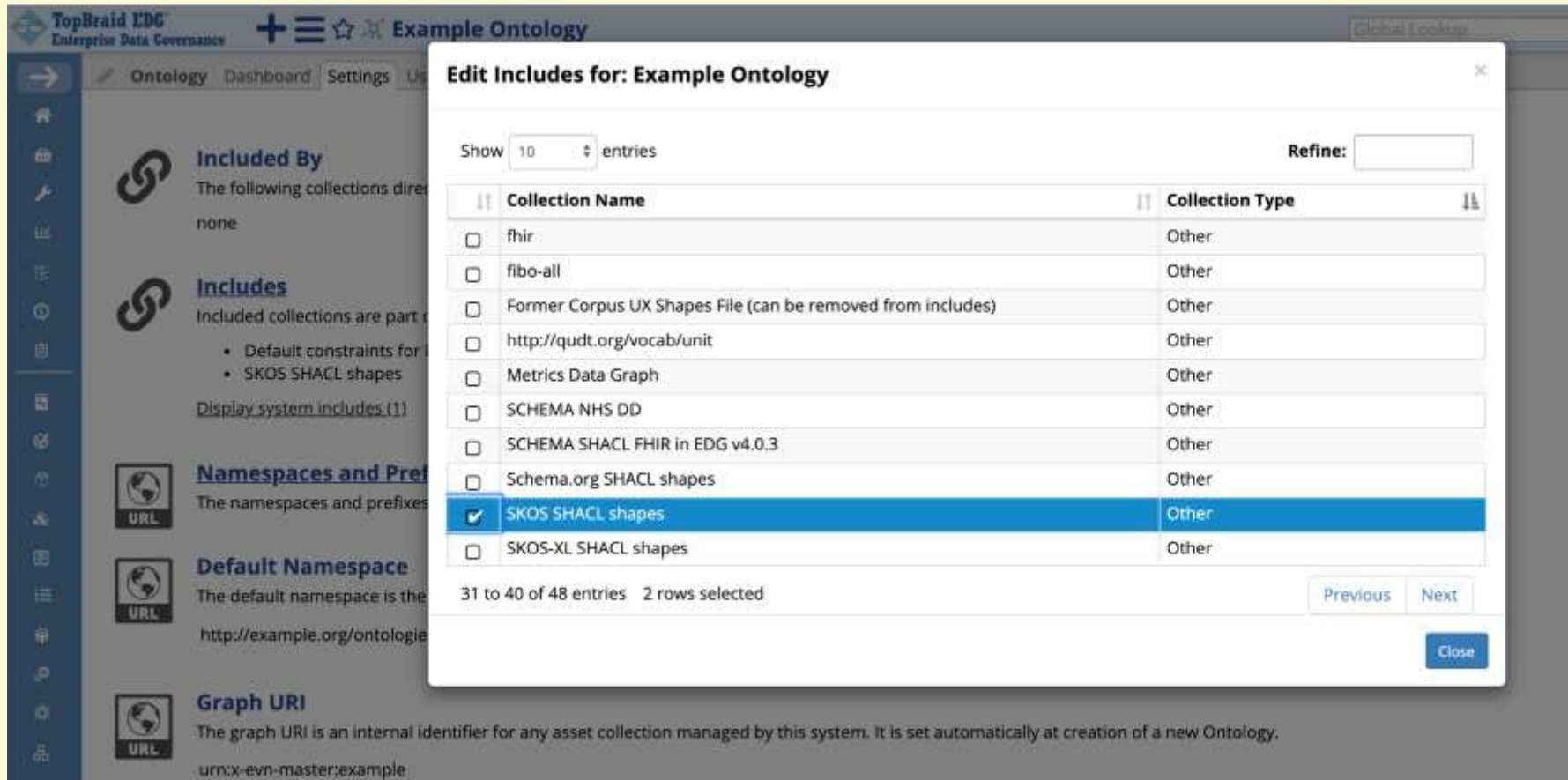


The screenshot displays the TopBraid EDG interface for configuring an ontology. The main window shows the 'Country' class configuration, which is derived from an external knowledge graph (Wikidata). The configuration includes the following details:

- Class Hierarchy:** Thing (parent), Airport (child), Country (child).
- Property Groups for Country:** Other Properties (parent), life expectancy (child), literacy rate (child), maximum temperature record (child), name (child), official language (child), population (child).
- Declared Properties:**
 - life expectancy: decimal, [0..1]
 - literacy rate: decimal, [0..1]
 - maximum temperature record: decimal, [0..1]
 - name: langString, [0..1]
 - official language: [0..*]
 - population: decimal, [0..1]
- Property Details for 'population':**
 - name:** population
 - property path:** <http://www.wikidata.org/prop/direct/P1082>
 - max count:** 1
 - datatypes:** decimal

Model Re-Use - 1

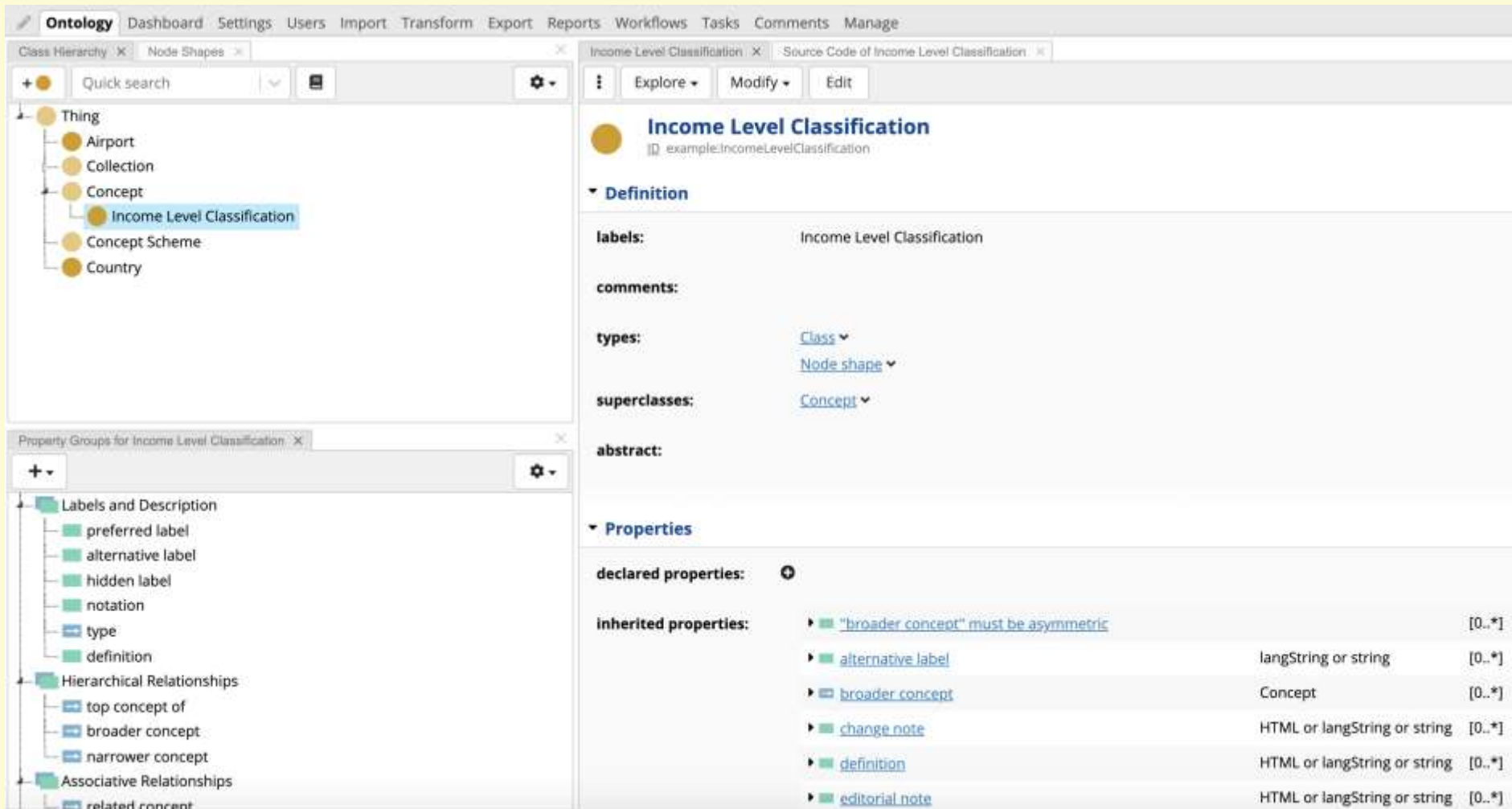
1. Include an external ontology



2. Integrate it e.g., add your own subclasses, use classes in the ontology as targets of relationships, etc.
3. For classes that you will use, deactivate properties you will not use

2

1. Include an external ontology
2. Integrate it e.g., add your own subclasses, use classes in the ontology as targets of relationships, etc.



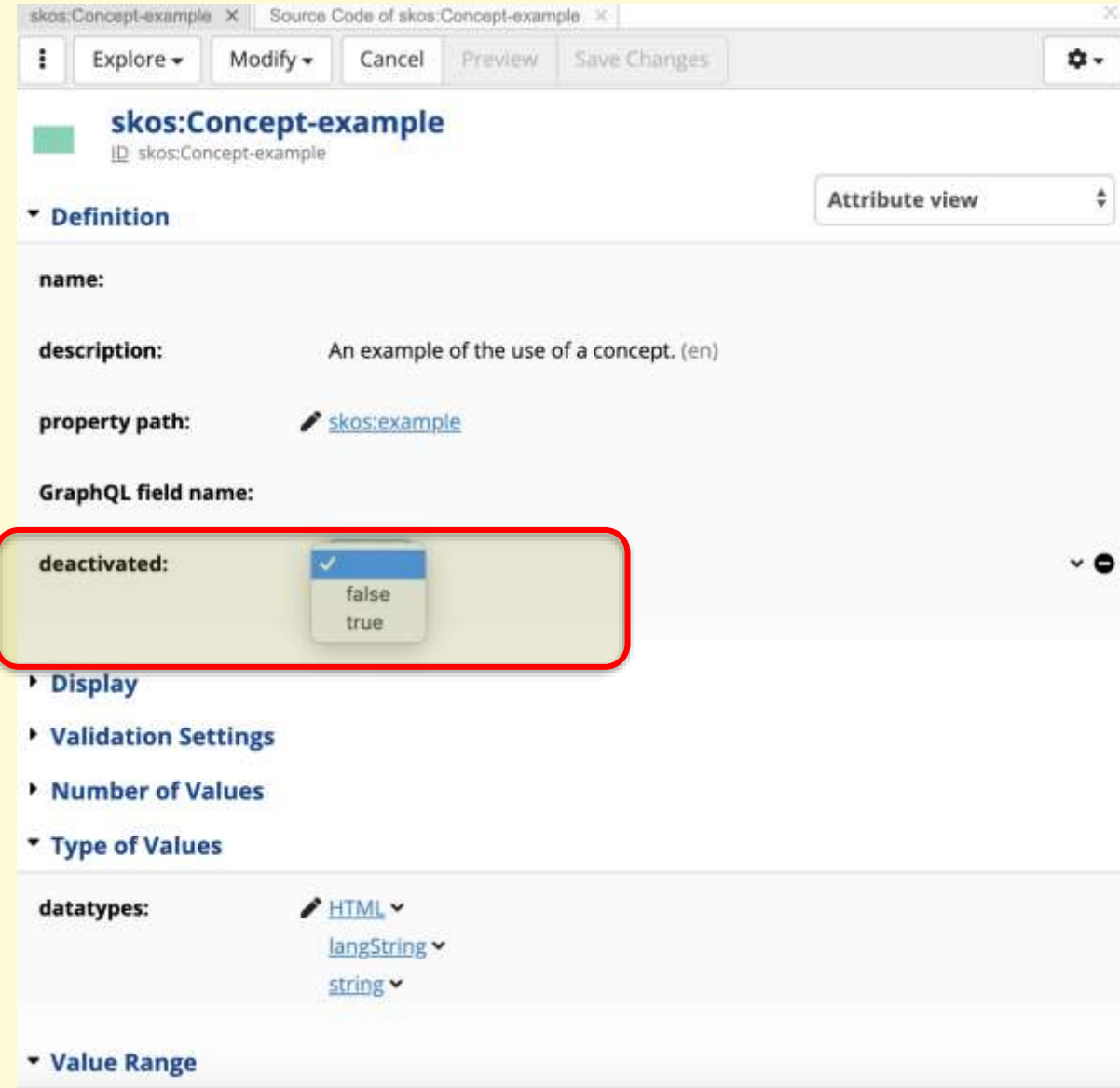
The screenshot displays an ontology editor interface. On the left, a 'Class Hierarchy' pane shows a tree structure starting with 'Thing' as the root. Under 'Thing', there are several classes: 'Airport', 'Collection', 'Concept', 'Income Level Classification' (highlighted in blue), 'Concept Scheme', and 'Country'. Below the hierarchy is a 'Property Groups for Income Level Classification' pane, which is expanded to show 'Labels and Description' properties: 'preferred label', 'alternative label', 'hidden label', 'notation', 'type', and 'definition'. Other groups include 'Hierarchical Relationships' (top concept of, broader concept, narrower concept) and 'Associative Relationships' (related concept).

The main right-hand pane shows the detailed view for the 'Income Level Classification' class. It includes a title bar with 'Explore', 'Modify', and 'Edit' buttons. The class name is 'Income Level Classification' with the URI 'example:IncomeLevelClassification'. The 'Definition' section shows 'labels: Income Level Classification'. The 'types' section has dropdown menus for 'Class', 'Node shape', and 'Concept'. The 'superclasses' section shows 'Concept'. The 'Properties' section is expanded to show 'declared properties' (none) and 'inherited properties':

Property Name	Domain	Cardinality
"broader concept" must be asymmetric		[0..*]
alternative label	langString or string	[0..*]
broader concept	Concept	[0..*]
change note	HTML or langString or string	[0..*]
definition	HTML or langString or string	[0..*]
editorial note	HTML or langString or string	[0..*]

Model Re-Use - 3

1. Include an external ontology
2. Integrate it e.g., add your own subclasses, use classes in the ontology as targets of relationships, etc.
3. For classes that you will use, deactivate properties you will not use



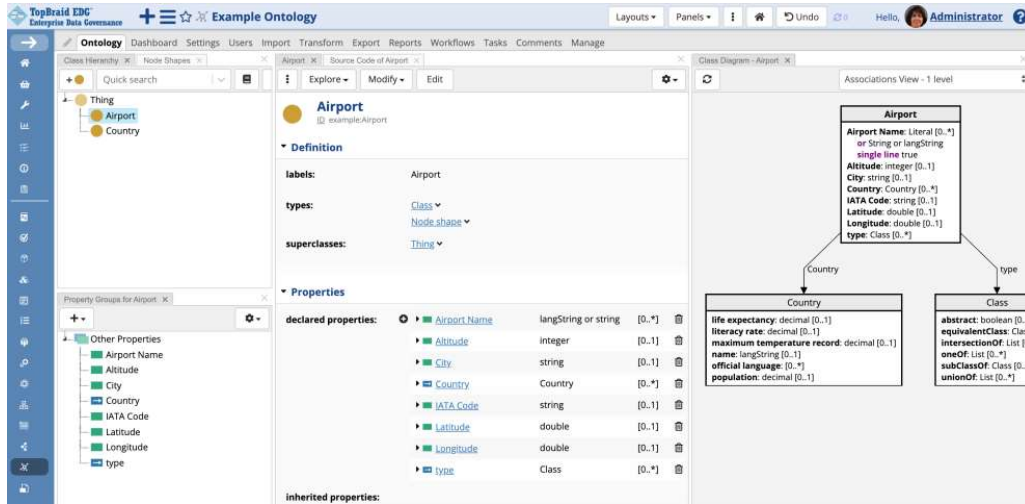
The screenshot shows the configuration page for the class `skos:Concept-example`. The interface includes a toolbar with 'Explore', 'Modify', 'Cancel', 'Preview', and 'Save Changes' buttons. The class name and ID are displayed at the top. The 'Definition' section is expanded, showing fields for 'name', 'description' (An example of the use of a concept. (en)), 'property path' (`skos:example`), and 'GraphQL field name'. The 'deactivated' field is highlighted with a red box, and its dropdown menu is open, showing 'false' selected and 'true' as an option. Below the 'Definition' section, there are sections for 'Display', 'Validation Settings', 'Number of Values', 'Type of Values' (with datatypes: HTML, langString, string), and 'Value Range'.

Challenge 3: Ensuring that Developers Like to Work with RDF



Use of introspection to query the semantic models

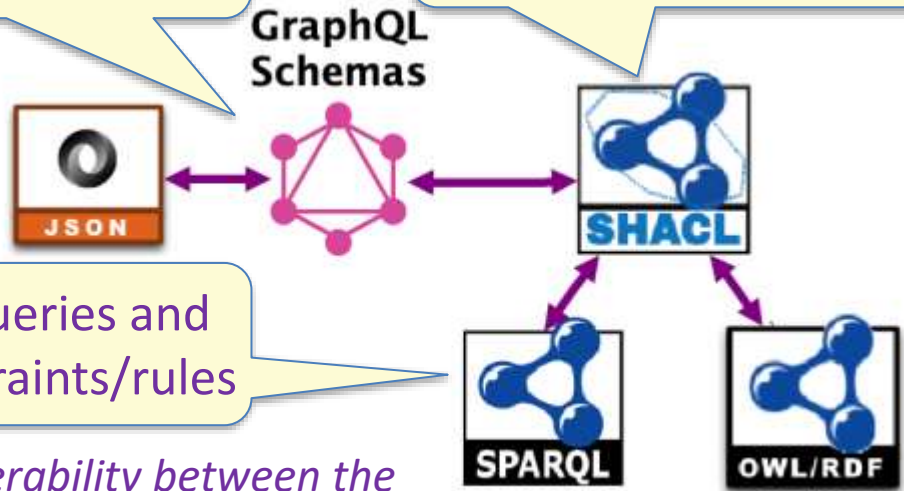
Translates GraphQL to and from SHACL and RDF



Bridging the Knowledge Graph and JSON worlds

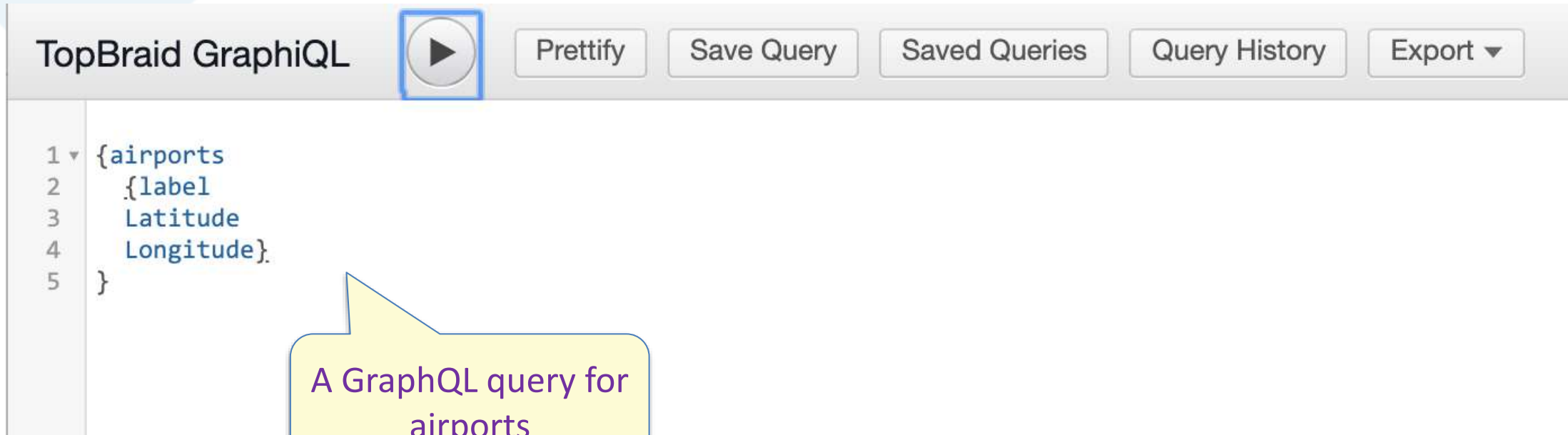
SHACL Schema and Data Shapes Models

SPARQL for queries and complex constraints/rules



GraphQL provides an interface technology for interoperability between the Javascript world of User Interfaces and SHACL-based Data Models and Rules

Example of Querying Knowledge Graph Data with GraphQL



The screenshot shows the TopBraid GraphiQL interface. The top bar contains the text "TopBraid GraphiQL" and a play button icon (highlighted with a blue box), followed by buttons for "Prettify", "Save Query", "Saved Queries", "Query History", and "Export". Below the top bar, a text editor displays a GraphQL query:

```
1 {airports
2   {label
3     Latitude
4     Longitude}
5 }
```

A yellow callout box with a purple border points to the query and contains the text: "A GraphQL query for airports".

For more see video at: https://www.topquadrant.com/project/graphql_json_rdf/

Example of Querying Knowledge Graph Data



```
1 {airports
2   {label
3     Latitude
4     Longitude}
5 }
```

```
{
  "data": {
    "airports": [
      {
        "label": "Yenbo",
        "Latitude": 24.144244,
        "Longitude": 38.06335
      },
      {
        "label": "Namorik Atoll Airport",
        "Latitude": 5.63167,
        "Longitude": 168.125
      },
      {
        "label": "Bantry Aerodrome",
        "Latitude": 51.668598,
        "Longitude": -9.48417
      },
      {
        "label": "Labe",
        "Latitude": 11.326058,
        "Longitude": -12.28685
      },
      {
        "label": "Chennai Intl",
        "Latitude": 12.994414,
        "Longitude": 80.180517
      },
      {
        "label": "John F Kennedy Intl",
        "Latitude": 40.639751,
        "Longitude": -73.778925
      },
      {
        "label": "Guemar Airport",
        "Latitude": 33.5114,
        "Longitude": 6.77679
      },
      {
        "label": "Chimbu Airport",
        "Latitude": -6.02429,
        "Longitude": 144.971
      }
    ]
  }
}
```

JSON Results

For more see video at: https://www.topquadrant.com/project/graphql_json_rdf/

Example of Querying Knowledge Graph Data with GraphQL

TopBraid GraphQL
▶
Prettify
Save Query
Saved Queries
Query History
Export ▼

```

1 {airports {
2   label
3   latitude
4   longitude
5 }}

```

```

{
  "data": {
    "airports": [
      {
        "label": "Yenbo",
        "latitude": 24.144244,
        "longitude": 38.86335
      },
      {
        "label": "Namorik Atoll Airport",
        "latitude": 5.63167,
        "longitude": 168.125
      },
      {
        "label": "Bantry Aerodrome",
        "latitude": 51.668598,
        "longitude": -9.48417
      },
      {
        "label": "Labe",
        "latitude": 11.326058,
        "longitude": -12.28685
      },
      {
        "label": "Chennai Intl",
        "latitude": 12.994414,
        "longitude": 80.180517
      },
      {
        "label": "John F Kennedy Intl",
        "latitude": 40.639751,
        "longitude": -73.778925
      },
      {
        "label": "Guemar Airport",
        "latitude": 33.5114,
        "longitude": 6.77679
      },
      {
        "label": "Chimbu Airport",
        "latitude": -6.02429,
        "longitude": 144.971
      },
      {
        "label": "Mosul International Airport",
        "latitude": 36.305833,
        "longitude": 43.1475
      }
    ]
  }
}

```

< Schema
RootRDFQuery
✕

No Description

FIELDS

airports(
 queryText: String
 filter: String
 uri: ID
 orderBy: Airport_FieldsEnum
 orderByExpr: String
 orderByDesc: Boolean
 orderAll: Boolean
 altitude: Int
 city: String
 country: ID
 iataCode: String
 latitude: Float
 longitude: Float
 rdf_type: ID
 where: Airport_where
 first: int
 skip: int
): [Airport]

graphReport(excludeIds: [String], resourceType: String): _GraphReport

The results report of the data graph, validating all resources.

_rootTypeShapes: [_TypeShape]

Requests the shape information for all root types.

_typeShapeByName(name: String!): _TypeShape

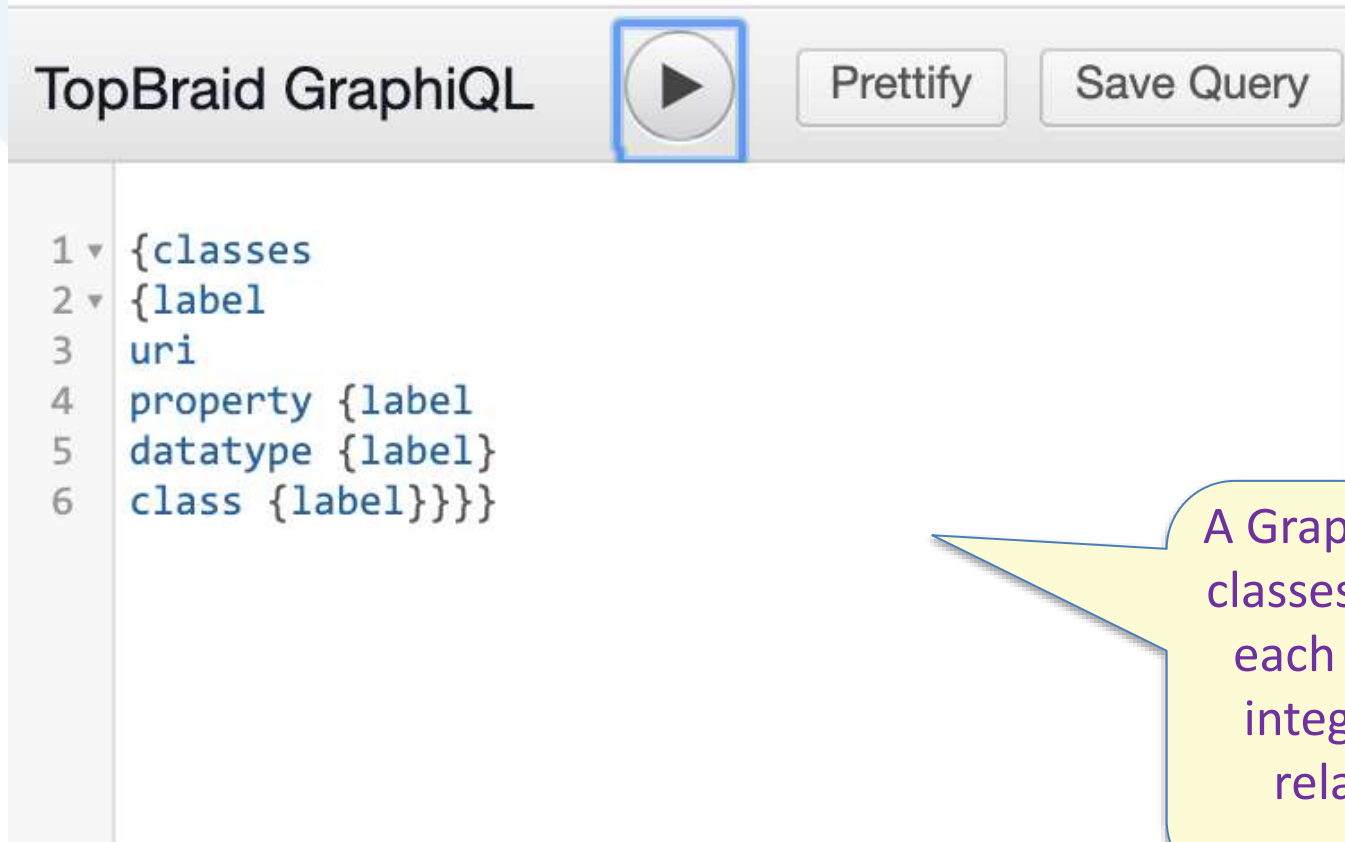
Requests the shape information of a type with a given name.

_typeShapeByURI(uri: String!): _TypeShape

QUERY VARIABLES

The GraphQL schema automatically transformed from a SHACL model

Example of Querying Model with GraphQL



The screenshot shows the TopBraid GraphiQL interface. The title bar reads "TopBraid GraphiQL". On the right side of the title bar, there are three buttons: a play button (highlighted with a blue box), "Prettify", and "Save Query". The main area contains a GraphQL query with line numbers 1 through 6 on the left:

```
1 {classes
2 {label
3 uri
4 property {label
5 datatype {label}
6 class {label}}}}
```

A GraphQL query over a SHACL model – all classes with their properties, returning for each property a datatype (e.g., string or integer) and class (this is applicable for relationships such as a link between airports and countries)

Example of Querying SHACL/RDF with GraphQL

TopBraid GraphiQL ▶ Prettify Save Query Saved Queries Query History Export ▾

```

1 ▾ {classes
2 ▾ {label
3   uri
4   property {label
5   datatype {label}}
6   class {label}}}}

```

```

{
  "label": "Airport",
  "uri": "http://example.org/ontologies/Example#Airport",
  "property": [
    {
      "label": "example:Airport-Longitude",
      "datatype": {
        "label": "double"
      },
      "class": []
    },
    {
      "label": "example:Airport-label",
      "datatype": null,
      "class": []
    },
    {
      "label": "example:Airport-type",
      "datatype": null,
      "class": [
        {
          "label": "Class"
        }
      ]
    }
  ]
},
{
  "label": "example:Airport-City",
  "datatype": {
    "label": "string"
  },
  "class": []
},
{
  "label": "example:Airport-Latitude",
  "datatype": {
    "label": "double"
  },
  "class": []
}

```

JSON Results

Example of Querying SHACL/RDF with GraphQL

TopBraid GraphiQL

RootRDFQuery

```

1 * {classes
2 * {label
3 * uri
4 * property {label
5 * datatype {label}
6 * class {label}}}}
        
```

```

{
  "label": "Airport",
  "uri": "http://example.org/ontologies/Example#Airport",
  "property": [
    {
      "label": "example:Airport-Longitude",
      "datatype": {
        "label": "double"
      },
      "class": []
    },
    {
      "label": "example:Airport-label",
      "datatype": null,
      "class": []
    },
    {
      "label": "example:Airport-type",
      "datatype": null,
      "class": [
        {
          "label": "Class"
        }
      ]
    }
  ],
  {
    "label": "example:Airport-City",
    "datatype": {
      "label": "string"
    },
    "class": []
  },
  {
    "label": "example:Airport-l",
    "datatype": {
      "label": "double"
    },
    "class": []
  },
  {
    "label": "example:Airport-Country",
    "datatype": null,
    "class": [
      {
        "label": "Country"
      }
    ]
  }
}
        
```

```

classes{
  queryText: String
  filter: String
  uri: ID
  orderBy: Class_FieldsEnum
  orderByExpr: String
  orderByDesc: Boolean
  orderAll: Boolean
  allSuperShapes: ID
  closed: Boolean
  datatypes: ID
  hasSubClass: Boolean
  hasSubShape: Boolean
  hasVisibleSubClass: Boolean
  hidden: Boolean
  isAbstract: Boolean
  isClass: Boolean
  isHidden: Boolean
  isHiddenClass: Boolean
  isImported: Boolean
  newURIBase: String
  node: ID
  pathToRootClass: String
  pathToRootShape: String
  privateShapeOf: ID
  property: ID
  detectedShapeOf: ID
  publicClassOf: ID
  publicShapeOf: ID
  classCount: Int
  classOf: ID
  shapeCount: Int
  superClassOf: ID
  itemLabel: String
  getClass: ID
  targetObjectsOf: ID
  targetSubjectsOf: ID
  inheritedProperty: ID
  applicableToClass: ID
  rdf_type: ID
  defaultViewForRole: ID
  abstract: Boolean
  deactivated: Boolean
}
        
```

The GraphQL schema for models automatically transformed from SHACL

Challenge 4: Capturing Statements About Statements

Some facts have additional information that you may want to capture

population	80,500,000±500
point in time	31 December 2012
↳ 1 reference	
imported from Wikimedia project	DESTATIS
reference URL	https://www.destatis.de/DE/ZahlenFakten/GesellschaftStaat/Bevoelkerung/Bevoelkerung.html
retrieved	14 June 2014
81,752,000±500	2010
↳ 1 reference	
82,002,000±500	2008
↳ 1 reference	
82,315,000±500	2006
↳ 1 reference	

This is typically called “reification” - of a property value

Two approaches

1. Turn an attribute into a relationship
Use Population as a class with instances
For example:
ex:Germany ex:population ex:P123.
ex:P123 a ex:Population;
ex:value 80500000;
ex:date 2012.
2. Keep an attribute and its value as-is
Add a statement with the date “on top” of the population value

Enabling of a “less verbose” Way to Capture Statements About Statements

- Create a node shape that will define facts a user may need to add to values

- Identify a property which values should be annotated with additional facts and connect it to the shape



Provenance shape
schemaont:ProvenanceShape

▼ Definition

labels: Provenance shape

types: [Node shape](#) ▼

▼ Properties

declared properties:

▶ effective_date	date	[0..1]	🗑️
▶ source	string	[0..*]	🗑️



Explore ▼ Modify ▼ Edit

example:Country-P1082
ID example:Country-P1082

Attribute view

▼ Definition

name: population

property path: [wikidata:P1082](#)

▼ Type of Values

datatypes: [decimal](#) ▼

▼ Reification (Statements about Statements)

reifiable by: [Provenance shape](#) ▼

Example of Reification: Adding Annotations

In this example, users working with data can view and add a source and an effective date to statements capturing country's population

Reification Button opens the sub-form of reification properties

population: 80500000

effective date: 2018-12-31

source:

81752000

life expectancy:

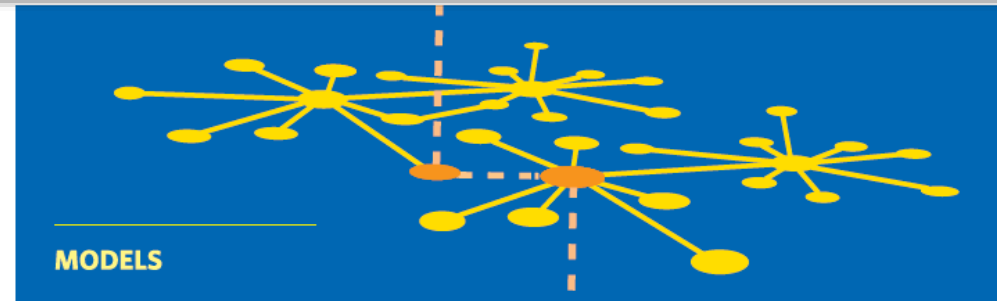
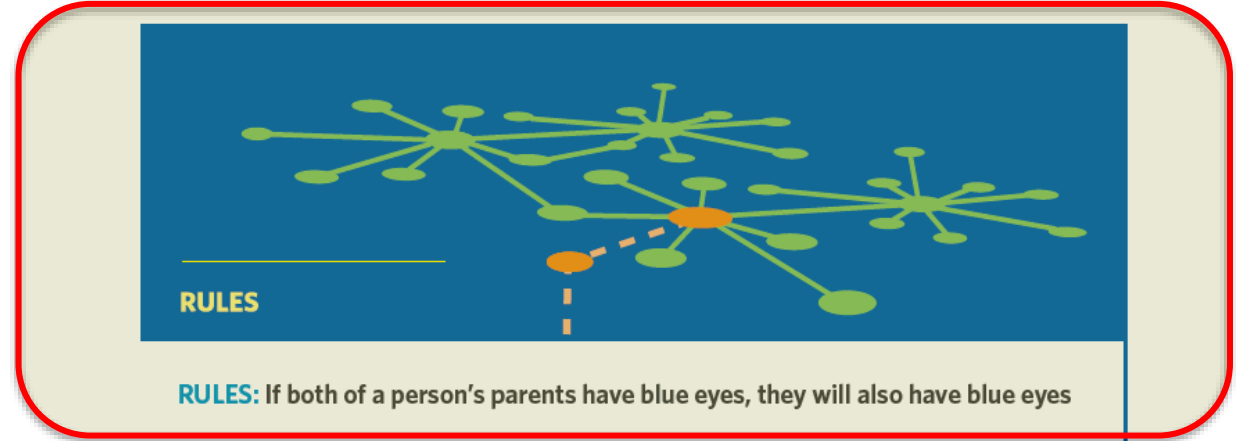
Stored as a single extra triple fact

```

Germany x | Source Code of Germany x
Cancel Save Changes
@prefix a ↔ .
airports:Germany
  a example:Country ;
  rdfs:label "Germany" ;
  wikidata:P1082 "80500000"^^xsd:decimal [[ example:effectiveDate "2018-12-31"^^xsd:date ] ] ;
  wikidata:P1082 "81752000"^^xsd:decimal ;
.
  
```

Challenge 5: Effective Ways to do Reasoning

- One of the advantages of the Knowledge Graphs is ability to do reasoning
 - Infer additional facts based on the facts that were provided



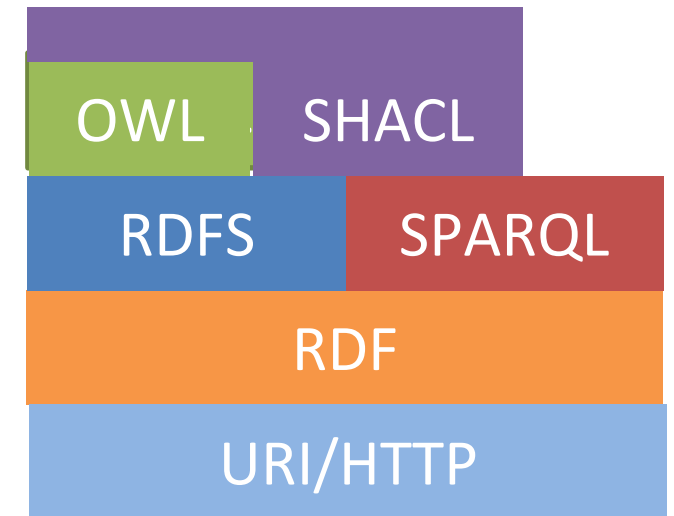
MODELS : A person has eye color. A person has two parents. A person's father is also a person and he is male.



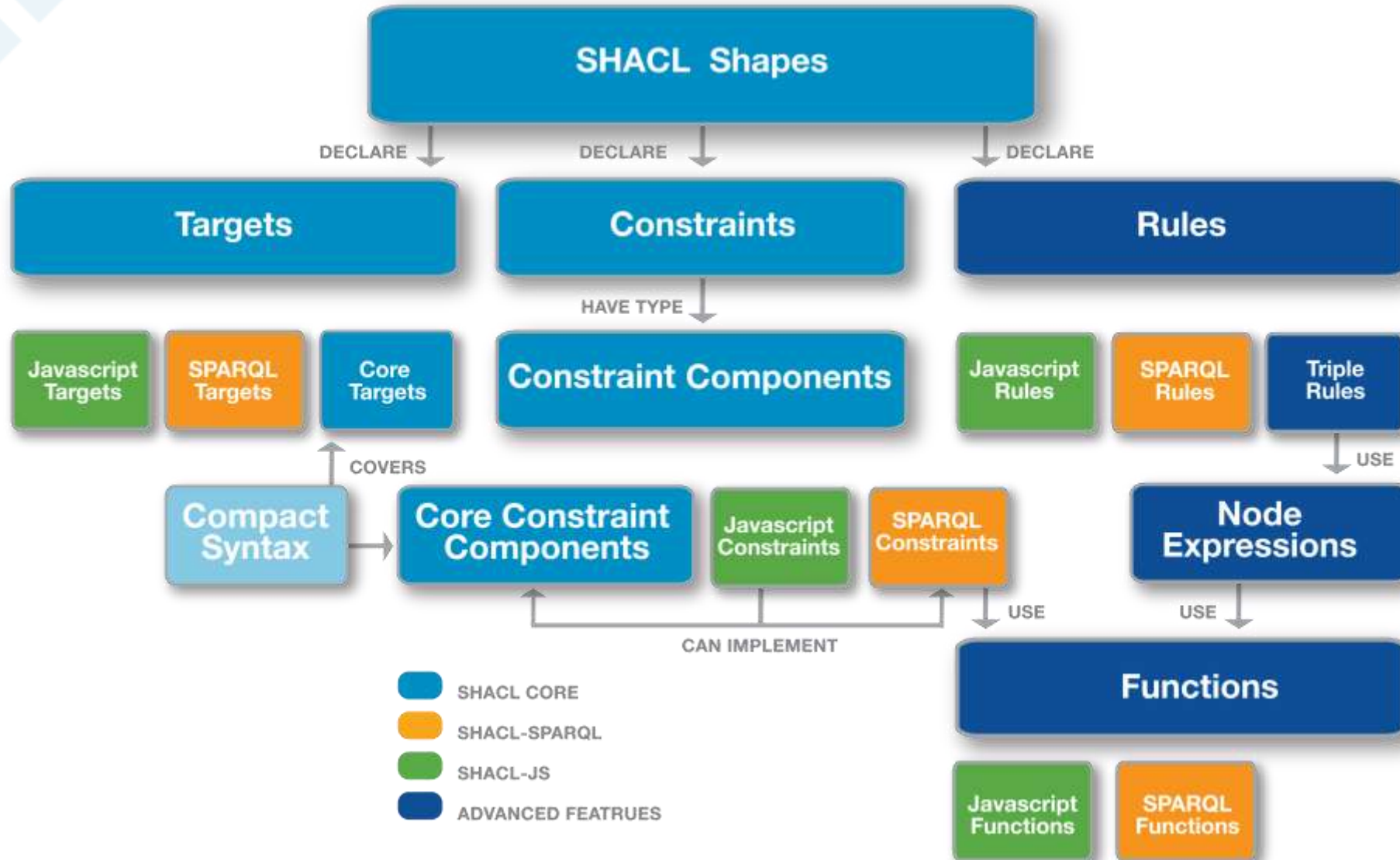
FACTS: James has blue eyes. James' father is Andrew. James is a person.

Reasoning or inferencing is a complex topic, often misunderstood

- OWL is difficult
 - for classification not for specification (open world)
 - Instance-level (A-Box) constraints and rules are hard if not impossible
- Emerging from practice, SHACL (previously SPIN) has proven success:
 - Developed from experiences of real-world problems across several application domains
 - Shapes as models of constraints
 - Highly expressive
 - Node Shapes
 - Property Shapes
 - Highly effective for data (and model) validation
 - Supporting rules and computed values



SHACL Components

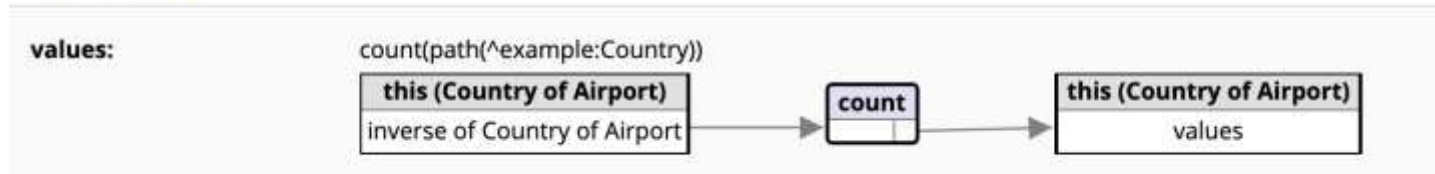


Example of a Computed Value

Several templates are available to simplify rule creation. More can be added.

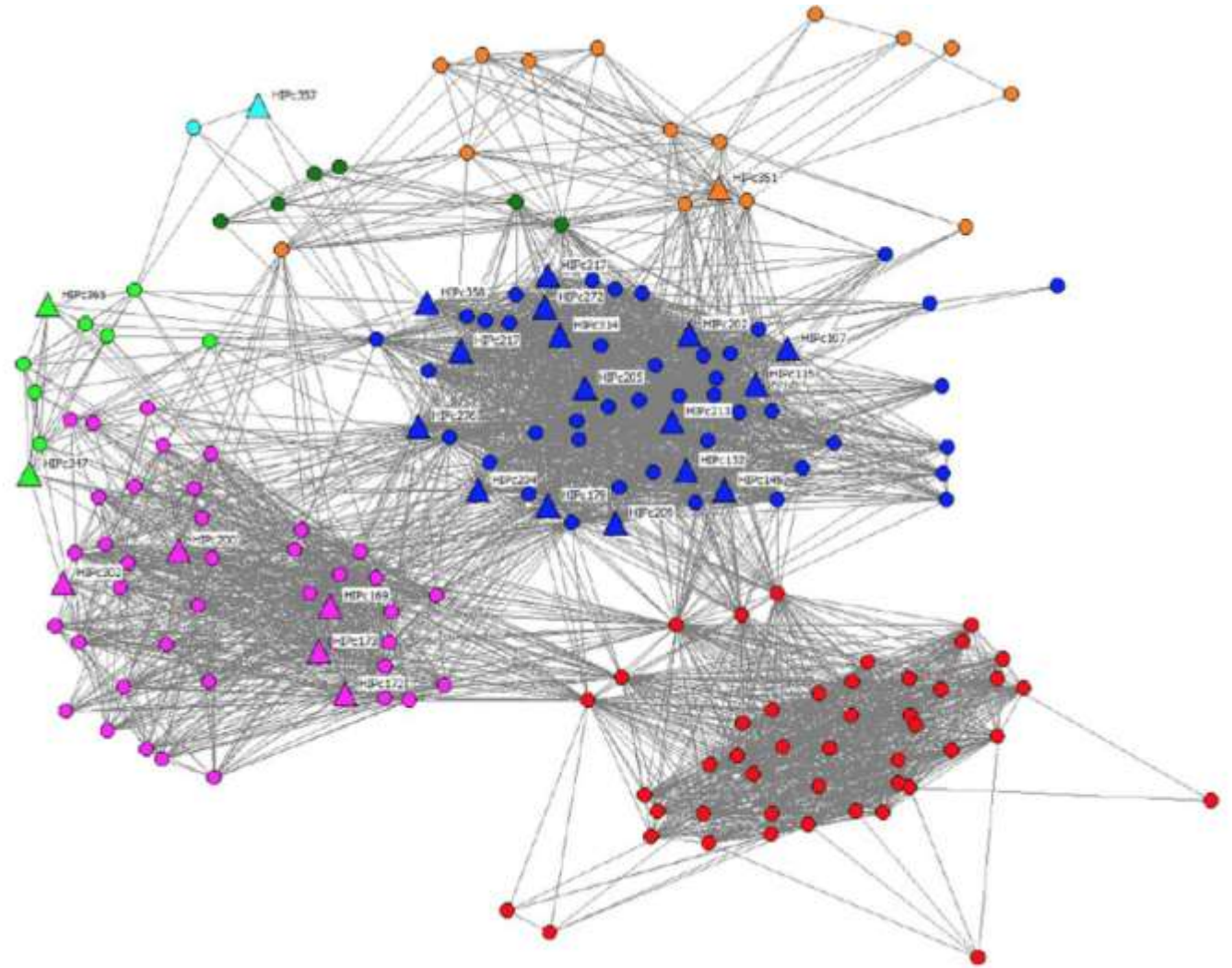
Inferences

3



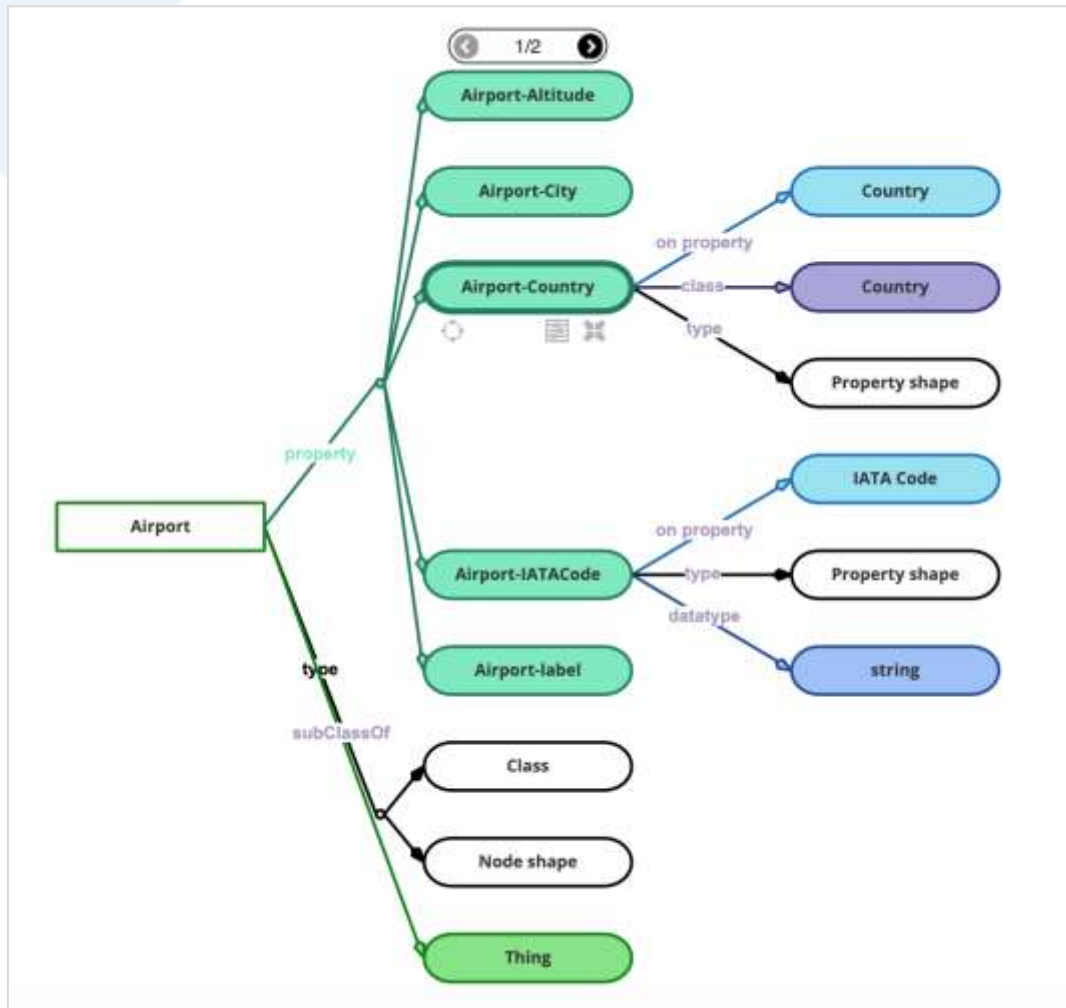
Challenge 6: Task Appropriate Visualizations

- Commonly used, basic graph visualization shows all nodes and links in a graph
- Useful to identify clusters of data or to graphically see all links from/to a given node
- Does not necessarily fit all use cases and activities

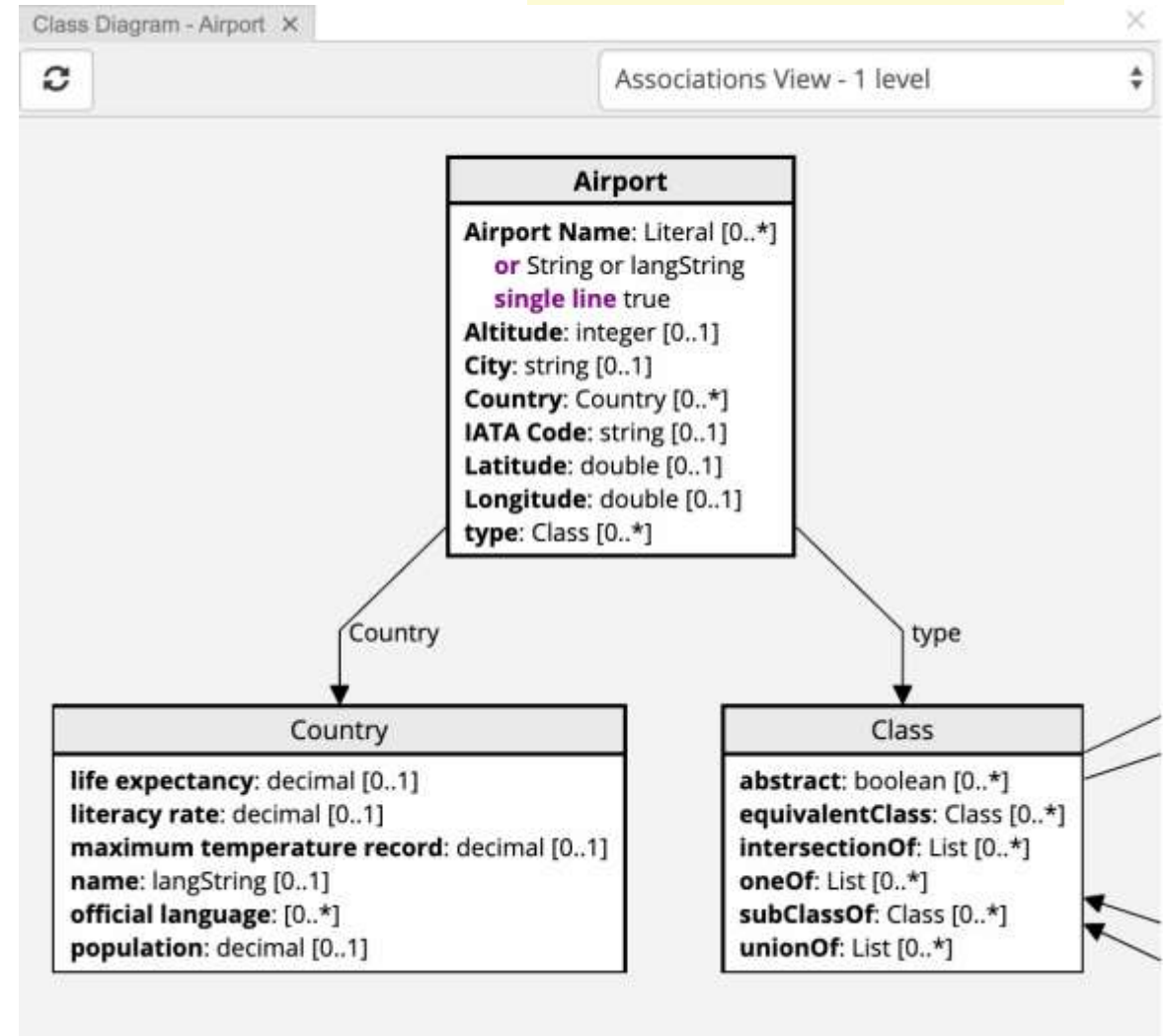


Example of Task Appropriate Visualization for an Ontology

Graph Diagram

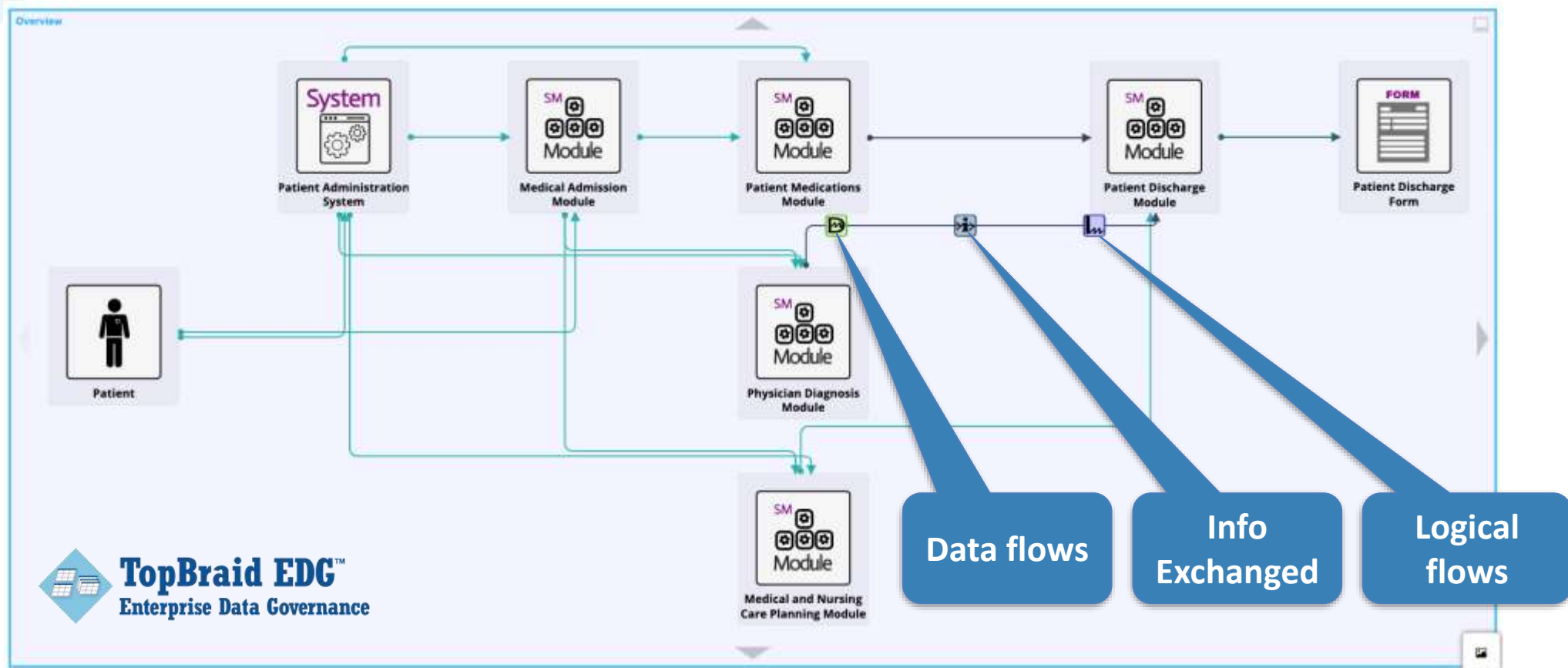


Class Diagram

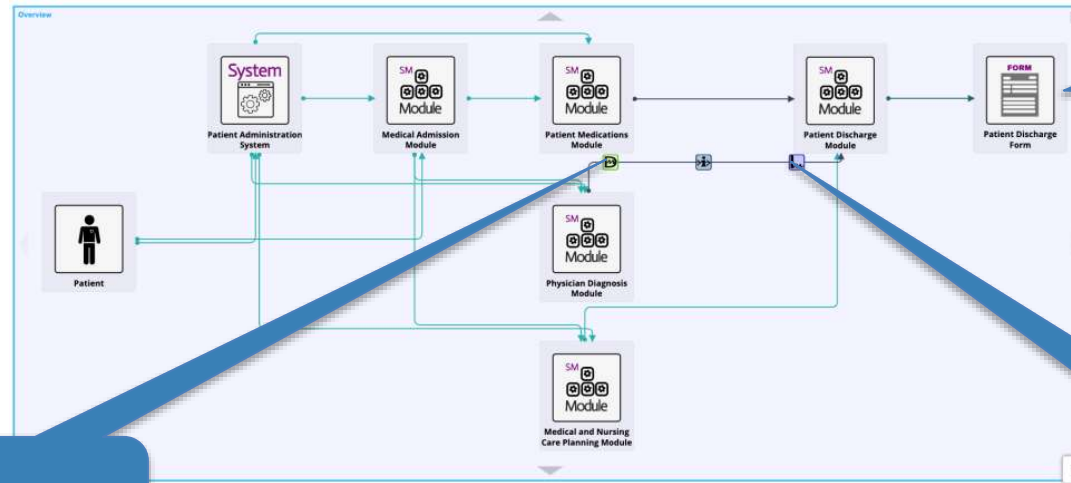


Example of Data Lineage Diagram

- A Viewpoint-specific visualization that uses reasoning. For example, to present links that are semantic inferences of dependencies.



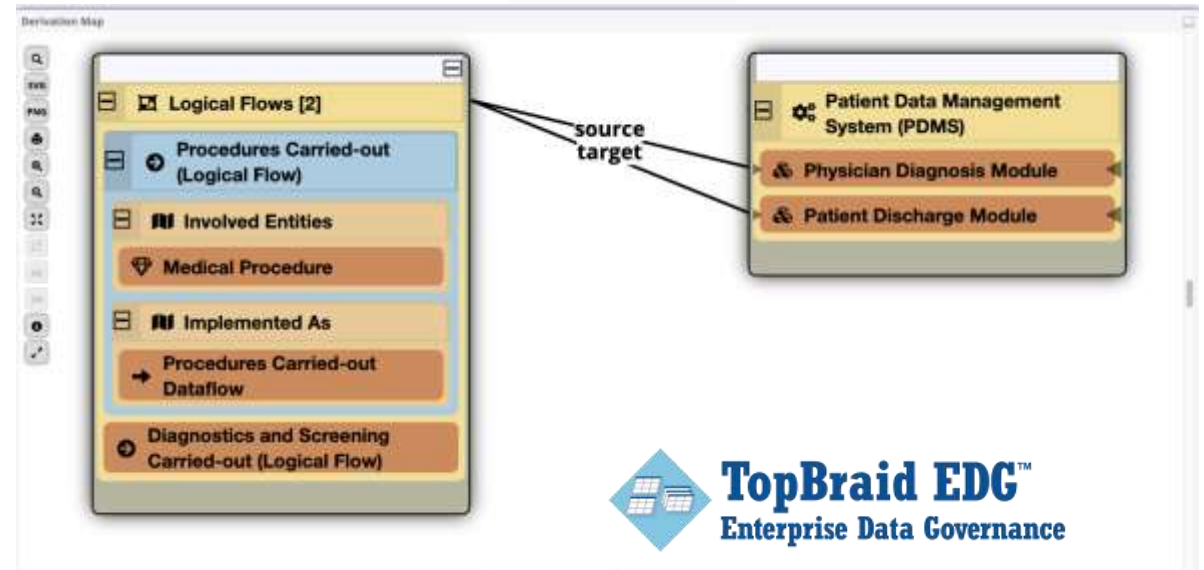
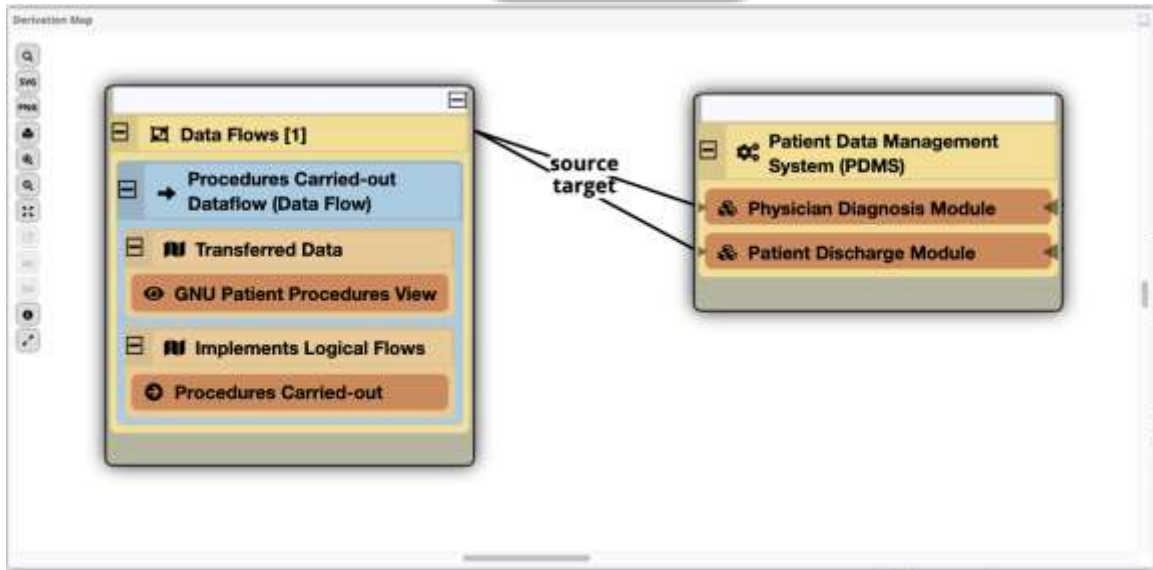
Digging Deeper from Logical (what) to Physical (how) Levels



How a "Patient Discharge Form" gets created

Data flows

Logical flows

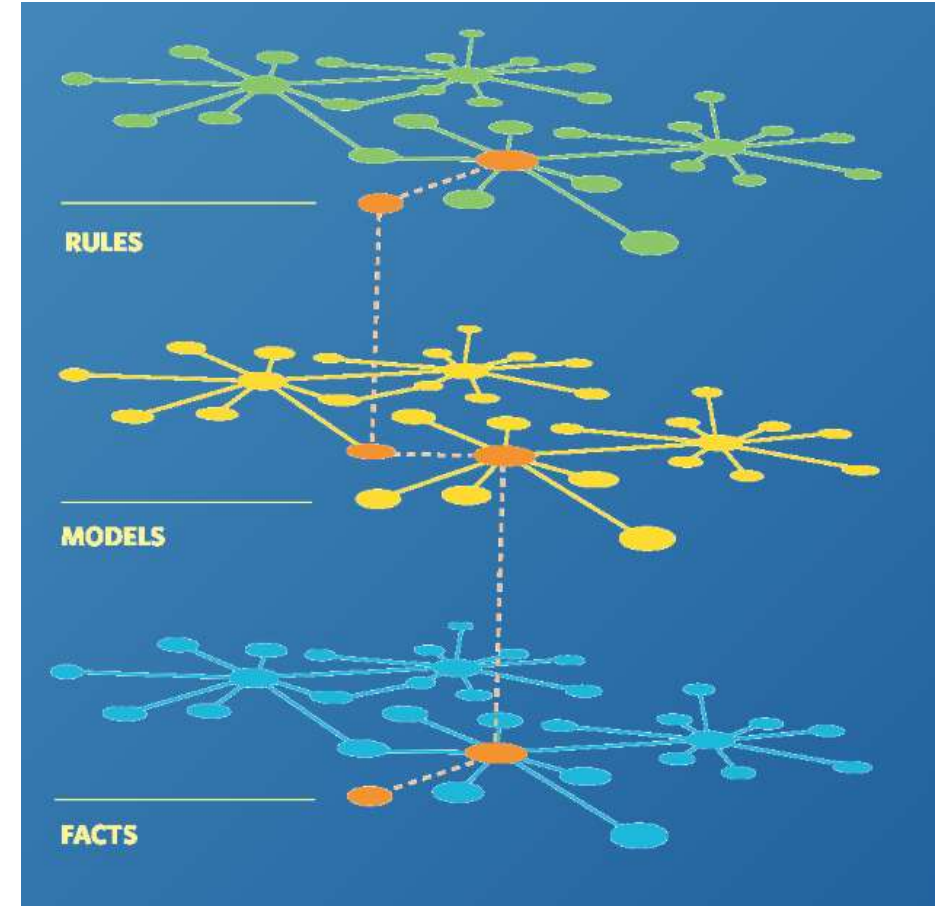


Concluding Remarks

- Practices around RDF stack have matured
- Being too different (and too academic) hindered adoption
- Move to the mainstream means aligning with and enhancing broadly used development and data management approaches
- Products like TopBraid EDG help this happen by embedding and facilitating best practices

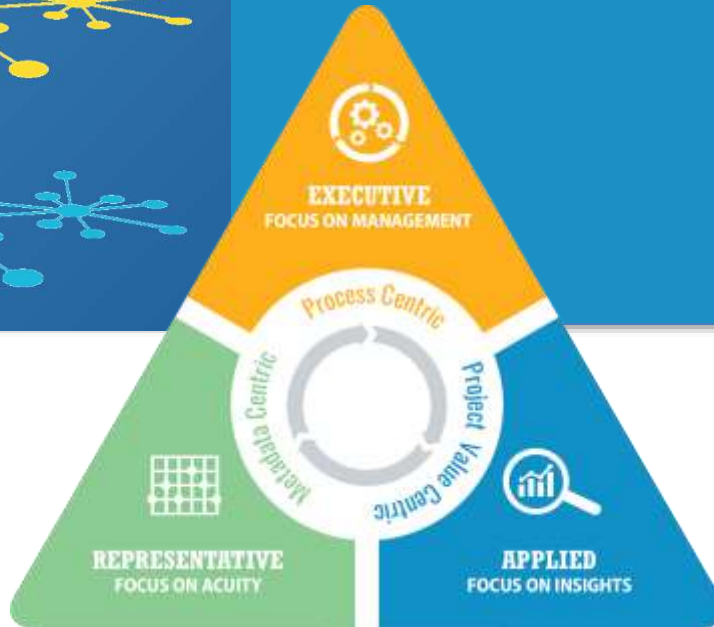
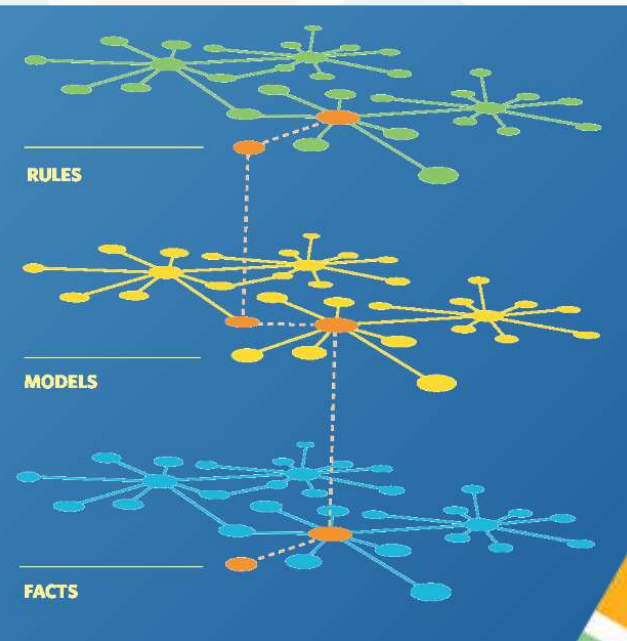
Benefits of a Knowledge Graph based Platform for Data Governance 2.0

- Flexibility and extensibility based on standards
- Integration of reasoning and machine learning
- Enabling people (UI) and software (APIs/web services) to view, follow and query
- Bridging of data and metadata “silos” to provide seamless data governance
- Delivery of Knowledge-driven data governance



As an enterprise knowledge graph infrastructure, TopBraid EDG supports Data Governance 2.0 and applications of AI / ML

Thank You !



... Questions?

To Learn More about TopBraid EDG and Knowledge Graphs:

EDG Product Info:

- [TopBraid Enterprise Data Governance \(TopBraid EDG\)](https://www.topquadrant.com/products/topbraid-enterprise-data-governance/)
(<https://www.topquadrant.com/products/topbraid-enterprise-data-governance/>)

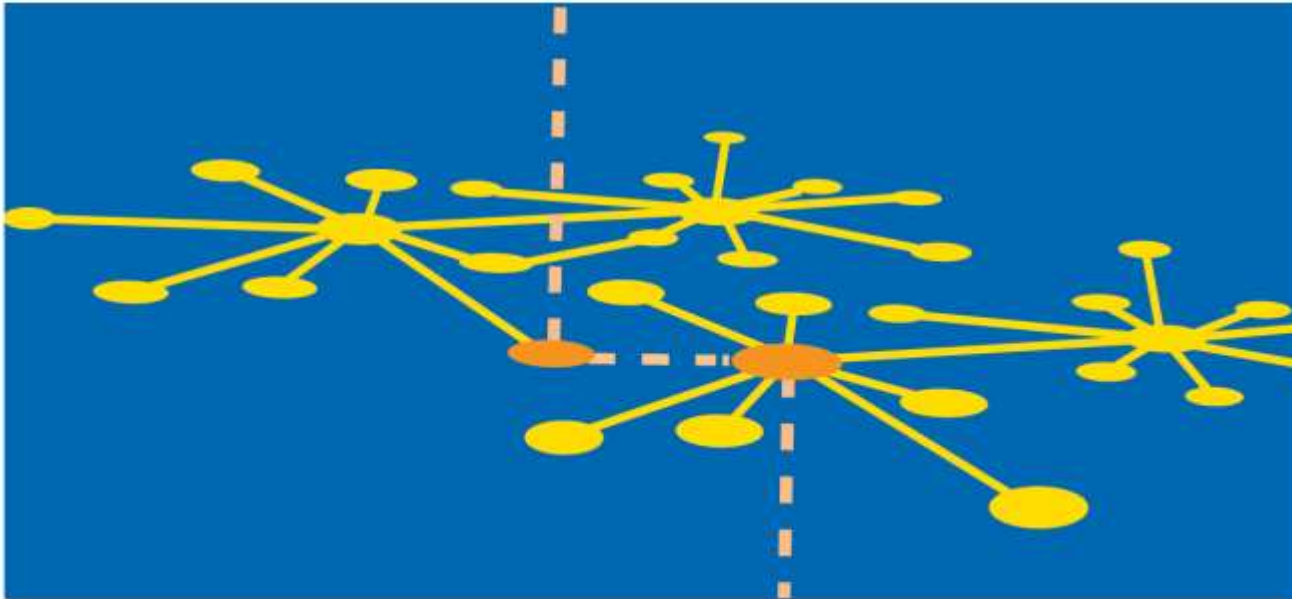
Contact us: at info@topquadrant.com to:

- Discuss data governance and knowledge graphs
- Request a more targeted demo of TopBraid EDG
- Ask for a free EDG evaluation account

Latest Whitepaper



Knowledge Graphs versus Property Graphs: Similarities, Differences and Some Guidance on Capabilities



Download a copy on our website at:
topquadrant.com/knowledge-assets/whitepapers/

More Resources ...

More Webinar Recordings, Slides, Q&A:

- <https://www.topquadrant.com/knowledge-assets/topquadrant-webinars/>

Short Videos:

- TopBraid EDG “Quick Grok” Videos
<https://www.topquadrant.com/knowledge-assets/videos/>
- TopBraid EDG Animated Video
https://www.topquadrant.com/project/edg_agile_modular/

Blog:

- <https://www.topquadrant.com/the-semantic-ecosystems-journal/>

Data Governance White Papers

- <https://www.topquadrant.com/knowledge-assets/whitepapers/>

References - SHACL

- SHACL:

- Constraints:

- <https://www.w3.org/TR/shacl/>

- Functions and Rules

- <https://www.w3.org/TR/shacl-af/>
 - <https://www.topquadrant.com/graphql/values.html>

- Reification with SHACL:

- <http://datashapes.org/reification.html>

References - GraphQL

■ GraphQL

— An introduction to GraphQL in EDG:

- <https://www.topquadrant.com/technology/graphql/>

— Querying using GraphQL:

- <https://www.topquadrant.com/graphql/graphql-queries.html>

— TopQuadrant's Webinar on GraphQL:

- <https://www.youtube.com/watch?v=Muj0m8Qrdig>