Master Thesis

Leveraging Semantic Datalake Capabilities Using Knowledge Graphs and Graph Technologies

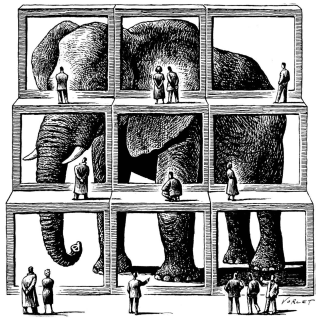
Evangelos Garaganis

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
MSC IN INFORMATION SYSTEMS DEVELOPMENT AND SECURITY

Section I

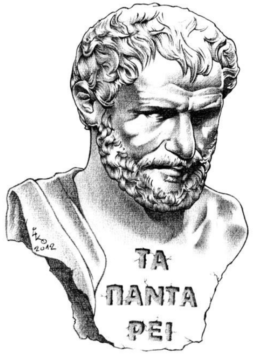
Cultivating towards the current semantics anode

Data are characterized by the following critical dimensions:



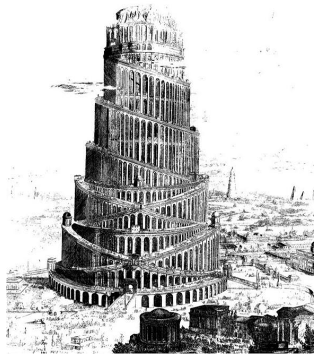
Data are contextual

Only when data are presented within context, does it become meaningful.



Data are dynamic

Data are ongoing and subject to change.



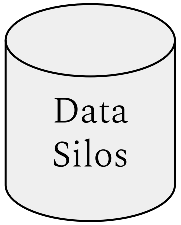
Data are diverse

Data are heterogeneous.

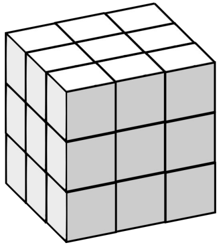
Data are connected

Data reside in highly complex network of data elements.

The need to sort, process, filter and analyze data against those dimensions has persistently vexed humanity the last decades.



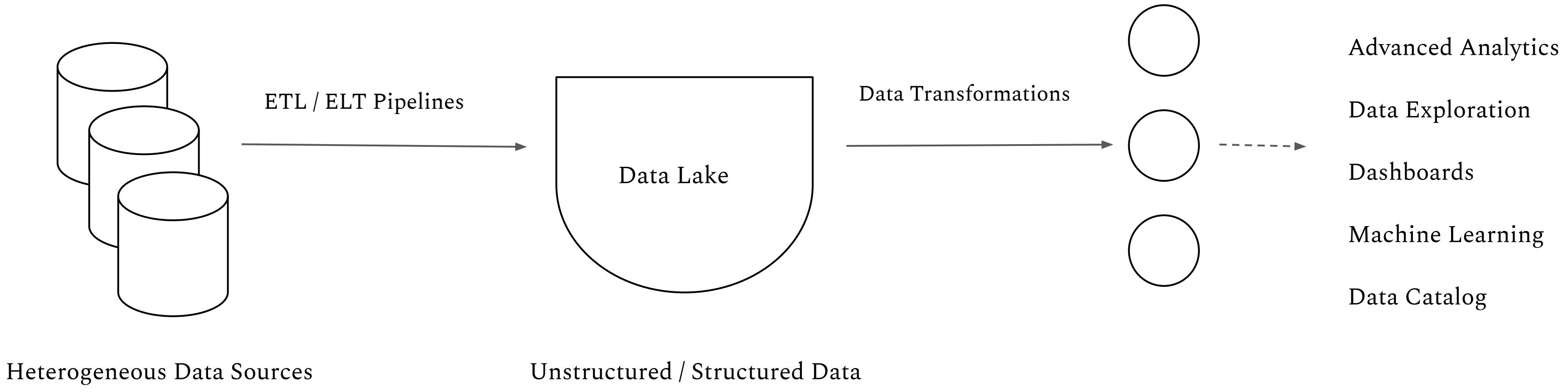
Organizations and individual departments started by relying on **data silos** as their chosen method for handling this challenge. While data silos granted organizations the capacity to conduct business intelligence data analysis, their segregation from the rest of the organization units hindered collaboration, data sharing and visibility across different departments, leading to duplication of effort, inconsistencies and data quality issues.



Data Warehouse

In response to the insular characteristics of data silos and the exponential growth in data volume, **data warehouses** gained prominence. These data repositories accommodated a multitude of data sources, both internal and external, allowing users to extract data from diverse origins, convert it into a structured format, and load it into the warehouse. This development afforded organizations a unified, single source of truth throughout their domain, along with robust querying and data analysis capabilities. A significant drawback associated with data warehouses was the necessity for intricate preprocessing and the establishment of data integration or ingestion pipelines to structure data prior to its entry into the data warehouse. This entailed complex Extract, Transform, Load (ETL) processes, rendering data ingestion a costly operation that demanded substantial reworking to align with a 'meaningful' data model.

In response to data warehouses, **data lakes** emerged. Data lakes are centralized data repositories for storing large and heterogeneous sets of data, that can be structured, semi-structured and unstructured. In contrast to data warehouses, data lakes are highly accessible and quick to update.



Data are dumped into the lake unstructured and are transformed on demand for various analysis and statistics reasons, including machine learning, predictive analytics, data discovery and profiling. They also offer means to securely store and catalog data, while favoring data movement and data transformations.

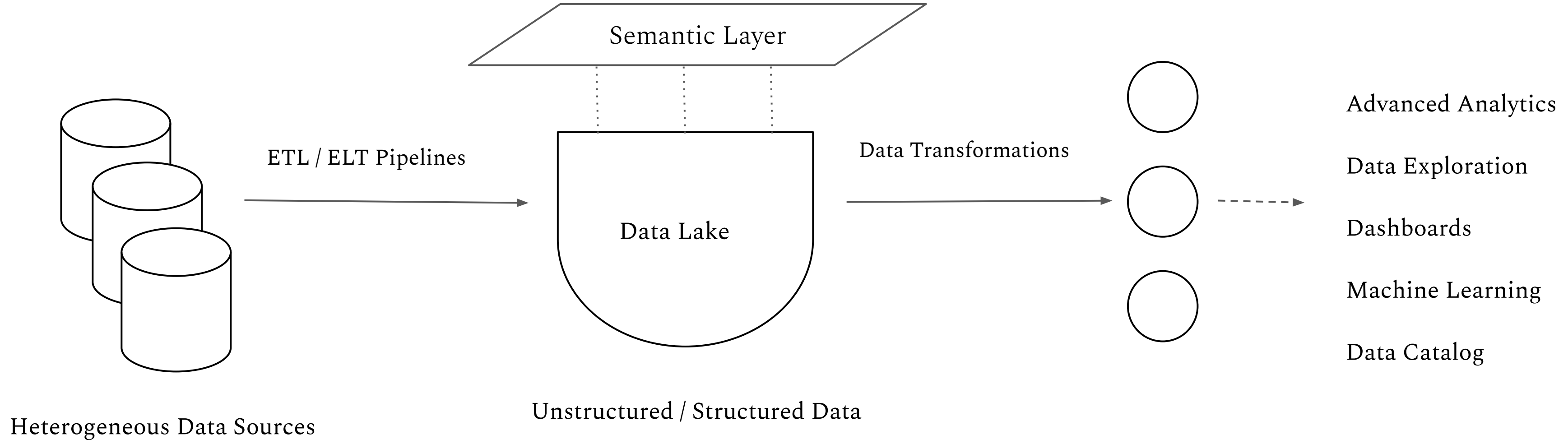
While data lakes afford users the capability to efficiently import large volumes of data at a minimal cost, indiscriminately depositing data into a data lake can render it inaccessible, cumbersome, and ultimately unproductive, leading to the emergence of a **data swamp**.

Without descriptive metadata, ELT can tank data quality, as data that come into warehouses without cleansing, duplicates, makes it hard for audit and managing authorization realms. For a data lake to operate effectively, it must ensure the integrity of its data assurance, provenance / lineage, and governance components.

*"What's a data lake for?
So you can drown in more data even faster!"*

It seems that while humanity has acquired the capacity to efficiently store and effectively access and query vast amounts of data, there remains a substantial gap in our ability to comprehensively process this data within a broader context.

In order to address this limitation, there is a growing trend towards relying on a **semantic layer** built on top of the data fabric within data lakes, that could enhance our comprehension of the interconnections among the data elements within a dat alake, ultimately yielding valuable insights.



In their paper "**Enriching Data Lakes with Knowledge Graphs**" [1], there is a list of semantic utilization examples within the context of a Data Lake.

Using Knowledge Graphs to Manage a Data Lake et al. [4] discussed how to address data findability, accessibility, interoperability, and reuse for data stored in a data lake. They showed the benefits provided to a data lake through the Support of ontologies and knowledge graphs which provide cataloguing of data, tracking provenance, access control, and semantic search. In particular, they built the DCPAC ontology (Data Catalog, Provenance, and Access Control) related to the management of data produced by vehicles.

Applying semantics to reduce the time to analytics within complex heterogeneous infrastructures et al. [7] to reduce the time from the collection to the analysis of data, they centralised the data in a data lake. Instead of populating the data lake of unstructured data, they proposed a semantic data platform called ESKAPE for the semantic annotation of the ingested data. Furthermore, a knowledge graph has been defined to act as an index that evolves over time according to the data that are included. In this way, users can easily identify and analyse the data coming from the different places.

Personalised exploration graphs on semantic data lakes et al. [8] proposed a semantics-based approach for the personalised exploration of data lakes within the domain of smart cities. First, they provided the data lake with a semantic model using domain ontologies. Then, another ontology was adopted to describe indicators and analysis dimensions. Finally, personalised exploration graphs were generated for different types of users.

Semantic profiling in data lake (2018) et al. [9] proposed a semantic profiling tool for metadata extension in data lake systems. Its aim was to understand the meaning of data. Their tool recognised the meaning of data at schema and instance level using domain vocabularies and ontologies

A semantic data lake model for analytic query-driven discovery et al. [6] presented a semantic model for the correct data fruition stored into a data lake. They mapped the indicators of interest, the dimensions of analysis and formulas into a knowledge graph to support the correct identification of data.

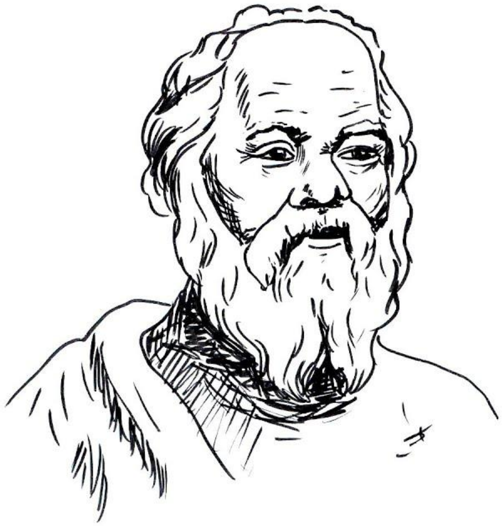
Strategies for a Semantified Uniform Access to Large and Heterogeneous Data Sources et al. [10] proposed a physical and a logical data integration whose goal was to query large and heterogeneous data sources. For the physical data integration they defined an ontology to transform the data into RDF.

Enriching Data Lakes with Knowledge Graphs et at. [1] proposed a methodology to extend a data lake containing data extracted from touristic platforms with a semantic layer and produce a knowledge graph. They engineered an ontology in the touristic domain integrating already existing ontologies and extending them with classes. However, the focus of this manuscript is not on the ontology but on the extracted knowledge graph and the steps we performed to transform the data from the data lake to the knowledge graph.

But since semantics is an overloaded term, what does the semantic layer construction actually mean ?

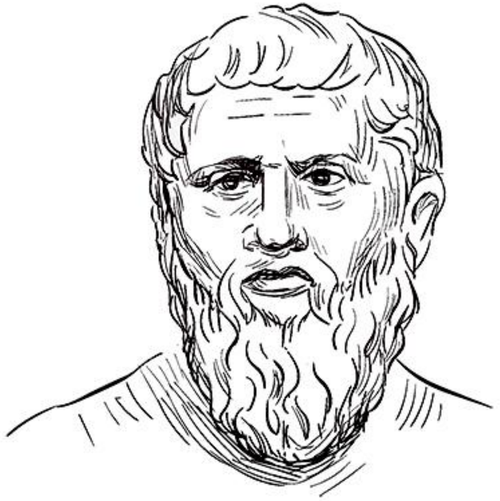
"As your new CEO, I'd like to make my data strategy clear"

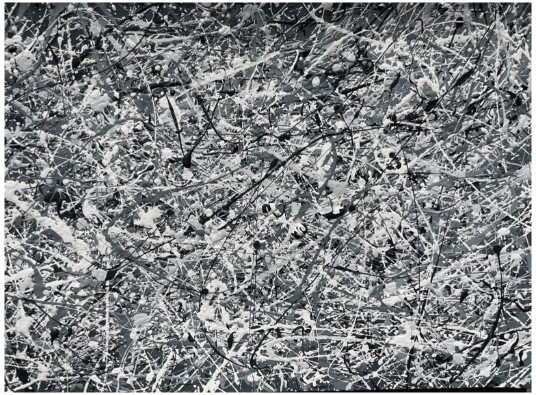
The foundations of knowledge and semantics were established between 300-400 BC by Socrates and Plato. They introduced the initial categorization of knowledge into two distinct realms: ideas, which are abstract, and things, which are concrete and observable. Ontologies are derived from this fabric.

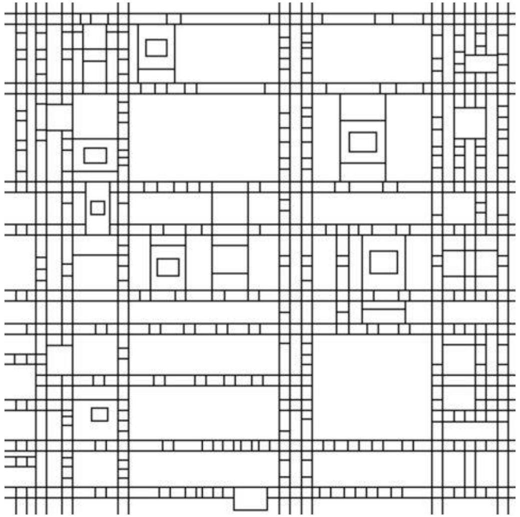


“Well, Plato, if you create an ontology and nobody understands it, does it even exist?”

"Socrates, it exists, but it's like a philosopher's joke without a punchline – you know it's there, but it just leaves everyone scratching their heads!”







This thesis found its articulation of semantics within the framework of a graph, notable a knowledge graph.

Graphs represent the inherent structure within our data realms.

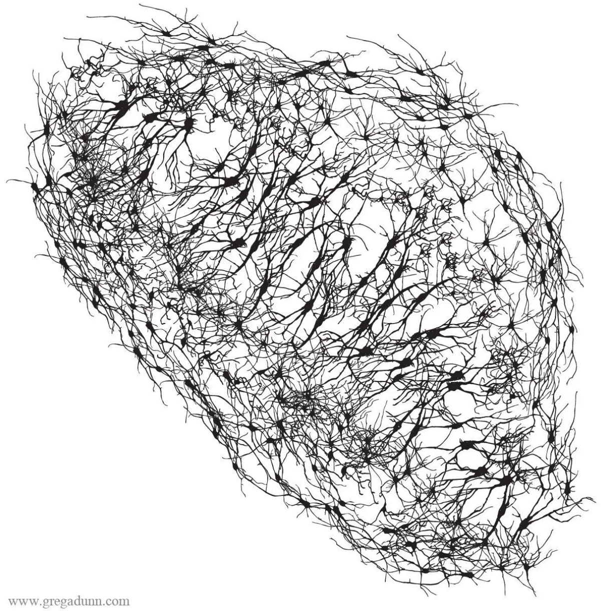
At a conceptual level, schemas and ontologies or other types of mappings establish a contractual agreement between humans and machine models. This agreement encompasses multiple relationships that encapsulate diverse semantic constructs, including but not limited to inheritance, taxonomies, and so forth.

Modelled entities are been instantiated in the form of objects that are stored in volatile or non-volatile format, can also contain multiple references that can exponentially grow in size. Data undergo continuous modifications or conversions, transitioning from one state to another. This transformation occurs during data integration phases within transformation pipelines and also arises through events, resulting in the accumulation of a historical record of alterations.

Finally, individuals engage in project-related endeavors, delivering these projects to other stakeholders through various artifacts, fostering communication among the involved parties.

Drawing from this philosophical standpoint, having a graph to counter this entropic tendency of data to relate, transform and flow, imparts a dynamic contextual understanding of our data, which can offer assistance in:

- Modeling real-world information in way closer to brain's mental model of the world.
- Data exploration and deep understanding of the data.
- Managing and storing heterogeneous data.
- Performing logical reasoning (algorithmic decision making, pattern finding, etc).
- Richer data management, personalization and recommendations.
- Enable graph algorithms utilization.
- Removing redundancy, compared to e.g. tabular data.
- Furnishing expressive graph queries.
- Using automated tasks from different software.


www.gregadunn.com

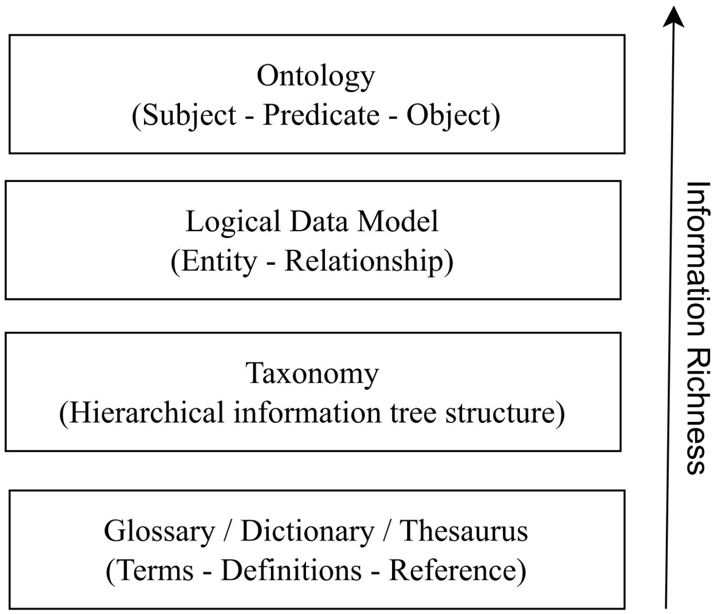
Due to these rationales, this thesis has opted for a graph-based approach to capture the emanating semantics and implement the semantic layer with a knowledge graph. A knowledge graph can help us surface hidden lines of communication and identify facets or disconnected information.

Section 2

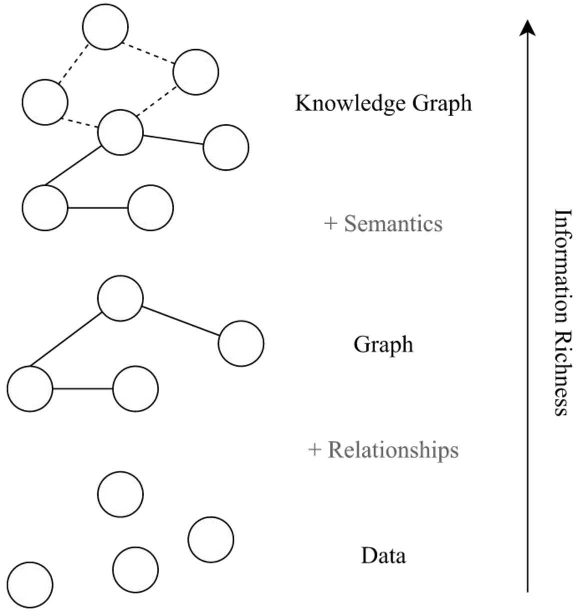
Using knowledge graphs to implement data lake’s semantic layer

In their book ”Knowledge Graphs: Data in Context for Responsive Businesses”, Jesús Barrasa, Amy Hodler & Jim Webber shed ample light on knowledge graphs and how to implement the semantic layer with the help of knowledge graphs.

Adding a semantic layer atop data is a process where we enrich data with semantics. This happens by applying a set of organization principles upon data, such us dictionaries, taxonomies, logical data entity - relationships models, ontologies.


Ontology
(Subject - Predicate - Object)
Logical Data Model
(Entity - Relationship)
Taxonomy
(Hierarchical information tree structure)
Glossary / Dictionary / Thesaurus
(Terms - Definitions - Reference)
Information Richness

In a similar manner as in the previous proposition, we can add a semantic layer atop our graphs making it a knowledge graph. This happens by applying the aforementioned organization principles of the previous section on our graph and transforming it to a property graph model. This means that:



1. We can store information, semantics and meta-data on graph nodes or edges.

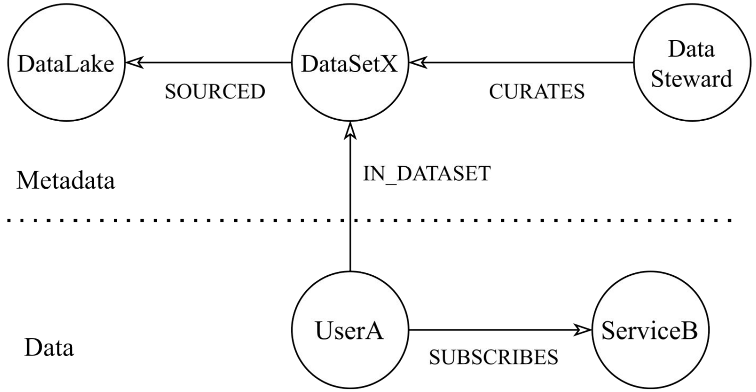
2. Instead of storing only the objects in graph format, we can also introduce newly created nodes that represent different semantics or metadata, such us the model the each object is instance of, or other organizational tools, such as thesaurus.

3. Convey on the same graph other organizational possibilities, including but not limited to inheritance properties, diverse taxonomies, custom ontologies, and associations among objects.

There are two types of knowledge graphs proposed in the book: (a) Actioning Knowledge Graphs and (b) Decisioning Knowledge Graphs.

Actioning Knowledge Graphs

Information and people tend to become siloed. This results into disjointed data domains that undermine the radical visibility of organization's data ecosystem. An Actioning knowledge graph help us understand that ecosystem, by capturing its semantics from the production of a data point to its consumption from various endpoints and provides the ability to invoke action. With an actioning knowledge graph we can connect data with metadata in a non-invasive manner. This knowledge graph of metadata offer a connected view of the integrated domains by combining data stored in a local graph with data retrieved on demand from third-party systems.



1. For data integration we can relate datasets to their originated data sources.

2. For data lineage we keep track about data's historical archive.

3. Semantically enrich existing data by linking them to vocabularies, taxonomies, and ontologies to enable interoperability.

An Actioning Knowledge Graph designed for metadata management serves as a catalyst for fostering confidence in data and facilitating self-service data utilization throughout the organization. This type of knowledge graph constitutes the fundamental underpinning for critical data management functions such as data quality assurance, data stewardship, and data governance.

More specifically, an Actioning Knowledge Graph can be used for:

Data Lineage: Traces all steps in data pipelines from data sources to data consumers to provide trust and high-fidelity provenance information.

Data catalog: Actionable inventory of all data assets with their detailed structure.

Data Insights: Impact analysis and root cause analysis, by exploiting the transitive dependencies modeled as relationships in the underlying graph.

Information Search: Actioning knowledge graphs are also used to enhance information search. Documents, lessons learned, and knowledge in general can be indexed in a knowledge graph, making it possible to search for things instead of strings.

Single view of X: Also known as X360, the actioning kg provides trusted views and a contextualized understanding of X and consequently provides the right data and context from which to suggest actions.

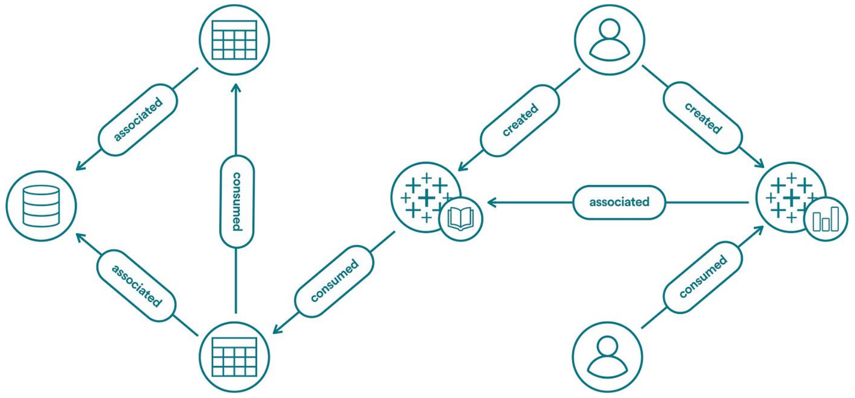
Democratizing Data at Airbnb: Dataportal

Dataportal is a data resource search and discovery tool that democratizes data by empowering data exploration, discovery and trust. The explosive growth in data and internal data sources (data tables, dashboards, reports etc) :

• Makes difficult to employees to navigate the data landscape.
• The absence of metadata and contextual information has eroded trust in the data, making it challenging for employees to rely on information outside their areas of expertise.
• The fear of using outdated or erroneous data has led to the creation of redundant resources, further complicating the data ecosystem.

This led to information and individuals to become siloed, forcing them to navigate an opaque landscape of specialized knowledge, providing a myopic localized view of the dataspace while lacking global context.

In order to stop relying on tribal knowledge that undermined data discovery, Airbnb created an actioning knowledge graph with the goal to integrate the data-space and offer a holistic and single-lens view to users, providing them with the essential context to be data informed and feel confident about its trustworthiness and relevance. Employees can search schemas, data tables etc, and further explore the surfaced metadata, in order to remove the consolidation barriers.



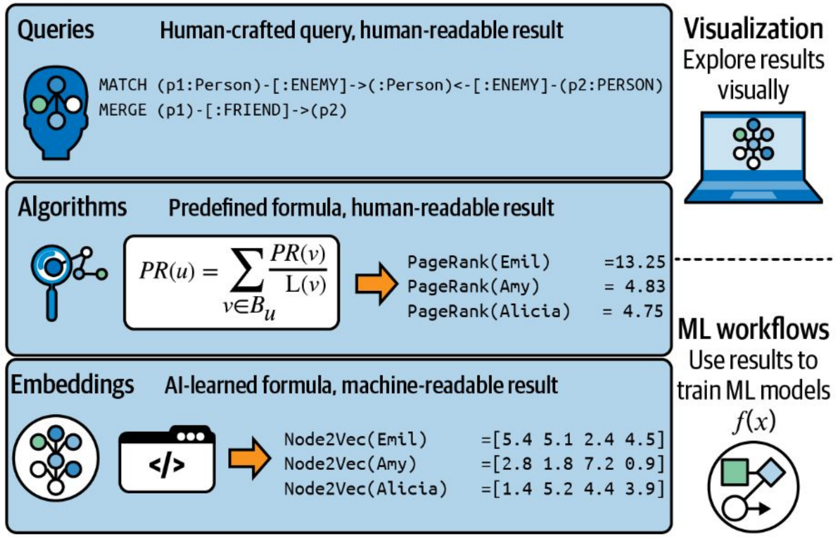
Decisioning Knowledge Graphs

The processing of graph data commonly employs methodologies derived from graph analytics, graph machine learning, and graph data science. These techniques excel in identifying unobvious connections by revealing patterns within our data. Processing data based on relationships that connect it can help us answer specific questions using existing data, discover how the connections in that data might evolve and generally help us gain insights.

The Decisioning knowledge graph is a type of knowledge graph that employees graph data science and graph machine learning, in order to improve decisions made by human or software agents. A Decisioning knowledge graph supports:

Graph Queries: Graph queries are used for real-time pattern matching, with the aid of a graph database and a dedicated graph query language.

Graph algorithms: Graph algorithms excel in detecting global patterns and trends, but the selection and fine-tuning of these algorithms should align with the specific questions at hand. A decision-making knowledge
graph should support a variety of algorithms and allow for customization to accommodate future growth.



Queries
Human-crafted query, human-readable result
Algorithms
Predefined formula, human-readable result
Embeddings
AI-learned formula, machine-readable result
Visualization
Explore results visually
ML workflows
Use results to train ML models
$f(x)$

```
MATCH (p1:Person)-[:ENEMY]->(:Person)<-[:ENEMY]-(p2:PERSON)
MERGE (p1)-[:FRIEND]->(p2)
```

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

```
PageRank(Emil)      =13.25
PageRank(Amy)       = 4.83
PageRank(Alicia)    = 4.75
```

```
Node2Vec(Emil)      =[5.4 5.1 2.4 4.5]
Node2Vec(Amy)       =[2.8 1.8 7.2 0.9]
Node2Vec(Alicia)    =[1.4 5.2 4.4 3.9]
```

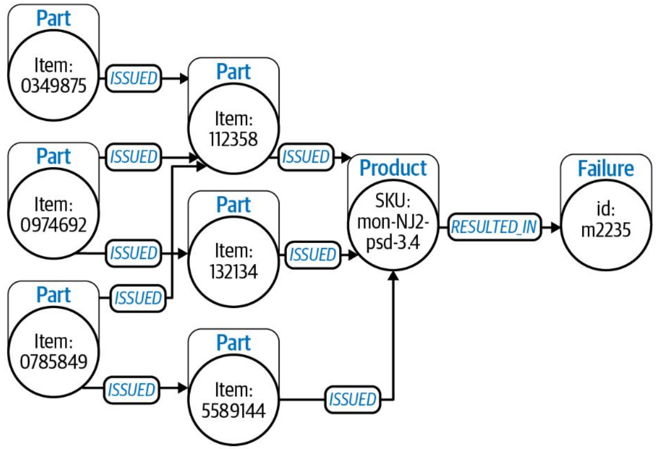
Graph Embeddings: In addition to enhancing our understanding of data, the results of graph queries and algorithms can be leveraged to train machine learning (ML) models. Graph embeddings are algorithms that encode the structure of a graph, encompassing its nodes and relationships, into a format that can be readily utilized by ML processes, thus utilizing the graph's inherent structure as a predictor.

Graph visualization: The visualization of data enables to delve deeper into connections and draw meaningful inferences. Exploring graphs intuitively involves actions such as traversing relationships, expanding scenes, refining perspectives, and following specific paths. An effective visual representation of a graph must possess dynamic qualities and customization options, empowering users to engage interactively.

Boston Scientific Decisioning KG

Boston Scientific is a global medical device company with million patients using its products. Predicting and preventing device failures early in the process of supply chain is crucial. In order to pinpoint the root cause of deflects, it created a decisioning knowledge graph, where they included parts, finished products and failures.

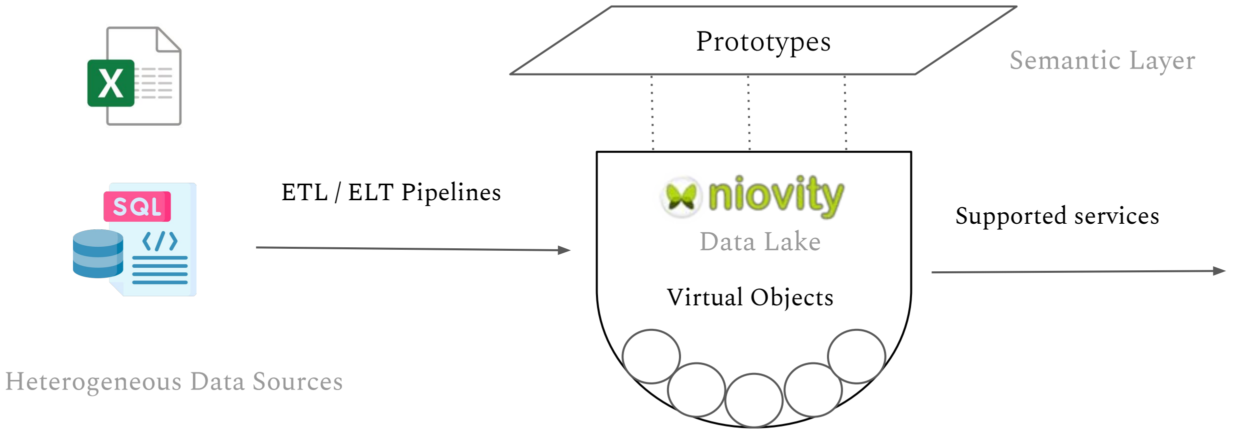
Using graph queries, Boston Scientific is able to quickly reveal subcomponents, complex relationships and trace any failures to relevant parts. The company was able to identify previously unknown vulnerabilities by adding graph algorithms to rank parts based on their proximity to failures and match other components based on similarity.



Section 3

The case study on Niovity’s semantic data lake

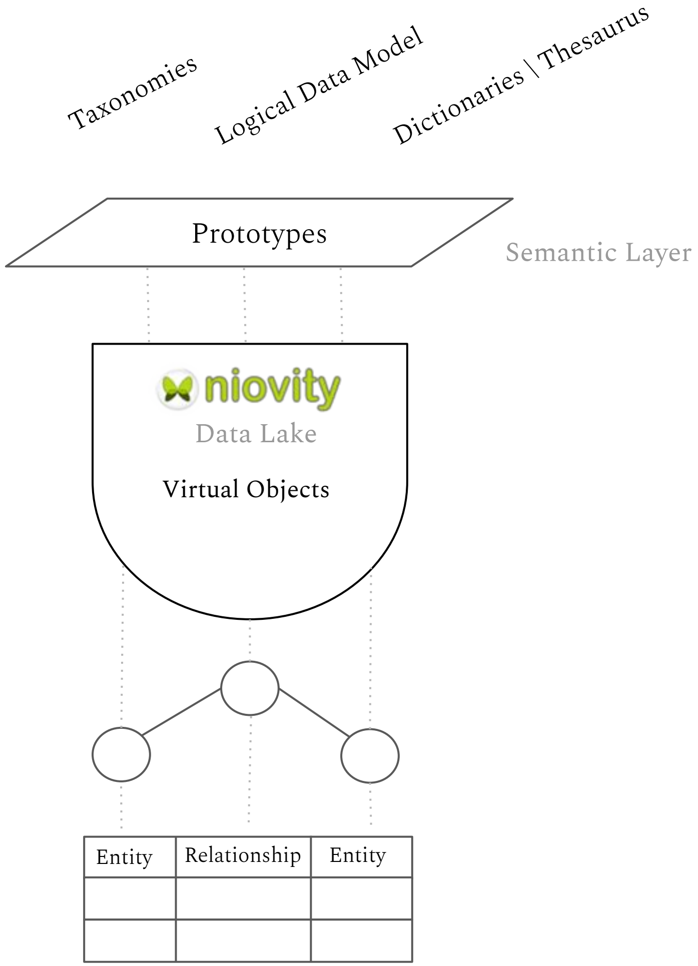
Niovity is a multi-tenant data platform that presents an extensive array of solutions, ranging from institutional repository management to comprehensive software development services, including Backend-As-Service solutions.



- Rich data model
- ACID CRUD api with batch capabilities.
- Powerful search api with expressive DSL
- Minimal vendor lock in with data export
- Code generation in Java
- ACID transactions
- Middleware and backoffice exposed backends
- Digital Asset Management through filestores
- Event stream
- Data vocabularies
- Data analytics and visualizations
- Data and user authentication
- JVM and JS integration
- CMS and website management

Data from heterogeneous sources are being modeled by custom domain ontologies / prototypes and stored centrally within datalake in the form of virtual objects. Prototypes, which cover each project's domain of knowledge, are defined by a subset of different fields and facilitate inheritance or custom metadata incorporation.

The entire data management lifecycle of the datalake and Niovity's development and operational framework, which encompasses tasks such as record instance management, data integration, querying, data analytics, and development, is being executed in a model-driven fashion guided by prototypes.



The term ”semantics” is a conceptual notion with diverse interpretations. In the context of Niovity semantic data lake, semantics find expression through:

1. Prototype definitions model the domain of knowledge by forming ontologies supported by a rich logical data model.
2. Prototype definitions encapsulate various semantics, ranging from custom metadata to taxonomies or thesaurus definitions.
3. Throughout its data management lifecycle there are various metadata produced and stored in various formats and stores.
4. Cross objects relationships along with their surrounding context or semantics are captured in the form of a knowledge graph stored in a triple store.

Those semantics are maintained and managed by Niovity’s core framework, DOLAR, along with its simple knowledge graph implementation that is stored in the form of tripe store.

Prototype definitions

Data outlive applications: data models should be expressed in isolation of any application and transcend the application boundaries. For these reasons, prototypes model the specific domain of knowledge. They form the ontological semantic core within Niovity's data lake.

Each prototype is defined by a subset of field definitions, it can capture various semantics, including custom metadata and taxonomies while also forming cross prototype relationships.

Prototypes are the driving factor for Niovity's development and data management lifecycle, including generated CRUD, Search, Data analytics, Java APIs.

```xml
<prototype>
    <inherits id="/butterfly/core/Container"/>

    <label lang="en">Doctoral Dissertation</label>

    <meta id="Factory"/>
    <meta id="Indexable"/>

    <field id="creationYear">
        <label lang="en">Year</label>
        <description lang="en">Fill out the creation year of
            the dissertation.</description>
    </field>

    <field id="title">
        <label lang="en">Title</label>
        <description lang="en">Fill out the title of the
          dissertation.</description>
    </field>

    <field id="academicUnit">
        <label lang="en">Academic unit</label>
        <type>REF</type>
        <constraint>Unit</constraint>
    </field>
</prototype>
```

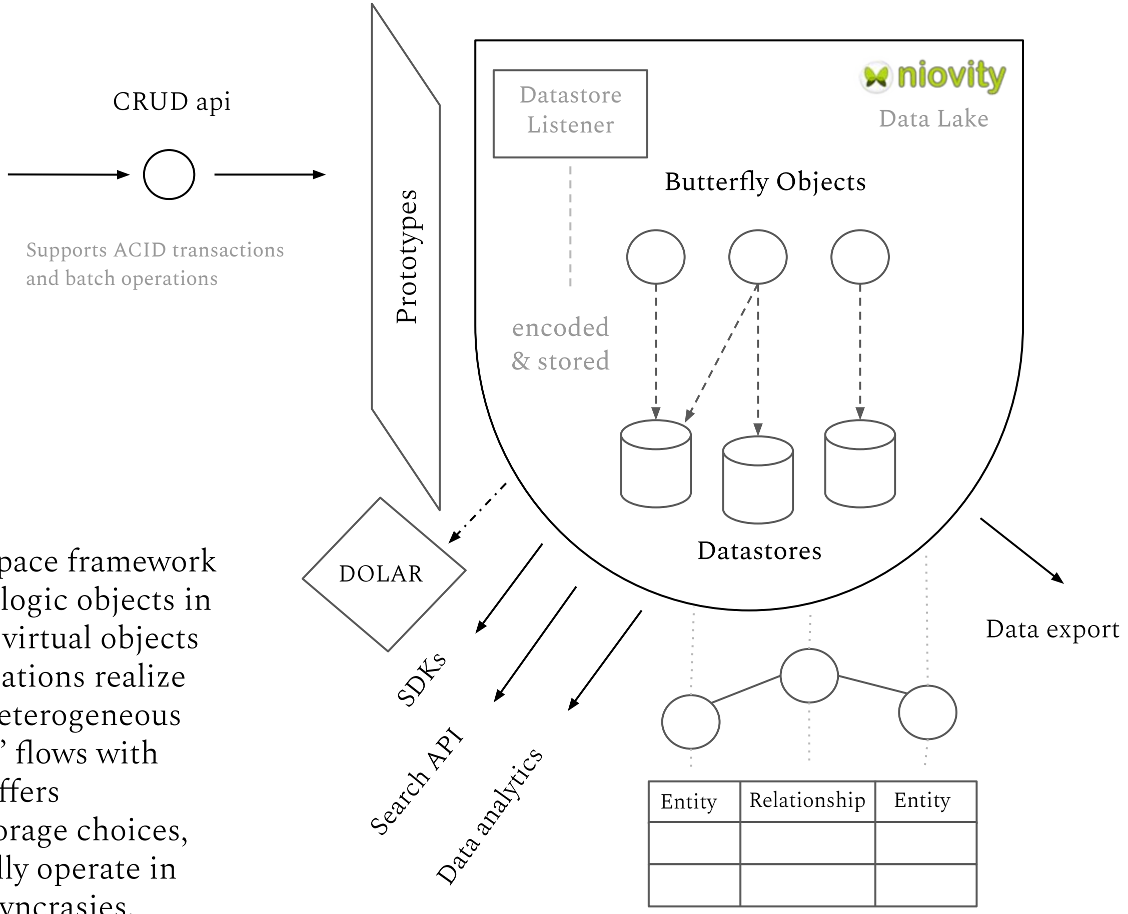
```xml
<field id="parents">
    <label lang="el">Parent</label>
    <label lang="en">Parent</label>
    <type>REF</type>
    <array>true</array>
    <constraint>/butterfly/core/Node</constraint>
</field>
```

Butterfly Objects

Butterfly objects or virtual objects are prototypes instances. Butterfly objects are stored in a data-store agnostic manner to different datastores and can be encoded in various formats, given the application context and needs. They are characterized by their object coordinations (store name and id).

DOLAR

DOLAR is a service-neutral virtual information space framework that automates the introduction of new business-logic objects in terms of virtual “content objects”. User-specified virtual objects are connected to storage artifacts and help applications realize uniform “store-to-user” information flows atop heterogeneous sources, while offering the reverse “user-to-store” flows with identical effectiveness and ease of use. DOLAR offers mechanisms to gradually support new types of storage choices, while virtual objects help business-logic essentially operate in isolation of any low-level information space idiosyncrasies.



Niovity's Search API

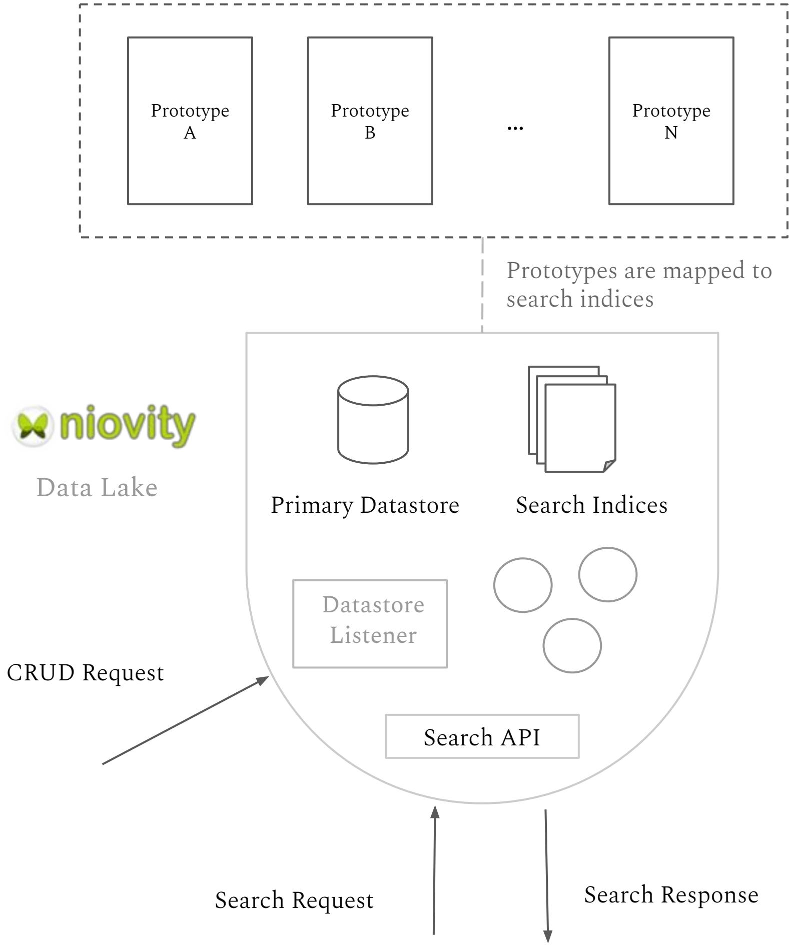
Niovity's search API offers offers a uniform search service for all applications.

Firstly, the service offers effective support for multiple, nested, and inter-connected data models, abstracting away the datastores indexing details (multiple indices, deeply nested documents, cross-index document relationships).

Secondly, the service exposes a simple, yet powerful, query DSL that hides the underlying datastores search/scroll APIs idiosyncrasies.

The current search API supports a handful search operators, including binary, fulltext, unary, graph and compound operators, while also supporting various aggregation capabilities.

The search implementation utilizes query federation mechanisms, to query data across multiple datastores, taking advantage of the unique features offered by each datastore, offering cross datastores decoding and encoding mechanisms to support uniformity.

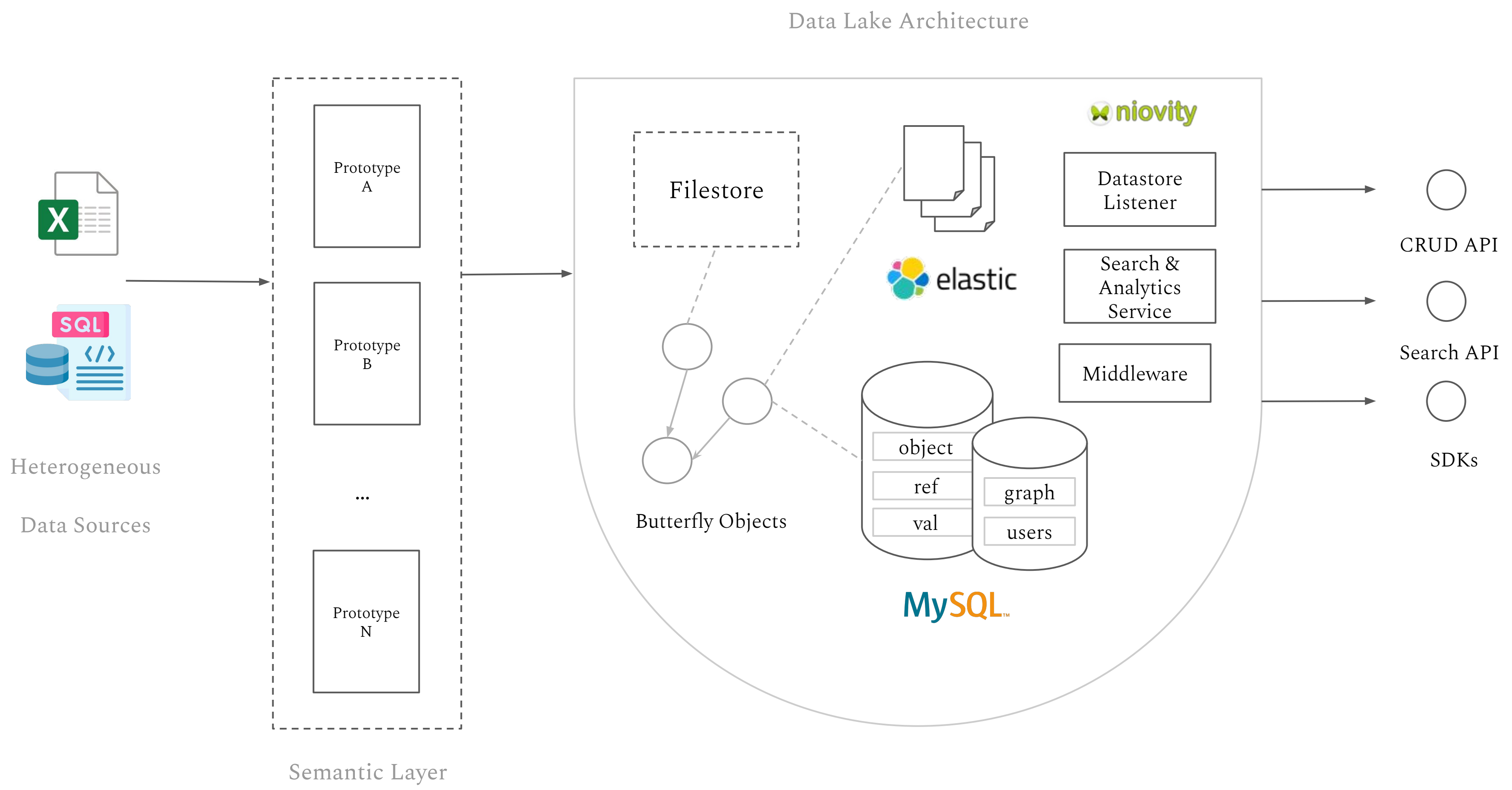


```json
{
    "start": 0,
    "count": 1,
    "models": [
        "Photo"
    ],
    "fields": [
        "subject",
        "photographer"
    ],
    "query": {
        "filters": {
            "exists": "photographer"
        },
        "aggregations": [
            {
                "id": "byPhotographer",
                "field": "photographer.Person.name::en",
                "type": "terms"
            }
        ],
        "sorting": [
            "title:desc"
        ]
    },
    "strict": false
}
```

```xml
<prototype>
    <inherits id="Model"/>
    <inherits id="CommonMetadata"/>
    <inherits id="/butterfly/core/PhotoContainer"/>

    <label lang="en">Photo</label>

    <meta id="Factory" />
    <meta id="Indexable" />

    <field id="caption">
        <label lang="en">Caption</label>
        <type>TEXT</type>
        <map>true</map>
    </field>

    <field id="depictedSubject">
        <label lang="en">Depicted subject</label>
        <type>TEXT</type>
    </field>


    <field id="photographer">
        <label lang="en">Photographer</label>
        <type>REF</type>
        <constraint>Person</constraint>
    </field>
</prototype>
```

```
{
    "start": 0,
    "count": 1,
    "models": [
        "Photo"
    ],
    "fields": [
        "subject",
        "photographer"
    ],
    "query": {
        "filters": {
            "exists": "photographer"
        },
        "aggregations": [
            {
                "id": "byPhotographer",
                "field": "photographer.Person.name::en",
                "type": "terms"
            }
        ],
        "sorting": [
            "title:desc"
        ]
    },
    "strict": false
}
```

```
{
    "total": 9961,
    "start": 0,
    "count": 1,
    "items": [
        {
            "id": "<objectIdentifier>",
            "storeName": "<dataStoreIdentifier>",
            "prototype": "<internalTypeIdentifier>",
            "publicType": "Photo"
        }
    ],
    "aggregations": {
        "byPhotographer": {
            "John Doe": 5261,
            "Hans Meier": 4700
        }
    }
}
```


Data Lake Architecture
Heterogeneous
Data Sources
SQL
Prototype A
Prototype B
...
Prototype N
Semantic Layer
Filestore
Butterfly Objects
elastic
niovity
Datastore Listener
Search & Analytics Service
Middleware
object
ref
val
graph
users
MySQL
CRUD API
Search API
SDKs

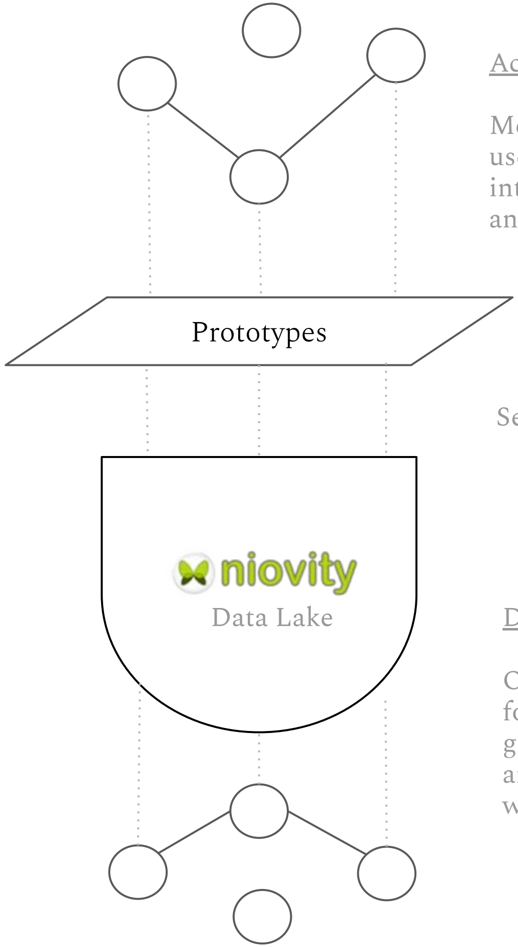
Niovity's data lake model-driven approach counters many of challenges imposed on data lakes. It offers rich data modeling and a robust architecture for data integration, reuse and interoperability, in order to favor the development of application ecosystems.

The current semantic layer, constituting of its (a) prototype modelling capacities (b) currently implemented knowledge graph (c) custom metadata capturing, along with the data lake's capabilities to handle data, agnostically to its datastores, forms a fertile groundwork for further utilization.

This thesis delves into the practical implementation of a Decisioning and Actioning knowledge graph atop Niovity's current data lake, in order to leverage its existing semantics and empower its query and analytics engine with the aid of cutting edge graph technologies. This could be beneficial for:

- Evolve and expand its current query and data analytics engine.
- Enable datalake with AI & ML workflows and pipelines.
- Suggest a disciplined workflow for metadata capturing, management and utilization.

The aim of this thesis is to create a unified knowledge graph, that would act as an integration joint and a sophisticated index, and supply its users with powerful tools to harness its value throughout the data, applications and operations lifecycle.



Actioning KG

Metadata capturing used for data lineage, integration, governance and housekeeping.

Semantic Layer

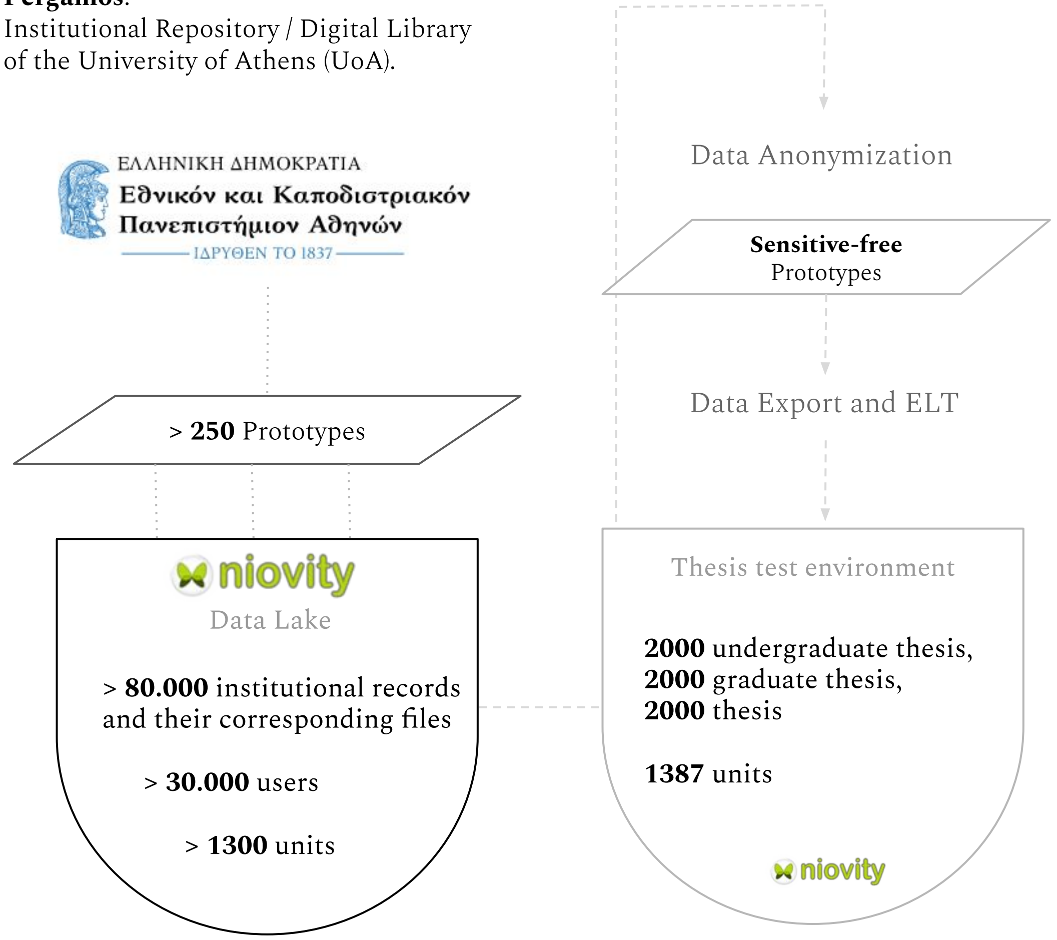
Decisioning KG

Objects graph allowing for data visualization, graph querying, graph analytics and ML / AI workflows.

Section 4

Building a unified knowledge graph atop Niovity's data lake

Pergamos:
Institutional Repository / Digital Library of the University of Athens (UoA).



In order to demonstrate its work, we utilized real datasets imported from the institutional repository of Pergamos, which serves as the digital repository of the Kapodistrian University of Athens.

Pergamos houses an extensive collection of both units and gray literature items.

To exemplify the implementation of the knowledge graph, we imported all units stored in Pergamos, along with a subset of gray literature items consisting of 2000 graduate theses, 2000 post-graduate theses, and 2000 theses.

These data serve as a pertinent use case, as they encompass diverse relationships and inheritance structures. Units are associated with other units, and gray literature items are linked to specific units, among other associations.

Building a decisioning knowledge graph

Graph Queries

Graph Data Science

Graph Traversal

Graph Visualization

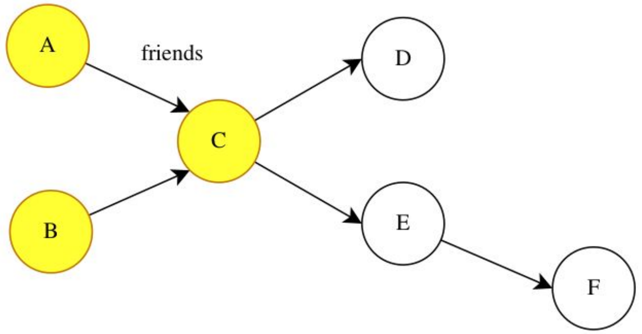
Graph Operator Syntax

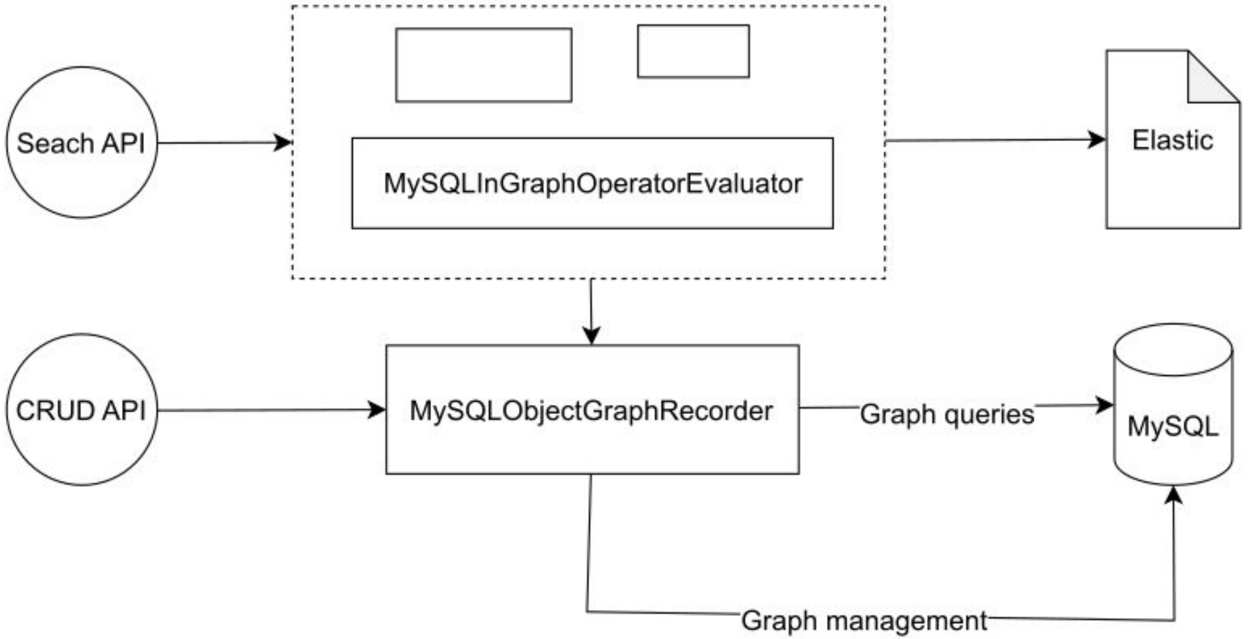
```
in_graph: {
   roots: <Search Input>,
   sinks: <Search Input>,
   follow: [<field_id>],
   transitive: boolean
}
```

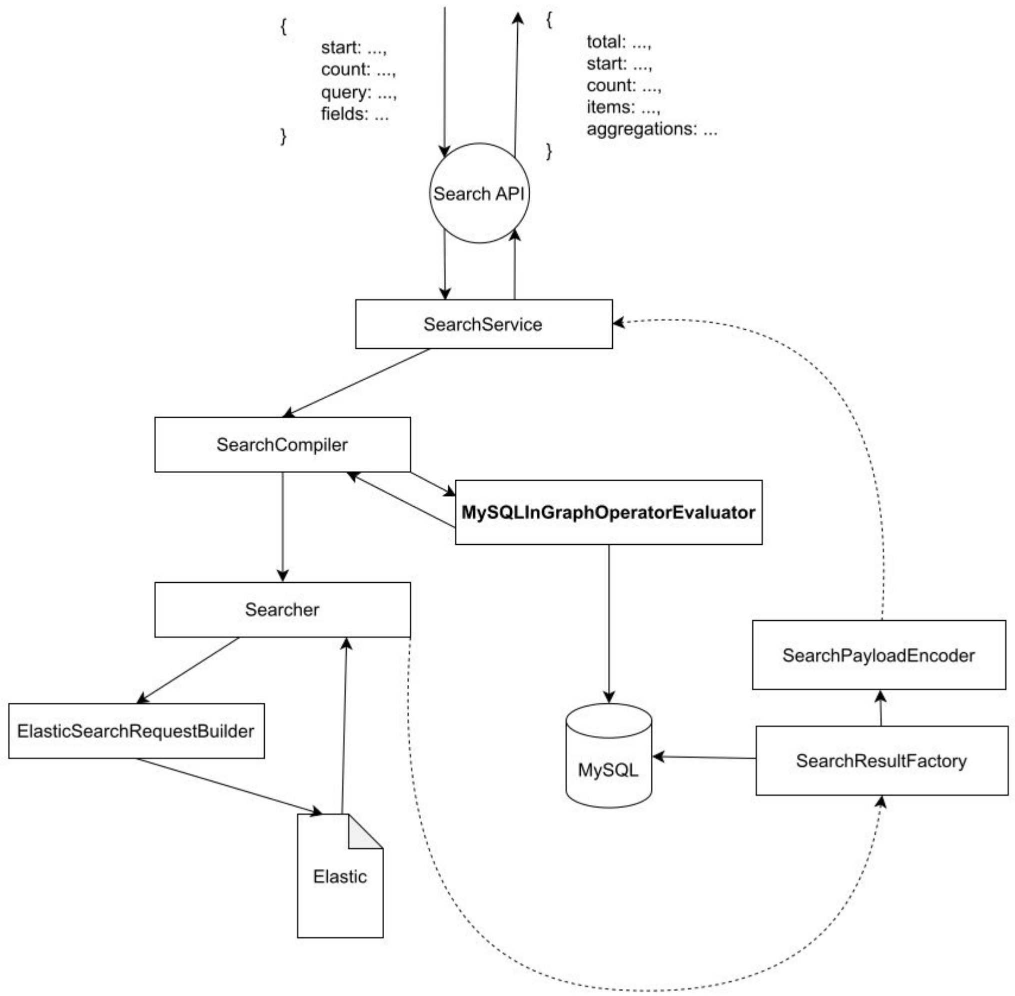
The graph operator starts by searching all root and sink objects that satisfy the ”Search Input” conditions and then finding paths that start from those roots or lead to that sink nodes, by following the fields specified (transitively or not).

Graph Search Example

```
{
     models: ["Person"],
     in_graph: {
         sinks: {
             eq: {
                 name: "C"
             }
         },
         follow: ["friends"],
         transitive: false
     }
}
```




Seach API
MySQLInGraphOperatorEvaluator
Elastic
CRUD API
MySQLObjectGraphRecorder
Graph queries
MySQL
Graph management

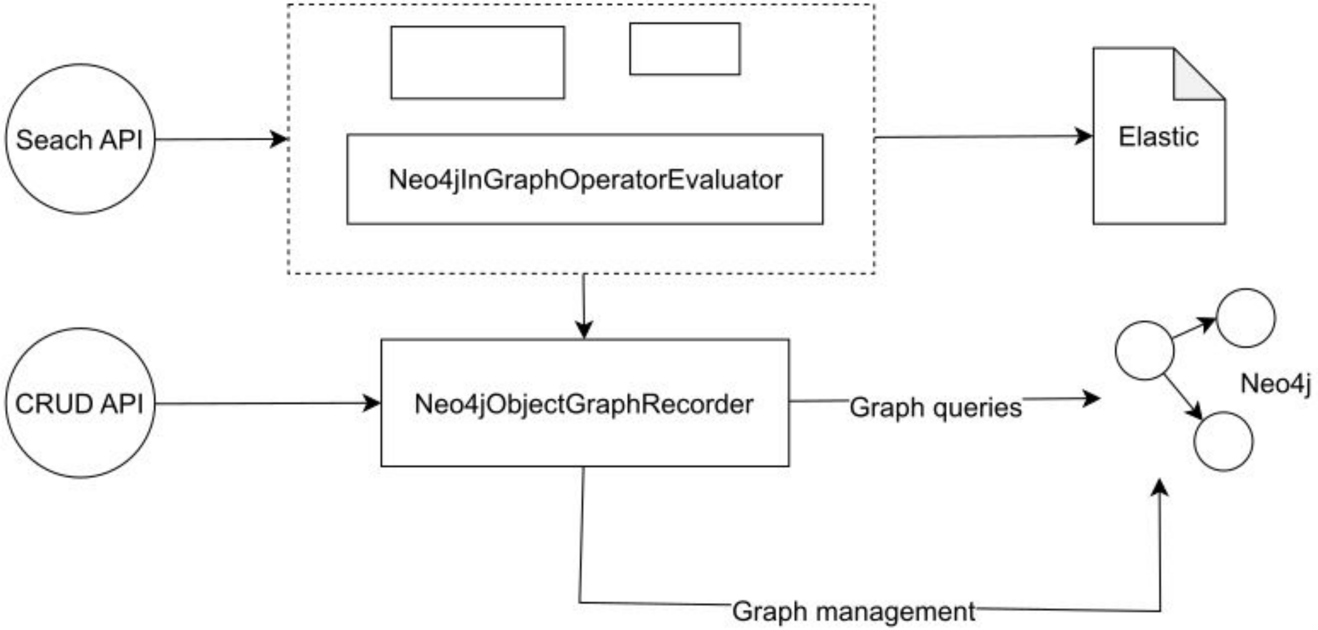


The graph operator is evaluated from the InGraphOperatorEvaluator. It will start by performing a search based on the root's and / or sink's search input, returning the records that satisfy the required search criteria.

Then it will try to find paths, by following (transitively or not) the root objects to the - if present - matched sink nodes.

The path finding is being performed by the MySQLObjectGraphRecorder and it queries that graph table where the graph is stored in MySQL.

Notably, to enable nesting or combination of the in graph operator with operators evaluated in another data store, such as Elastic, the search compiler assumes responsibility for transforming the in graph operator from a graph query executed on the graph's data store to a search query executed on the search data store.


Seach API
Neo4jInGraphOperatorEvaluator
Elastic
CRUD API
Neo4jObjectGraphRecorder
Graph queries
Neo4j
Graph management

By evaluating graph queries in Neo4j we were able to:

- Gain access to more powerful graph queries. Given that the current graph api leverages the Neo4j querying engine, it can express complex graph queries with ease. For example, the previous query API could not find paths starting from specific root nodes to sink nodes. Currently, it possesses the capability to traverse any field (for instance, by offering support for the ”*” operator to traverse any reference field), while facilitating more straightforward transitive path discovery. Certainly, besides the graph API accessible to end users, data analysts have access to the native Neo4j environment, enabling them to employ Cypher queries according to their preferences.

- The existing implementation of graph operator evaluators has undergone significant simplification. Formerly, it necessitated extensive and highly complex coding for the exploration of graph nodes. Presently, these operations are managed gracefully and have been streamlined, making code much more scalable and robust.

- The evaluation of complex queries is significantly more efficient and performs optimally. This outcome is logical, as it aligns with the inherent capabilities of a graph database, surpassing the performance levels achievable by a relational database.

Steps forward

- The existing search API facilitates authorized searches that dynamically modify results in accordance with user authorization realm capabilities. These authorized searches are presently implemented on top of Elasticsearch, involving intricate and convoluted implementations to accommodate this functionality. Consequently, the introduction of a graph API offers an opportunity to redesign the current search API in a more elegant manner, augmenting its capabilities. With the graph API, the identification of authorized relationships between users and data becomes more straightforward and natural, aligning with a more intuitive approach.

- Data and subject to continuous change. To ensure data remains current, it is imperative to update it promptly whenever changes occur at the source of truth. Managing data currency can be a complex task, especially when dealing with multiple data stores, as data often possess intricate relationships that require simultaneous updates. Having efficient means to retrieve the dependency or reference graph of records can greatly expedite the process of reindexing and updating data related to a particular object within its corresponding data stores.

- The availability of robust graph queries will also contribute significantly to the extended utilization of the forthcoming actioning knowledge graph, as discussed in the subsequent section. Given the graph-based nature of knowledge graphs, the presence of comprehensive graph exploration tools is deemed imperative.

Building a decisioning knowledge graph

Graph Queries

Graph Data Science

Graph Traversal

Graph Visualization

Graph data science can help us distill useful information by bringing together graph analytics, statistics, and AI and ML techniques to improve their predictive and prescriptive model. Graph queries applied together with graph science tools can provide powerful analytics tools and set a solid foundation for machine learning workflows.

```
CALL gds.graph.project(
  "units_gds",
  ["_Unit"],
  ["REF_TO"]
)
YIELD
  graphName AS graph, nodeProjection, nodeCount AS nodes,
     relationshipProjection, relationshipCount AS rels
```

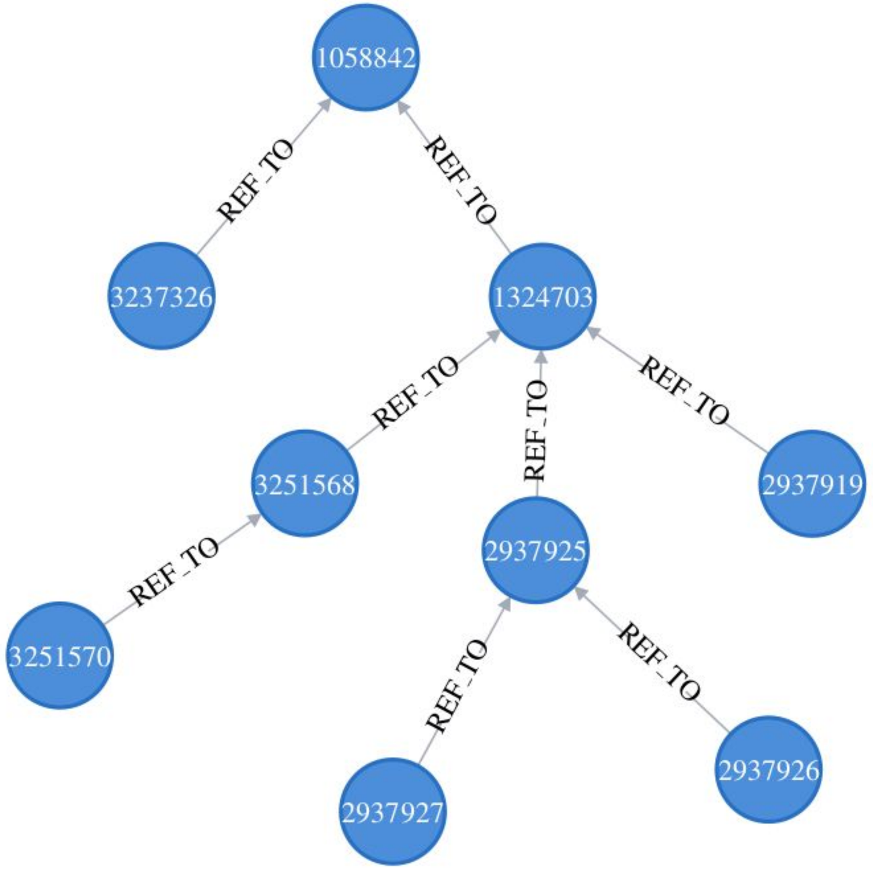
Graph Projection

Since our graph database of choice is Neo4j, we proceeded in the usage of Neo4j's Graph Data Science (GDS) library that provides efficiently implemented, parallel versions of common graph algorithms, exposed as Cypher procedures. Neo4j's GDS library requires a graph projection step that materializes a graph's subgraph and then offers:

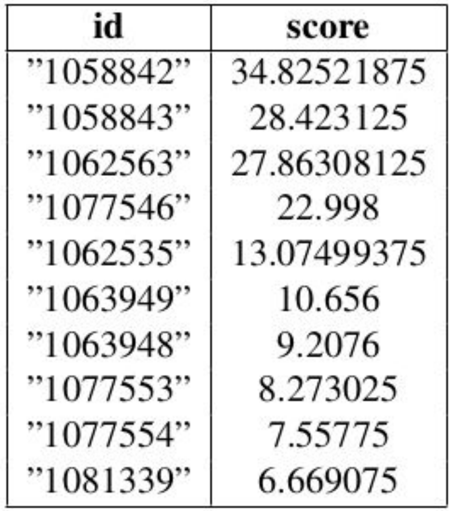
- Centrality, Community detection algorithms
- Graph traversal, path finding algorithms
- ML pipeless for graph predictions or classifications.

```
CALL gds.pageRank.stream("units_gds")
YIELD nodeId, score
RETURN gds.util.asNode(nodeId)._id AS id, score
ORDER BY score DESC
```

Page Rank Algorithm



Materialized Graph



id	score
"1058842"	34.82521875
"1058843"	28.423125
"1062563"	27.86308125
"1077546"	22.998
"1062535"	13.07499375
"1063949"	10.656
"1063948"	9.2076
"1077553"	8.273025
"1077554"	7.55775
"1081339"	6.669075

PageRank results

Steps forward

- Support and experiment with more graph science algorithms, integrating them in Niovity’s datalake.

- Leverage those statistics and graph algorithms through our search API. Since graph projection and graph algorithms can be automated and be driven from the prototype definitions, we could introduce new AI operator or statistics that could allow end users to perform graph science analytics out of the box, without having to delve into the underlying graph data store’s internals.

- Support ML Workflows for graphs in Datalake. Neo4j GDS supports ML pipelines, including node classification and link prediction pipelines. Also ML workflows can be improved from the usage of graph algorithms and graph queries, allowing for continuous updates to knowledge graph that predicts missing data and relationships.

Building a decisioning knowledge graph

Graph Queries

Graph Data Science

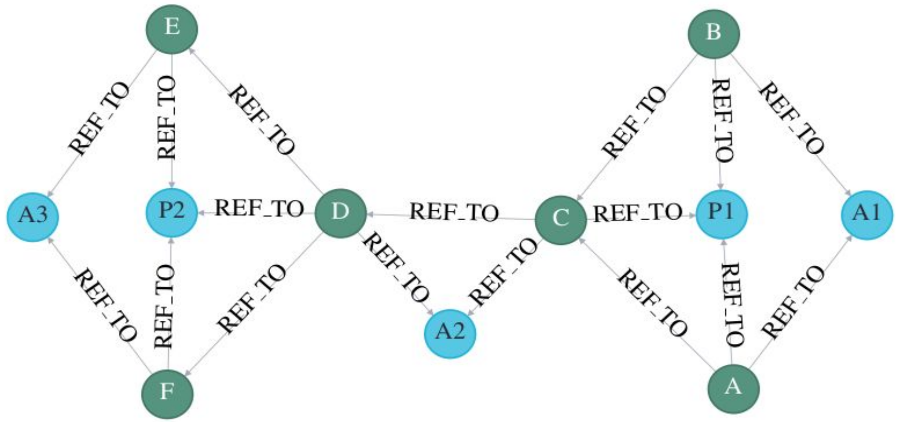
Graph Traversal

Graph Visualization

```
def "Graph Traversal Specification"() {
    given:
    List nodesTraversedWithBfs = []
    List nodesTraversedWithDijkstra = []

    BiConsumer<String, String> bfsNodesTraversalConsumer = {
        String storeName, String id ->
            nodesTraversedWithBfs
                .add(storeName + ":" + id)
    }

    BiConsumer<String, String>
       dijkstraNodesTraversalConsumer = {
        String storeName, String id ->
            nodesTraversedWithDijkstra
                .add(storeName + ":" + id)
    }

    when:
    graphTraversalHelper.bfs("C", bfsNodesTraversalConsumer)
    graphTraversalHelper.dijkstra("A", "F",
       dijkstraNodesTraversalConsumer)

    then:
    nodesTraversedWithBfs == ["data:C", "data:D", "data:F",
       "data:A3", "data:E", "data:P2", "data:P1", "data:A2"]
    nodesTraversedWithDijkstra == ["data:A", ’data:C’,
       "data:D", "data:F"]
}
```

Graph traversal or path finding algorithms are crucial for organizations in order to be able to implement custom business logic to specific graph paths.

By implementing simple consumer pattern in application level we were able to programmatically apply custom actions to the traversed nodes, ranging from domain specific business logic to esoteric use cases (for example reindex a node and its transitive neighbors after a record update).



Building a decisioning knowledge graph

Graph Queries

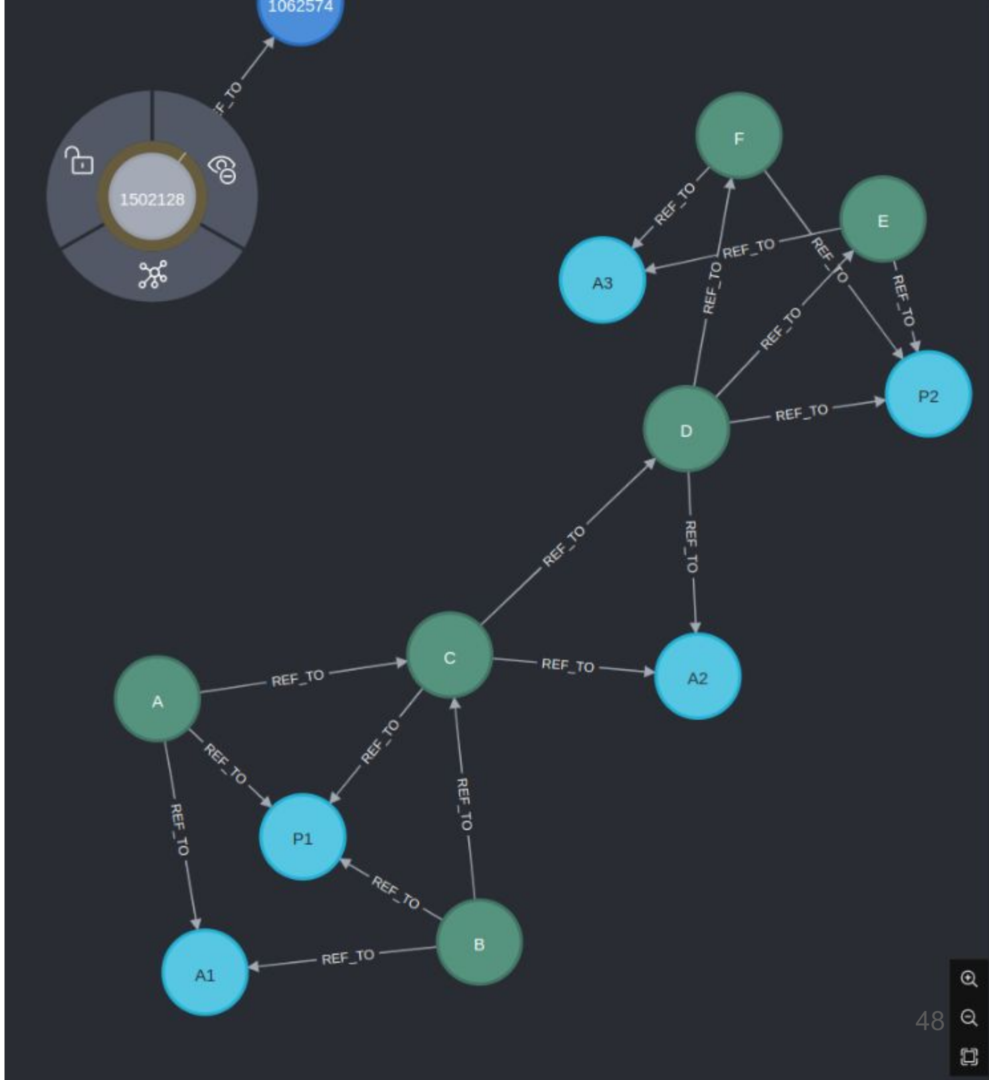
Graph Data Science

Graph Traversal

Graph Visualization

Exploring graphs can be a bogging task, as they tend to be noisy and chaotic. In order to effectively navigate within the graph landscape, we should be equipped with means to intuitively visualize the graph data, especially its relationships, so that we can better explore the connections and infer meaning. This includes interactive tools that can naturally allows users to walk through relationships and filter views dynamically with low to no-code actions.

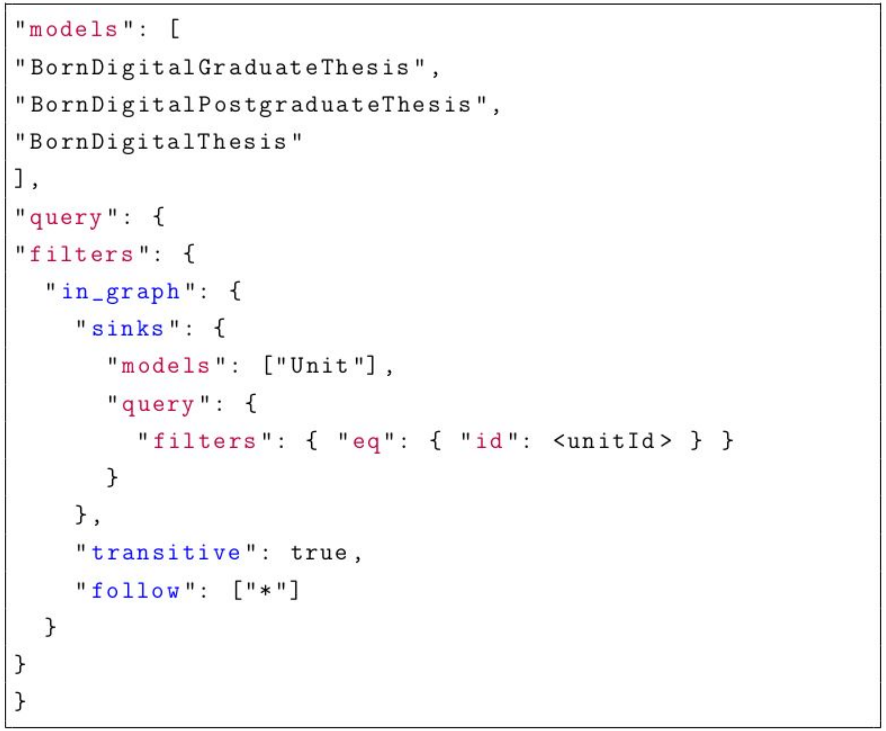
Since graph data are stored already in Neo4j, these visualization tools are provided out of the box, allowing for graph exploration to data, a notable enhancement in regard to the data previously stored in tabular format.



While Neo4j's graph visualization libraries hugely benefited data exploration within datalake, we further explored alternative visualization methods regarding custom use cases.

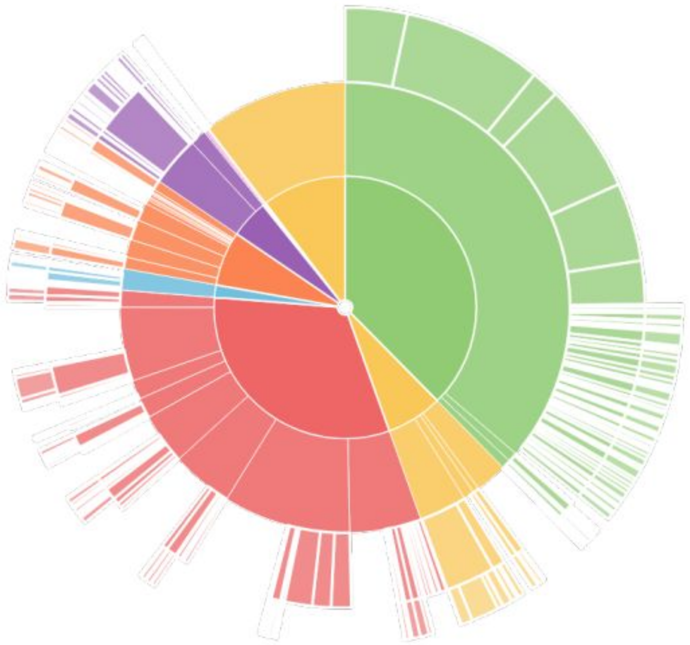
Taking for example Pergamos case, gray literature items (under-graduate, graduate thesis) are deposited to specific academic units. Having a diagram of unit's hierarchies, visualizing the number of records on each unit in an hierarchical manner, was a complex task that wasn't feasible without graph queries.

By employing the graph queries of the the aforementioned subsection, we were able to calculate transitively the number of records for each unit, allowing us for nested visualizations.



```
"models": [
"BornDigitalGraduateThesis",
"BornDigitalPostgraduateThesis",
"BornDigitalThesis"
],
"query": {
"filters": {
  "in_graph": {
    "sinks": {
      "models": ["Unit"],
      "query": {
        "filters": { "eq": { "id": <unitId> } }
      }
    },
    "transitive": true,
    "follow": ["*"]
  }
}
}
```

The above graph query searches for gray literature items that belong transitively to a specific unit, by following any field. That means that, unlike the previous graph implementation, for each unit we can efficiently count the number of items that transitively belong to it. By recursively running this query for all Pergamos units, we were able to generate the following graph visualization, generated with the help of Apache Echarts, which depicts the number of gray literature items each unit has, in a hierarchical manner.



Steps forward

- Integration with other data visualization tools such as Neo4j bloom, or other visualization libraries.

- Experiment with other types of visualizations (for e.g trees).

Revisiting the decisioning knowledge graph implementation and further steps

We have reached the final phase in the development of the decisioning knowledge graph. By transitioning our graph storage to Neo4j's graph database, we have been able to harness the engine's capabilities in graph querying, graph data science, and visualization. These components constitute a decisioning layer overlaying our data fabric, enabling the inference of improved decision-making and facilitating more accessible data exploration, while also layout in foundational groundwork for ML & AI pipelines.

Although the current state of the decisioning knowledge is in initial steps, it forms a firm base for future expansion, as we will acknowledge on the following section. It exhibits significant potential for enrichment with related semantics and capabilities.

For example we further experiment with the decisioning knowledge graph implementation to:

- Use the graph structure as ML predictor, that could further commoditize its tools for widespread business use.

- Utilize the current query federation architecture to query graph embeddings to the current or another data store.

Building an actioning knowledge graph

Model Exploration

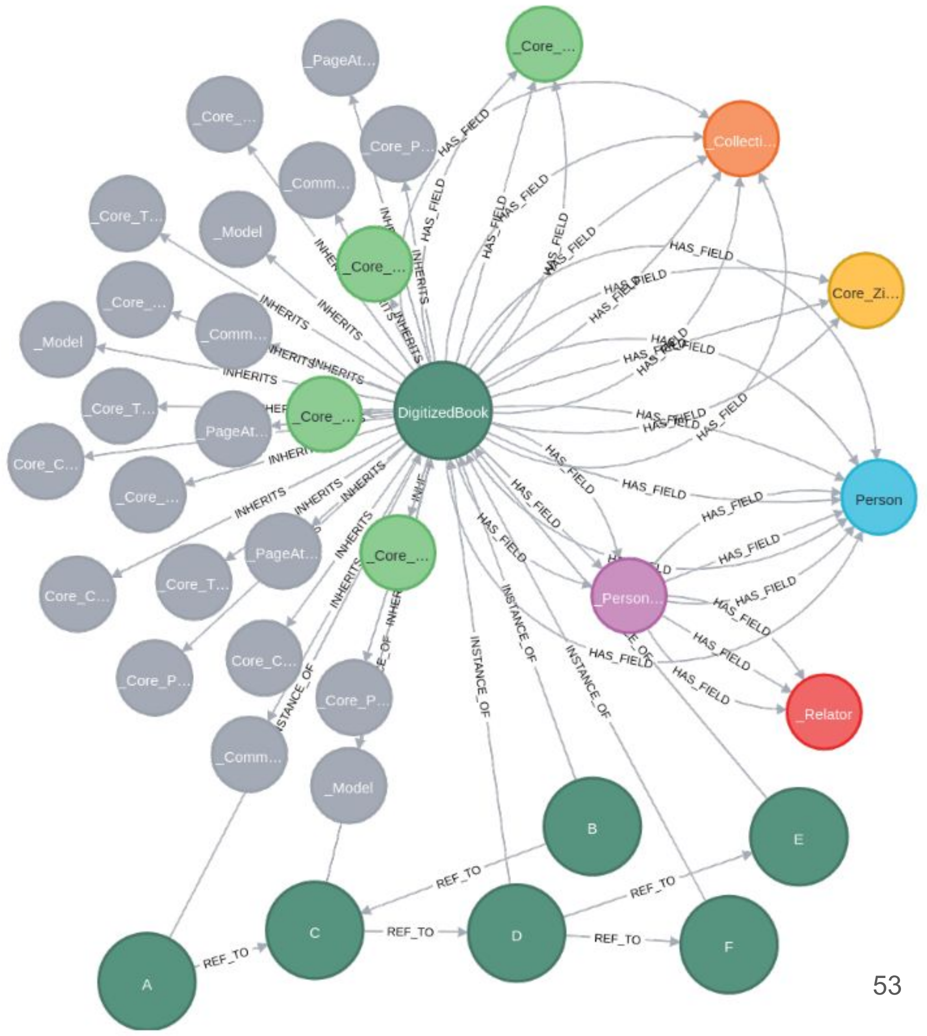
Data Provenance and Lineage

Data Integration

Operations

Prototypes are the driving factor within Niovity's datalake. They establish the domain model of knowledge within an organization by defining the conceptual model and the associated ontologies for the different entities that compose it. Prototypes drive all the aspects of Niovity's data lake, starting from record instantiating and search analytics to code generation. A domain can consist of hundreds of prototype definitions that are constantly being changed and updated, while inheriting or depending to prototype libraries that are kept in different prototype libraries.

That makes the management and data model exploration a complex task for data employees and end-users, undermining its capabilities. For these reasons, we started the Actioning graph creation by augmenting the knowledge graph with metadata and semantics regarding the prototype definitions. From inheritance to cross-references and field definitions, we ported the associated semantics and metadata about the prototype definitions to Neo4j, integrating it with the existing graph.



Each prototype is encoded to a node, with label its id and node properties the field definitions. For the ref fields we take the allowed ref types and model it to a graph node as well, while the ref field definition (regarding the field definition id or other metadata) is encoded to the graph edges. After prototype definition insertion to Neo4j, we continue by enriching the graph with the prototype inheritance, so that it can capture semantics regarding the taxonomies that abide the model definitions. Lastly, we connect the prototype definitions with their instances, which means the graph objects that have already been ported, as we saw in the previous section.

The prototype porting to Neo4j was completely automated in a model-driven manner with the aid of the Visitor design pattern.

Neo4j node properties

```
id: 16492
binding: TEXT
creationYear: NUMBER
description: BIGTEXT
dimensions: TEXT
isbn: TEXT
language: TEXT
pagesTotal: NUMBER
publish: TEXT
thematicCategories: TEXT
createdAt: NUMBER
modifiedAt: NUMBER
title: TEXT
```

Neo4j relationship properties

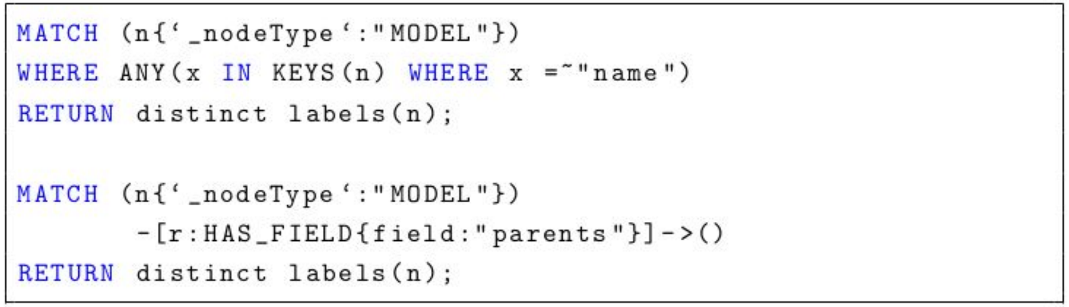
```
id: 9243
field: authors
```

By incorporating the prototype definitions in our existing decisioning knowledge graph, we were able to leverage its captured semantics and harness its capabilities for the following usages:

• Gain access to the visualization tools already established from the decisioning knowledge graph implementation. Those visualization tools can now also be utilized for the prototype exploration and model definitions, helping organizations to deeply understanding their domain model.

• By representing the prototype definitions in the form of graph, we are now able to execute graph queries to the model definitions itself along with the data instances. That packs invaluable potential that can make the model exploration even more specific. Example queries that can be asked directly to the actioning knowledge graph can be the following:

Find prototypes that contain a specific field (non-ref and ref fields)



```
MATCH (n{‘_nodeType‘:"MODEL"})
WHERE ANY(x IN KEYS(n) WHERE x =~"name")
RETURN distinct labels(n);

MATCH (n{‘_nodeType‘:"MODEL"})
       -[r:HAS_FIELD{field:"parents"}]->()
RETURN distinct labels(n);
```

Count relationships for type

```
MATCH (n:DigitizedBook)-[r:HAS_FIELD]->()
RETURN COUNT(r)
```

Preview prototypes field definitions

```
MATCH (n{‘_nodeType‘:"MODEL"})
UNWIND keys(n) AS key
RETURN key,n[key]
```

Count the different field types

```
MATCH (n{‘_nodeType‘:"MODEL"})
UNWIND keys(n) AS key
RETURN DISTINCT n[key] as field_type, count(n[key]) as
   frequency
ORDER BY frequency DESC
```

Show instances of a type

```
MATCH (n:‘_BornDigitalPostgraduateThesis‘)
        -[r:INSTANCE_OF]
        ->(c:‘_BornDigitalPostgraduateThesis‘{‘_nodeType‘:"
           MODEL"})
RETURN count(n)
```

Steps forward

• Encoding prototype definitions in Neo4j as graph, not only enables us with graph queries and graph visualization tools. As we saw on the decisions graphs, there are powerful graph data science algorithms that can provide invaluable insights regarding the organization's domain model. Centrality algorithms, such as page rank, can rank prototypes according to the ”importance” of a prototype when it comes for e.g. to its size. Community detection algorithms can group prototypes to cluster or communities, providing clusters of similar prototypes. Path finding algorithms or graph traversal can help find paths or circles in prototypes definitions. Having access to such powerful tools in regard to modeling provides indispensable insights that can lead to better actions.

• Prototype definitions along the knowledge graph can solidify future ML pipelines. Structural information and data structures don't change as fast as records do. The means that predictive models can have longer shelf life with less training required. The semantics regarding the taxonomies of inheritance provide contextual information for investigations and counterfactual analysis by domain experts and can be fed to training algorithms to produce better results in regard to records classification or prediction but also to the model behavior itself (for e.g. we could find similar prototypes, we could classify an ontology to an existing one, we could generate from AI prototypes based on ontologies, or even link prediction among its definitions).

Building an actioning knowledge graph

Model Exploration

Data Provenance and Lineage

Data Integration

Operations

Data are dynamic, contextual and heterogeneous. The management of its lineage and its whole data genealogy, from the data origination to the data exhaustion, are key elements to data provenance. For a data lake to effectively and responsibly manage the data across its various domains and industries, it is critical to provide robust data provenance mechanisms.

Data provenance is a requisite for:

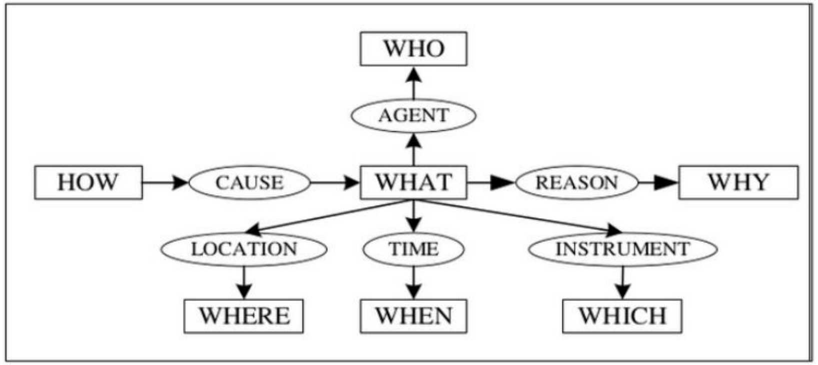
- Data quality assurance, auditing and compliance
- Auditing or troubleshooting
- Data security
- Data reproducibility
- Data collaboration and sharing
- Data trust and transparency

Data provenance, though, is an overloaded term and can mean many things. The **lack of consensus** on the semantics of the provenance has as a consequence to be incomplete and futile. In order to reach the required level of consensus, along with the potential benefits that data provenance offers, it's important to **define the semantics** of provenance in a more formal manner, for example in the **form of ontologies**.

The custom ontology that we picked was taken from **Ram & Liu (2009)** paper, A New Perspective on Semantics of Data Provenance. In their paper, the authors introduced a custom ontology, that captures the essential semantics regarding the data provenance, which they call the W7 model. The W7 model is a conceptual, general and extensible ontological model that represents data provenance as a combination of seven interconnected questions, that can ultimately answer any question regarding its data chain. It is adopted along with **Bunge (1977) ontology**, in order to define the above components and identify the relationships between them.

- **What**: Denotes an event that affected data during its lifetime.
- **When**: Refers to the time at which the event occurred.
- **Where**: Is the location of the event.
- **How**: Is the action leading up to the event.
- **Who**: Is the agents involved in the event.
- **Which**: Are the programs or instruments used in the event.
- **Why**: The reasons for the event.

W7 model as conceptual graph



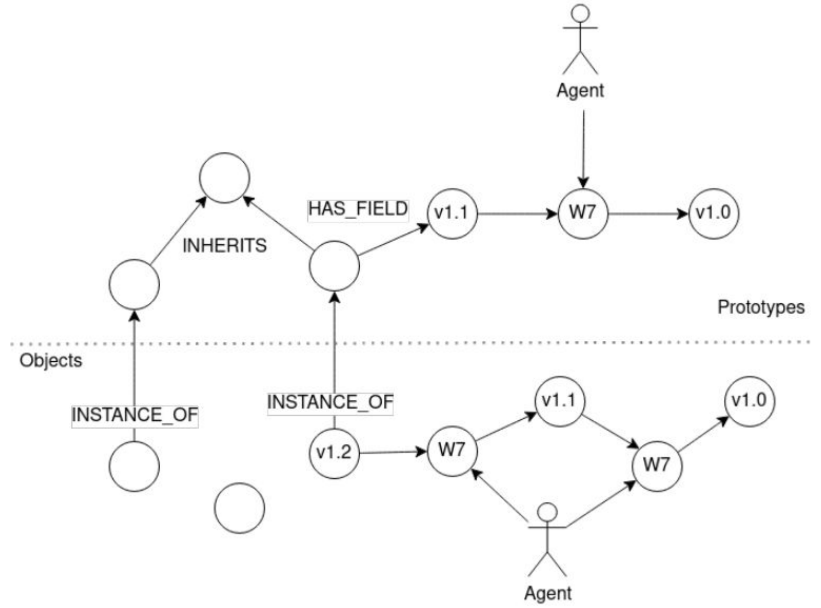
The idea

Based on the proposed ontology, we could adopt the same semantic model in our actioning knowledge graph to keep track of the metadata regarding the data lineage and data provenance in Niovity's data lake. Niovity's data lake captures semantics and offers tools that:

- Version and keep metadata about prototype changes
- For the prototype instances there under development a versioning control system, with the aim to offer traversal queries and branching along the timeline tracking of datalake's records.

By augmenting the current actioning knowledge graph with the W7 model, we could introduce the W7 ontology as a type of graph node that would contain the answers to W7's imposed questions.

That encoding process keeps track of the data provenance metadata and semantics in a disciplined and well-managed manner, allowing offers means to better understand **objects lineage** and **schema evolution**.



Steps forward

- Further experiment and enrich the current ontology with related semantics, for e.g. linking it with the data integration ontology that is proposed in the following subsection in order to capture the data genealogy starting from their data origin.

- Gain access to the graph database's graph queries and graph visualization.

- Further utilize the graph data science libraries for multiple reasons, including applying centrality algorithms to find records with long data chains, ml pipelines that could predict the next change on a record and generally gain insights that would be invaluable Niovity's employees or data scientists for data cleansing, debugging and various other scopes.

Building an actioning knowledge graph

Model Exploration

Data Provenance and Lineage

Data Integration

Operations

Data integration is can resolve to complicated task, as data heterogeneity jeopardizes big data integration. Different ELT or ETL pipelines can become quite overwhelming to manage, as multiple transformation phases from different data sources are performed along the data movement.

This can:

- Tank data quality
- Produce pipeline debt
- Cause trouble to troubleshooting and debugging
- Undermine the rest of the data management pillars within a data lake, such as data provenance

Actioning knowledge graphs can provide a sophisticated index and integration points, so that data across silos can be curated. By modeling the data integration components with various semantics and ontologies, we can form a disciplined framework to manage and understand the underlying data integration.

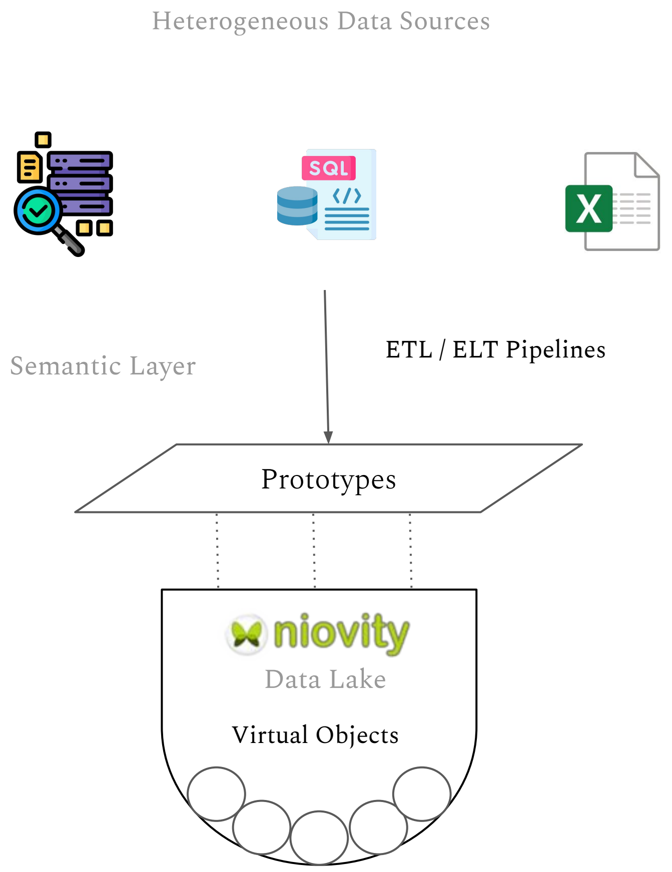
Choosing an actioning knowledge graph to store and handle the data integration metadata is a interesting approach, because against the dynamic nature of data fabric integration, the graph model remains flexible over the lifetime of the data domains.

Data integration in Niovity's data lake is usually performed through ETL processes that integrate data from heterogeneous data sources to single unified ontology, called prototype. Niovity's ETL processes offer various data integration tools, including **data importing tools** (excel and json), **schema discovery utilities** (that analyze the data source and can produce prototype definitions) and of course manual migration scripts.

The idea

By incorporating an ontology upon our actioning knowledge graph we can gain access to a general-purpose, organization-wide data access interface that offers a connected view of the integrated domains by combining data stored in a local graph with data retrieved on demand from third-party systems. This could help in:

- Data integration documentation
- Assist in the pipeline debugging.
- Empower the upcoming data lineage scenarios with even more data and context regarding the data genealogy.



An example data lake ontology that could capture the essential semantics of the data integration modeling could the following:

- **Data source:** From organization's data silos, to remote apis, data source refers to the origin or location from which data is obtained. The metadata regarding the data source could be as simple as its name, its type, a simple description or other custom notes that explains the data source.

- **Data set:** Data sets are the structured data that we collect from data sources. Data sets are being integrated into the data lake.

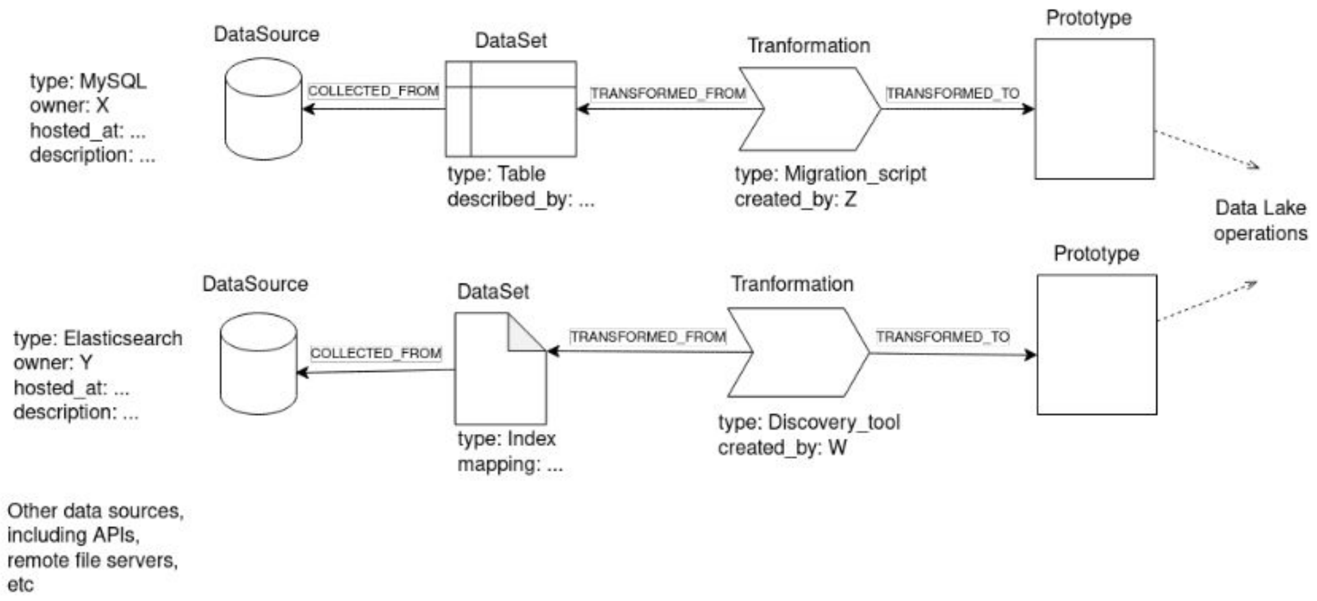
- **Transformation:** Data sets from different data sources are being cleaned, enriched and transformed to other formats, in order to then be integrated and unified in the data lake. It can also have a type, it can be connect with the agent that created it etc.

The above concepts could encoded as graph nodes in the Actioning knowledge graph, while we could introduce the following relationship types to be part of the proposed sample ontology:

- **collectedFrom:** This relationship type could relate the data set with the data source that was derived from.

- **transformedFrom / transformedTo:** The relationship connects the originated data set with transformed entity that was transformed to.

By incorporating the above ontology to our actioning knowledge graph allows the data integration pipelines can be traversed, we can conduct impact analysis queries for e.g. to analyse for example the disrupted migration channels when a service is unavailable. It also standardizes the integration processes by mapping the data assets, thus providing global visibility, making the data discoverable and accessible and thus promoting their consumption.



Steps forward

- By enriching the actioning knowledge graph with semantics regarding the data integration process, other important factors, such us data provenance, can also benefit. Data lineage starts from its data origin, so integrating the data provenance model that we presented on the previous section with the data integration semantics can bulletproof the data provenance within Niovity's datalake.

- Gain access to the graph database's graph queries and graph visualization.

- There has been ongoing academic research regarding the data integration within data lakes and data warehouses. In their paper "Ontology-Driven Conceptual Design of ETL Processes Using Graph Transformations", Skoutas et al. (2009) introduced a method for devising flows of ETL operations by means of graph transformations. In their paper "Modelling Data Pipelines", Raj et al. (2020) suggests an overview of how to design a conceptual model of data pipelines, used for automation of monitoring, fault detection, mitigation at different steps of a data pipeline. This provides a significant opportunity for in-depth exploration aimed at enriching the initial sample ontology with domain-specific knowledge.

Revisiting the decisioning knowledge graph implementation and further steps

We have reached the final phase in the development of the decisioning knowledge graph. By transitioning our graph storage to Neo4j's graph database, we have been able to harness the engine's capabilities in graph querying, graph data science, and visualization. These components constitute a decisioning layer overlaying our data fabric, enabling the inference of improved decision-making and facilitating more accessible data exploration, while also layout in foundational groundwork for ML & AI pipelines.

Although the current state of the decisioning knowledge is in initial steps, it forms a firm base for future expansion, as we will acknowledge on the following section. It exhibits significant potential for enrichment with related semantics and capabilities.

For example we further experiment with the decisioning knowledge graph implementation to:

- Use the graph structure as ML predictor, that could further commoditize its tools for widespread business use.

- Utilize the current query federation architecture to query graph embeddings to the current or another data store.

Building an actioning knowledge graph

Model Exploration

Data Provenance and Lineage

Data Integration

Operations

Actioning knowledge graphs are not only limited to those specific areas. Metadata revolve around every simple bit of data and Actioning knowledge graphs can be utilized to capture and leverage them and increase the operational efficiency among organizations.

Software artifacts are build and produced from various systems, are hosted to specific servers and delivered to their clients accordingly. The successful software delivery, release and deployment entails meticulous coordination and comprehensive documentation, in order to satisfy the quality assurance requirements for its clientele.

Having a robust framework to document and manage those processes can be of a great asset for release managers, dev-ops engineers and generally, for all stakeholders involved in the software implementation process, including release managers, DevOps engineers, quality assurance professionals, and others.

Niovity's software artifacts can roughly be classified to the following categories:

• The prototype libraries, which contain the model definitions, are released and versioned
following its schema evolution.
• Middleware and Backoffice applications, which form the backbone of the backend's architecture, are released, deployed and delivered to clients. The middleware and the backoffice apps need to be aligned with the latest version of the type libraries.
• Backend's consumers, varying from front-end applications to other third-party systems (mobile / desktop apps).

To ensure that all artifacts meet the compatibility requirements imposed by the prototype libraries and are easily discoverable and effectively managed, we devised a custom ontology that could drive the creation of yet another actioning graph.

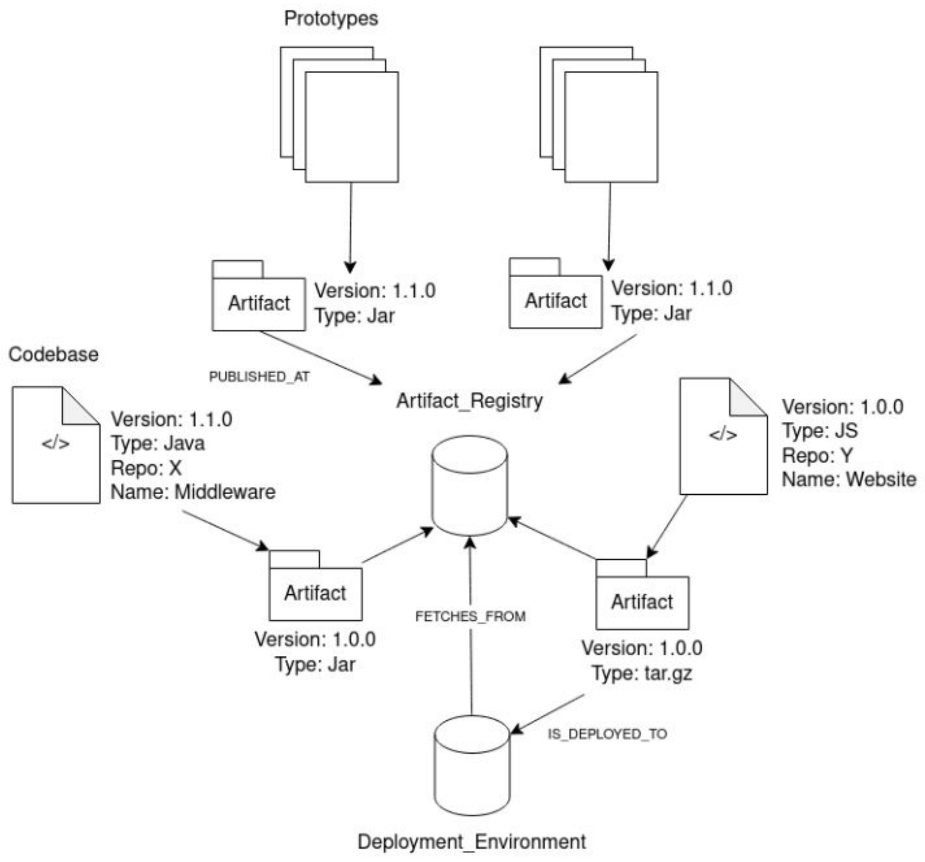
- **Codebase:** Models the various code bases that can produces artifacts.
- **Artifact:** Models Niovity's software artifacts.
- **Artifact Registry:** Software built artifacts are being stored to dedicated repository managers, so that client consumers can fetch from.
- **Deployment Environment:** From dedicated servers to cloud computing services, it represents the environment where the artifacts are deployed to.
- **Client:** The clients that software artifacts are being delivered to.

By projecting the metadata revolving the artifacts management and DevOps operations to a knowledge graph, we could:

- Visualize the software delivery pipelines
- Enhance the versioning systems, provide means for better documentation and risk management.

Steps forward

- Facilitate access to graph queries and leverage GDS library tools. These tools have the potential to model the software delivery and continuous integration pipelines, while providing optimization suggestions, aid in troubleshooting, and ensure the overall quality of the artifacts delivery procedures.



Revisiting the actioning knowledge graph implementation and further steps

Within this section, we have examined the potential avenues through which an actioning knowledge graph can deliver organizational benefits, particularly within the context of a data lake. From model exploration to data provenance and data integration, actioning knowledge graphs possess the capability to enhance the various facets of data management processes.

Actioning graphs exhibit a high degree of adaptability, allowing them to align seamlessly with the specific requirements of the organizations that employ them. By establishing custom semantics and ontologies, data scientists can seamlessly integrate metadata and diverse models into a unified graph structure, thereby facilitating a singular point of access. This integration serves to enhance organizational efficiency and efficacy. The mere imposition of metadata and semantics onto a graph equips users with access to a suite of data visualization processes, graph queries, and graph data science libraries, that can contribute to further operational efficiency and effectiveness.

The proposed actioning knowledge graph is a necessary step toward unleashing value for Niovity's data lake. There numerous domains of interest that could be benefited for the presence of a knowledge graph.

Steps forward

• Prototype definitions are scoped from specific organization domains. For Niovity to scale across multiple domains and maintain robust data management procedures, it shall afford single views of its customer's and its respective domains. By modeling projects, clients and their respective prototype libraries in a single graph, we could gain access to the single view of the clientele landscape.

• By providing context about data's surrounding information, circumstances and environment can lead to the positive influence of data and interactions. This includes the advancement of AI pipelines within the field of contextual AI, by empowering its adaptive learning procedures, while facilitating its explainability.

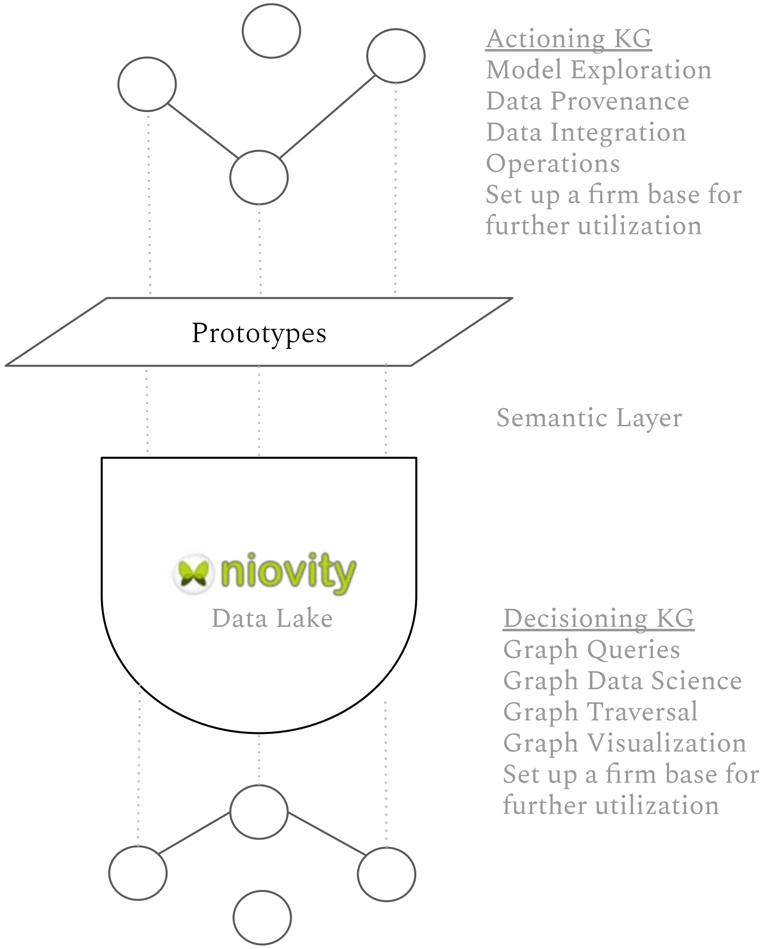
• Explore and integrate additional ontologies in either a horizontal or vertical fashion to enhance the processes of data governance and data stewardship. Each of the aforementioned subsections constitutes a prospective area warranting additional research and thorough investigation.

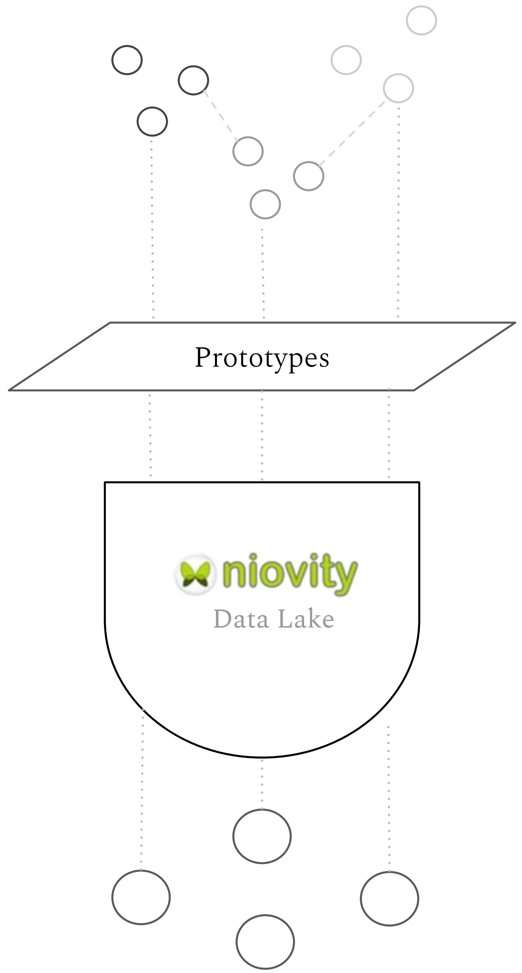
• In the following section, we examine the possibility of consolidating multiple knowledge graphs into a singular entity, that can assist owing to their interconnectedness within the underlying semantic structure.

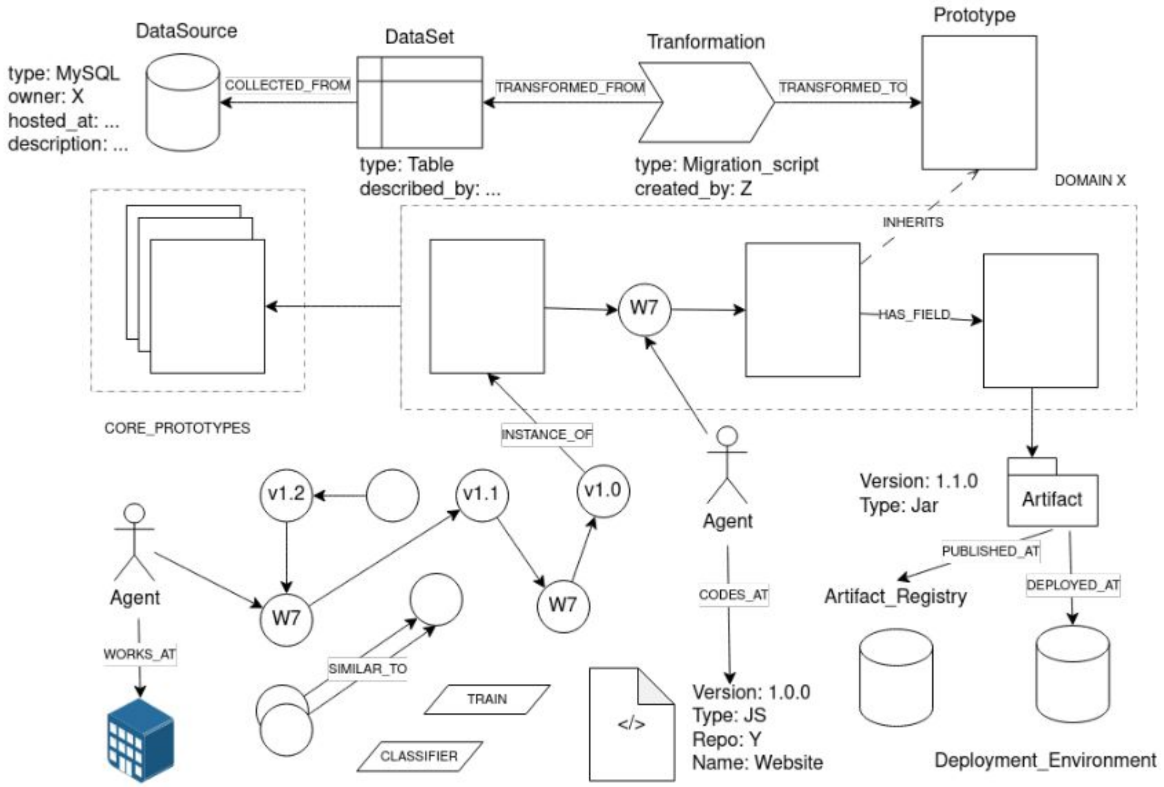
A unified knowledge graph

Having reached the end of this presentation, we have explored the practical and theoretical applications of a knowledge graph in the context of semantic data lake. By employing the advanced graph analytics and powerful graph engine of Neo4j, we were able empower Niovity's existing knowledge graph with enhanced capabilities.

The primary objective and ultimate aim of this thesis entail the integration of the actioning and decisioning knowledge graphs into a unified and consolidated view, with the overarching purpose of enhancing their interoperability. Through the representation of diverse facets of Niovity's data lake ecosystem within a single knowledge graph framework that facilitates robust analytics, it is conceivable that we may acquire deeper insights and unveil aspects pertaining to the critical dimensions of the organization. This approach also establishes a robust contextual environment that fosters transparency and serves as a foundation for operational efficiency, thereby aiding in the scalability and effective management of the data lake.




Prototypes
niovity
Data Lake


DataSource
type: MySQL
owner: X
hosted_at: ...
description: ...
COLLECTED_FROM
DataSet
type: Table
described_by: ...
TRANSFORMED_FROM
Tranformation
type: Migration_script
created_by: Z
TRANSFORMED_TO
Prototype
DOMAIN X
INHERITS
CORE_PROTOTYPES
W7
HAS_FIELD
INSTANCE_OF
Agent
v1.2
v1.1
v1.0
W7
W7
Agent
WORKS_AT
SIMILAR_TO
TRAIN
CLASSIFIER
</>
CODES_AT
Version: 1.0.0
Type: JS
Repo: Y
Name: Website
Version: 1.1.0
Type: Jar
Artifact
PUBLISHED_AT
Artifact_Registry
DEPLOYED_AT
Deployment_Environment

Section 5

Conclusion

Upon the conclusion of this thesis, we have delineated the capacity to leverage the semantics captured within a semantic data lake through the utilization of a knowledge graph and graph technologies. This study has centered on the specific use case of Niovity's semantic data lake, elucidating its potential to catalyze substantial improvements and transformations within it's data landscape.

Given the current semantic anode and drawing from the theoretical framework outlined in the book ”Knowledge Graphs: Data in Context for Responsive Businesses,” we have advanced the proposition of a unified knowledge graph. This comprehensive knowledge graph holds the potential to offer further insights into Niovity's ecosystem while concurrently fostering operational efficiency.

This thesis serves as an initial foray into the assessment of the value inherent in knowledge graphs and their exploitation through the utilization of graph technologies. Our presentation of various dimensions has been deliberately horizontal in structure, affording the flexibility for future expansion and deeper exploration. Knowledge graphs are dynamic and can grow and evolve organically, according to the organizational needs and objectives.

The responsibility for molding and advancing knowledge graphs in line with their desired outcomes rests with the respective organizations.

Steps forward

Given the experimental nature of this thesis, there exists substantial scope for further improvement and implementation. This includes:

• The integration of Neo4j's graph database has endowed us with a multitude of potential advantages, albeit not without associated costs. To ensure the efficient assimilation of Neo4j's graph engine, it is imperative that we conduct comprehensive performance assessments of the existing architecture, particularly in relation to the search API and graph creation processes.

• We may contemplate the adoption of Neo4j or an alternative graph database as our principal data store, thereby obviating the necessity for a relational database.

• Conduct experiments involving alternative ontologies or metadata management approaches that may yield advantages in terms of data stewardship and governance within the data lake.

Bibliography

ANSARI, JASIM WAHEED; NAILA KARIM; WAHEED ANSARI; OYA DENIZ BEYAN; und MICHAEL COCHEZ. 2018.
Semantic profiling in data lake.
URL https://api.semanticscholar.org/CorpusID:4875432.

BAGOZI, ADA; DEVIS BIANCHINI; VALERIA DE ANTONELLIS; MASSIMILIANO GARDA; und MICHELE MELCHIORI. 2019.
Personalised exploration graphs on semantic data lakes. On the move to meaningful internet systems:
Otm 2019 conferences, hrsg. von Herv´e Panetto, Christophe Debruyne,
Martin Hepp, Dave Lewis, Claudio Agostino Ardagna, und Robert Meersman, 22–39. Cham: Springer International Publishing.

BUNGE, MARIO. 1977.
Treatise on basic philosophy: Ontology i: the furniture of the world,
Ausg. 3. Springer Science & Business Media.

CHESSA, ALESSANDRO; GIANNI FENU; ENRICO MOTTA; DIEGO REFORGIATO RECUPERO; FRANCESCO OSBORNE; ANGELO SALATINO; und LUCA SECCHI.2022.
Enriching data lakes with knowledge graphs.
1st international workshop
on knowledge graph generation from text and the 1st international workshop on
modular knowledge, text2kg 2022 and mk 2022. URL https://oro.open.ac.uk/83013/.

DIAMANTINI, CLAUDIA; DOMENICO POTENA; und EMANUELE STORTI. 2022.
A semantic data lake model for analytic query-driven discovery.
The 23rd international conference on information integration and web intelligence, iiWAS2021, 183–186.
New York, NY, USA: Association for Computing Machinery. URL https://doi.org/10.1145/3487664.3487783.

DIBOWSKI, HENRIK, und STEFAN SCHMID. 2021.
Using knowledge graphs to manage a data lake.

JES ́US BARRASA, AMY E. HODLER, und JIM WEBBER. 2021.
Knowledge graphs data in context for responsive businesses.
United States of America.: O’Reilly Media.

MOHAMED NADJIB MAMI. 2021.
Strategies for a semantified uniform access to large and heterogeneous data sources.
Rheinische Friedrich-Wilhelms-Universit ̈at Bonn
Dissertation. URL https://hdl.handle.net/20.500.11811/8925.

POMP, ANDR ́E; ALEXANDER PAULUS; ANDREAS KIRMSE; VADIM KRAUS; und TOBIAS MEISEN. 2018.
Applying semantics to reduce the time to analytics within complex heterogeneous infrastructures.
Technologies 6. URL https://www.mdpi.com/2227-7080/6/3/86.

RAJ, AISWARYA; JAN BOSCH; HELENA HOLMSTR ̈OM OLSSON; und TIAN J. WANG. 2020.
Modelling data pipelines.
2020 46th euromicro conference on software engineering and advanced applications (seaa), 13–20.

RAM, SUDHA, und JUN LIU. 2009.
A new perspective on semantics of data provenance.
Proceedings of the first international conference on semantic web in provenance management
- volume 526, SWPM’09, 35–40. Aachen, DEU: CEUR-WS.org.

SKOUTAS, DIMITRIOS; ALKIS SIMITSIS; und TIMOS SELLIS. 2009.
Ontology-driven conceptual design of etl processes using graph transformations.
J. Data Semantics 13.120–146.

ZAHARIA, MATEI A.; ALI GHODSI; REYNOLD XIN; und MICHAEL ARMBRUST. 2021.
Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics.
Conference on innovative data systems research. URL https://api.semanticscholar.org/CorpusID:229576171.