Deliverable D6.2 – Script and Microservices Software - Intermediate

GRANT AGREEMENT NUMBER: 101070141

This project has received funding from the European Union's HE research and innovation programme under grant agreement No 101070141

Project acronym:	GLACIATION
Project full title:	Green Responsible Privacy Preserving Data Operations
Call identifier:	HORIZON-CL4-2021-DATA-01-01
Type of action:	RIA
Start date:	01/10/2022
End date:	30/09/2025
Grant agreement no:	101070141

D6.2 – Script and Microservices Software - Intermediate

Executive Summary:	This document reports on the work done in Work Package 6 "Edge-Core-Cloud Platform" which was carried out from month 8 to month 14 and is aligned to T6.2 "Script and Microservices Development"		
WP:	WP6		
Author(s):	Cem Tanrikut, Aidan OMahony, Guangyuan Piao, Peter Forgacs, Oleksandr Chepizhko, Flavio Scaccia		
Editor:	Cem Tanrikut		
Leading Partner:	HIRO-MICRODATACENTERS B.V.		
Participating Partners:	DELL, MEF, HIRO, LAKE, ENG		
Version:	1.0	**Status:**	Final
Deliverable Type:	Other	**Dissemination Level:**	PU - Public
Official Submission Date:	30/11/2023	**Actual Submission Date:**	30/11/2023

Disclaimer

The GLACIATION consortium consists of the following partners:

No.	Partner Organisation Name	Partner Organisation Short Name	Country
1	MINISTERO DELL'ECONOMIA E DELLE FINANZE	MEF	IT
2	EMC INFORMATION SYSTEMS INTERNATIONAL UNLIMITED COMPANY	EISI	IE
3	HIRO MICRODATACENTERS B.V.	HIRO	NL
4	GOTTFRIED WILHELM LEIBNIZ UNIVERSITAET HANNOVER	LUH	DE
5	THE LISBON COUNCIL FOR ECONOMIC COMPETITIVENESS ASBL	LC	BE
6	UNIVERSITA DEGLI STUDI DI MILANO	UNIMI	IT
7	UNIVERSITA DEGLI STUDI DI BERGAMO	UNIBG	IT
8	GEIE ERCIM	ERCIM	FR
9	EURECOM	EURECOM	FR
10	SAP SE	SAP SE	DE
11	UNIVERSITY COLLEGE CORK - NATIONAL UNIVERSITY OF IRELAND, CORK	UCC	IE
12	SOGEI-SOCIETA GENERALE D'INFORMATICA SPA	SOGEI	IT
13	LAKESIDE LABS GMBH	LAKE	AT
14	ENGINEERING - INGEGNERIA INFORMATICA SPA	ENG	IT
15	EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH	ETH	CH

Document Revision History

Version	Description	Contributions
0.1	Table of Content	HIRO
0.9	Ready for internal review	DELL, LAKE, MEF
1.0	Ready for submission	HIRO, MEF

Authors

Authors	Partner
Aidan O Mahony	DELL
Guangyuan Piao	DELL
Cem Tanrikut	HIRO
Peter Forgacs	LAKE
Oleksandr Chepizhko	LAKE
Flavio Scaccia	ENG

Reviewers

Name	Organisation
Aidan O Mahony	DELL
Peter Forgacs	LAKE
Oleksandr Chepizhko	LAKE
Guangyuan Piao	DELL
Alessio Chellini	MEF

Table of Contents

List of Figures

List of Tables

List of Terms and Abbreviations

Abbreviation	Description
ARIMA	AutoRegressive Integrated Moving Average
AWS	Amazon Web Services
BNN	Bayesian Neural Networks
DKG	Distributed Knowledge Graph
DSS	Data Storage Service
DTW	Dynamic Time Warping
HBNN	Hybrid Bayesian Neural Networks
KG	Knowledge Graph
MAPE	Mean Absolute Percentage Error
MSE	Mean Squared Error
OM	Ontology of unites of Measure
OWL	Web Ontology Language
PCA	Principal Component Analysis
QoS	Quality of Service
RDF	Resource Description Framework
SHACL	Shapes Constraint Language
SOM	Self-Organizing Maps
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
VRAE	Variational Recurrent Auto-Encoder

Executive Summary

In the realm of software development, GLACIATION addresses the critical need for streamlined and efficient approaches to script and microservices development. This innovative platform serves as a comprehensive solution specifically designed to empower developers in creating agile, scalable, and resilient software applications, aligning precisely with the objectives outlined in our deliverable.

GLACIATION, as a privacy-preserving data analytics platform, strategically combines robust scripting capabilities with a microservices-first approach. This integration empowers development teams to construct resilient, scalable, and agile software solutions in the continually evolving digital landscape, thereby serving as a transformative force to navigate the challenges inherent in our deliverable's objectives.

GLACIATION's Core Mission:

At the core of GLACIATION's mission is the establishment of a resilient, scalable, and agile framework tailored for modern computing. Scripts and microservices play pivotal roles as linchpins in achieving this mission. Through process automation, power consumption optimization, and facilitation of resource scaling, they fortify the foundation of the platform, aligning with the objectives outlined in our deliverable.

Key Functionalities of Scripts and Microservices:

In this deliverable we describe the intermediate software release of the script and microservices developed to enable the Novel Metadata Fabric, codenamed “IceStream”, which serves as the backbone of the GLACIATION platform. This Novel Metadata Fabric essentially is made up of a Distributed Knowledge Graph (DKG), and required microservices such as the prediction service, meta-data service, data storage service and others.

1 Introduction

In the ever-evolving landscape of software engineering, two paradigms have risen to prominence, revolutionizing the way we conceive, design, and deploy applications: Script and Microservices Development. In a world where speed, flexibility, and scalability are paramount, these approaches stand as beacons of innovation, providing developers with the tools to navigate the complexities of modern software architecture.

We underscore the pivotal role played by scripts and microservices as the backbone of IceStream, a novel metadata fabric to be created in the project. They are not mere components; they are the architects of efficiency, scalability, and agility within our Novel Metadata Fabric. As IceStream evolves, scripts and microservices stand as the catalysts propelling its capabilities, ensuring that our users experience not just a metadata fabric, but a dynamic, responsive, and finely-tuned ecosystem driven by innovation and precision.

At its core, IceStream is not just a repository of metadata; it is an intricate web woven by the seamless integration of scripts and microservices. These dynamic components are purposefully crafted to be the driving force, the powerhouse that breathes life into IceStream.

1.1 Script Development: The Art of Simplicity and Efficiency

Script development represents a paradigm shift in the traditional software development process. Rooted in simplicity, scripts are lightweight, highly readable, and execute sequentially, making them ideal for automating tasks, performing rapid prototyping, and enhancing the efficiency of development workflows. Whether in the realm of web development, system administration, or data analysis, scripts empower developers to wield the power of automation, reducing manual intervention and accelerating the pace of innovation.

At its core, script development embodies the philosophy of "code as glue," seamlessly connecting disparate components and systems. The ability to swiftly translate ideas into functional code fosters a culture of experimentation and iteration. As developers embrace the elegance of scripting languages, they discover a dynamic toolbox that transcends traditional programming boundaries, offering a pragmatic solution for solving complex problems with minimal overhead.

1.2 Microservices Development: Architecting for Scalability and Resilience

In tandem with the rise of script development, Microservices architecture has emerged as a transformative force in application design. Departing from monolithic structures, Microservices break down applications into modular, independent services, each responsible for a specific business capability. This architectural paradigm empowers organizations to build, deploy, and scale applications with unparalleled agility and efficiency.

The benefits of Microservices development are manifold. Scalability becomes inherent as services can be independently deployed and scaled horizontally to meet varying demand. Fault isolation and resilience are heightened, ensuring that the failure of one service does not compromise the entire system. Additionally, Microservices facilitate continuous integration and deployment, enabling organizations to deliver updates and features at a pace that aligns with the speed of business requirements.

1.3 The Synergy of Script and Microservices Development

While script and Microservices development may seem distinct, their synergy is a potent force in modern software engineering. Scripts play a pivotal role in automating the deployment, monitoring, and management of Microservices. They become the glue that orchestrates the seamless interaction between services, streamlining development workflows and enhancing the efficiency of DevOps practices.

This integration fosters a holistic approach to software development, where the nimbleness of scripts meets the scalability of Microservices. Developers find themselves equipped with a comprehensive toolkit, enabling them to rapidly prototype ideas, automate repetitive tasks, and architect distributed, resilient systems that can evolve alongside the ever-changing demands of the digital landscape.

1.4 Navigating the Landscape: Challenges and Best Practices

As we delve into the realms of script and Microservices development, it is essential to acknowledge the challenges that come with embracing these paradigms. From managing the complexity of distributed systems to ensuring the security of automated scripts, developers must navigate a landscape that demands continuous learning and adaptation.

1.5 Structure of the Deliverable

This document reports on the work done in Work Package 6 “Edge-Core-Cloud Platform” which was carried out from month 8 (May ’23) to month 14 (November ’23) of the project, and is aligned to T6.2 “Script and Microservices Software - Intermediate”. Work Package 6 is organized in 3 tasks and there is a chapter dedicated to each of the tasks, after a short introduction in Chapter 1.

Chapter 2 describes the section “Distributed Knowledge Graph State of the art”. This section delves into the essence of a Distributed Knowledge Graph, elucidating its fundamental concepts, the data models employed for organizing knowledge, and the key technologies instrumental in its implementation.

Chapter 3 describes the section “Intermediate Script and Microservice Software for Novel Metadata Fabric”. This project focuses on developing intermediate-level scripts and microservices software designed to enhance the functionality of a novel metadata fabric. The aim is to create a sophisticated and responsive infrastructure, referred to as IceStream, which spans from the edge to the cloud. These intermediate scripts and microservices play a pivotal role in optimizing the operations of the metadata fabric, contributing to the overall efficiency and adaptability of the system.

Chapter 4 describes the section “Future Work”. In this section, future work is mentioned. In short, these are: Metadata Service, Data Processing and Monitoring Service, Replica Service, Prediction Service, Data Storage Service and Trade-off Service.

2 Distributed Knowledge Graph State of the art

GLACIATION aims to create a novel metadata fabric, called IceStream in the project, that will cover all levels of the continuum, from the edge to the cloud. The primary aim of the project is to reduce energy consumption by using artificial intelligence to enforce minimal data movement operations using the fabric.

The Distributed Knowledge Graph is the foundation of this system as it offers a real-time perspective of relevant data distributed across the network. It collects and organises information about the cluster, resource usage and application status. This provides a comprehensive and global view, helping to make informed decisions to optimise resource allocation in line with changing data requirements and use cases. This approach ensures that the knowledge graph, which is distributed, dynamic and adaptive, is not just a static construction but a responsive, living entity supported by functions and microservices that are further described in this deliverable.

This section will explain what a Distributed Knowledge Graph is, its fundamental concepts, data models used to organise the knowledge and the principal technologies that implement it.

2.1 Preamble

The DKG represents a paradigm in the realm of knowledge representation and data management. DKG provides a distributed, interconnected framework for storing, querying, and inferring knowledge, making it a crucial asset for applications where distributed data and microservices play a central role.

Before going in deep of this concept, it is important, firstly, to understand what a Knowledge Graph is. A Knowledge Graph is defined by Hogan in [HB21] as a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent potentially different relations between these entities. Data graph conforms to a graph-based data model, which may be a directed edge-labelled graph, a heterogeneous graph, a property graph, and so on.

Knowledge graphs are similar to specialized databases that collect information from various sources on diverse subjects. This amalgamation of data is like a melting pot, but what sets knowledge graphs apart is their ability to scrutinise data for hidden similarities and present it in one place. As a result, they earn their name 'knowledge' graphs. Thus, it is essentially a chart that consolidates data and connections from various sources that were previously separate.

Defining knowledge graphs can be challenging due to the general approach to schemas that describe data. If schemas exist, they may be ad-hoc and unenforced, leading to data that feels half-structured. This is not a reflection of poor quality or problematic incompleteness, but rather a critical distinguishing feature of knowledge graphs, giving them more flexibility.

There are new challenges for research and development including dealing with the **scale** and the **degree of distribution**, while monitoring and maintaining data quality and privacy. Initially, knowledge graphs were predominantly centralized, monolithic structures. However, as the need for distributed, scalable, and responsive systems grew, there arose a necessity to decentralize knowledge graphs. This shift in focus led to the emergence of **Distributed**

Knowledge Graphs, where the knowledge is distributed across multiple nodes or locations, enabling greater scalability and fault tolerance.

In recent years, this topic has become increasingly relevant in the IT sector and the European Cooperation in Science and Technology (COST) agency [CO23] is at the forefront of this work. Among the various campaigns, there is one dedicated to DKGs, for which the mission statement is quoted: Distributed Knowledge Graphs is a COST Action, i.e., a research and innovation network to connect research initiatives around Europe. A COST Action supports activities such as short-term scientific missions, workshops, and training schools. Our specific Action is centred around the topic of Distributed Knowledge Graphs, i.e., Knowledge Graphs that are published in a decentralised fashion, thus forming a distributed system. Essentially, the main aim of the Action is therefore to create a research community for deployable Distributed Knowledge Graph technologies that are standards-based, and open, embrace the FAIR principles, allow for access control and privacy protection, and enable the decentralised publishing of high-quality data.

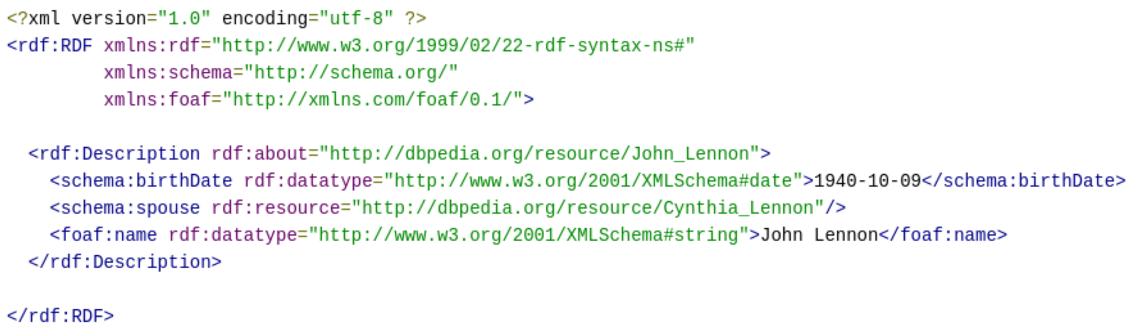
2.2 Fundamental Concepts

In a DKG, the fundamental building blocks include nodes, edges, properties, and schema. Nodes represent entities, concepts, or instances of knowledge, while edges establish relationships between nodes. Properties (or labels) are used to assign attributes or metadata to nodes and relationships. In the end, the schema specifies the structure and semantics of the graph, defining the types of nodes, edges, and properties that can exist.

Distributed Knowledge Graphs could employ various data models and representation techniques to efficiently organize, represent, and query knowledge in distributed environments. Efforts to standardize its representation are ongoing. RDF (Resource Description Framework), OWL (Web Ontology Language), SPARQL (a query language for RDF data), and the recent JSON-LD are some of the key standards used for knowledge graphs:

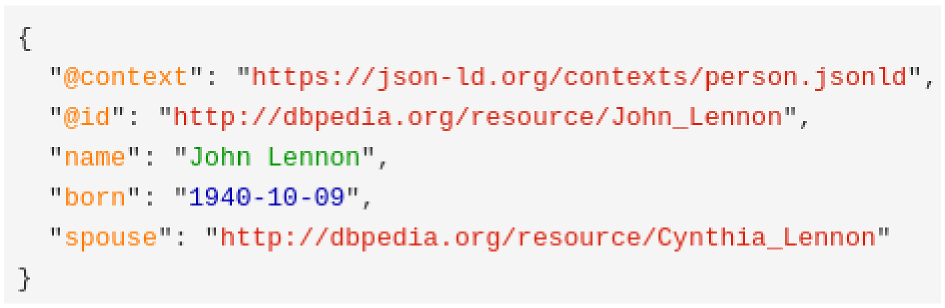
- RDF is a framework for modelling and exchanging data, it provides a set of specifications for representing data in the form of triples, which consist of a subject, a predicate, and an object. RDF can be used to create ontologies and to represent knowledge in a graph model.
- OWL is a language for creating ontologies. It is based on RDF and provides a more expressive way of describing concepts and relationships than RDF. OWL includes a set of constructs for representing classes, properties, and relationships between classes.
- SPARQL is a query language for RDF. SPARQL provides the functionality to query sets of RDF triples and also a communication protocol that allows queries to be made in a web environment.
- JSON-LD (JSON for Linking Data): JSON-LD is a data format for representing Linked Data in JSON format (while RDF is an XML-like format). It is designed for simplicity and ease of integration, making it suitable for scenarios where data interchange and interoperability across web-based sources are important. The exact same representation done in RDF are now displayed in JSON-LD:

A difference in syntax between RDF and JSON-LD could be visualized in these two examples, used for the representation of a person and its relevant information as the birth date, name and spouse.



```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:schema="http://schema.org/"
         xmlns:foaf="http://xmlns.com/foaf/0.1/">

  <rdf:Description rdf:about="http://dbpedia.org/resource/John_Lennon">
    <schema:birthDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1940-10-09</schema:birthDate>
    <schema:spouse rdf:resource="http://dbpedia.org/resource/Cynthia_Lennon"/>
    <foaf:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">John Lennon</foaf:name>
  </rdf:Description>

</rdf:RDF>
```

Figure 1 RDF example



```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

Figure 2 JSON-LD example.

While JSON-LD is very prominent and compact, RDF is the most mature standard for describing complex data and complex interrelationships for semantic data that do not fit into traditional relational database rows and columns or simple XML. It is also the standard that allows for data exchange of semantic graph databases that support contextual and conceptual meaning. RDF has features that dramatically simplify data merging even if the underlying schemas differ. Specifically, it supports the evolution of schemas over time without requiring all the data consumers to be changed – a powerful construct for today's fast pace of change. This flexibility is another advantage when dealing with complex data, where legacy systems and datasets are merged or brought together with new, and rapidly changing data sources. If you have complex data, RDF is an excellent, extremely robust data representation 'staging' format for data import and export.

2.3 Graph Databases

Here we present a table of the main open source as well as non-free graph databases in circulation, used also for the implementation of knowledge graphs.

Table 1 Graph-based DB.

Framework	Description
Apache Jena	Is an open-source Java framework for building Semantic Web and Linked Data applications. It provides a versatile and extensible platform for creating, storing, and querying knowledge graphs based on the RDF data model. Jena includes a triple store (TDB) for efficient RDF data storage and supports SPARQL, a query language for querying RDF data.
GraphDB	Developed by Ontotext, is a high-performance, semantic graph database designed for managing RDF data and knowledge graphs. It is known for its robust support for SPARQL querying and reasoning capabilities. GraphDB also offers features for semantic annotation, full-text search, and versioning, making it suitable for complex knowledge graph projects.
Neo4j	Is a leading graph database that specializes in property graph data modelling. It is widely used for building knowledge graphs and offers a powerful graph-based data model that includes nodes, relationships, and properties. Neo4j provides advanced querying, traversal, and graph analytics capabilities, making it well-suited for applications that require real-time graph processing.
Amazon Neptune	Is a managed graph database service provided by Amazon Web Services (AWS). It supports both property graph and RDF data models, making it versatile for various types of knowledge graphs. Neptune offers high availability, scalability, and ease of deployment in the cloud, making it a popular choice for building and managing knowledge graphs at scale.
Stardog	Is a semantic graph database designed for managing RDF data and creating knowledge graphs. It offers powerful reasoning capabilities, which means it can infer new information based on existing data. Stardog is suitable for applications that require advanced semantic reasoning and knowledge representation.

Next sections will provide more information related to script and microservice at support of the generation and management of DKG.

3 Intermediate Script and Microservice Software for Novel Metadata Fabric

3.1 Schema

A Knowledge Graph (KG) schema serves as the foundational blueprint for organizing and structuring information within a knowledge graph. It defines the essential components, including entities, properties, relationships, and their interconnections, to represent the interconnected knowledge of a specific domain. This schema acts as a guide for building and querying the knowledge graph, ensuring a systematic and coherent representation of information. By outlining the structure and semantics of data, a knowledge graph schema facilitates effective data integration, retrieval, and analysis, empowering applications ranging from semantic search to machine learning [ZM20].

A schema, providing a high-level framework as a guide, dictates the structure and semantics of a KG. Domain-specific KGs often have manually defined schemas with limited scale. Unlike traditional KGs with fixed schemas, open-domain KGs like Wikidata[1] and Freebase [BEP+08] lack a unified structure, with entity types evolving over time. Extracting KG schemas (or ontologies) has been explored for refining, completing, or adapting ontologies to new KGs, employing methods like ontology learning or taxonomy extraction. While traditional approaches involve manual efforts, recent solutions, including graph embedding techniques, address the schema modelling problem in fact-level KGs under hyper-relational representations, deviating from the assumption of a predefined ontology-level graph. [LDY+23]

In our specific scenario, the schema of our KG (to be more precise, Distributed Knowledge Graph - DKG) is deliberately and systematically crafted through manual definition, offering a fixed and unchanging structure. This intentional approach ensures a high level of precision and control over the representation of entities, properties, and relationships. To navigate this carefully curated knowledge landscape, it is imperative to accurately identify and acknowledge the ontologies that underpin our schema, signifying a commitment to clarity and coherence in our information architecture.

The identification of ontologies and the development of the metadata model constitute ongoing efforts within GLACIATION's *Task 6.1: Technical development of metadata model for data-centric architectures*. Initially, the work anticipated in T6.1 would have concluded by now and integrated into this section. However, due to valid considerations, the task has been extended. The following ontologies have been identified so far across distinct use cases:

3.1.1 MEF use case

- Human Resources Management Ontology (https://oeg.fi.upm.es/index.php/en/ontologies/99-hrmontology/index.html) - A comprehensive framework serving as a standardized vocabulary for expressing job postings and job seeker resumes. Developed using the NeOn Methodology and ontology

[1] https://www.wikidata.org/

engineering tools, it consists of thirteen modular components, covering areas such as Competence, Compensation, and Education.
- NoiPA Ontology (https://sparql-noipa.mef.gov.it/endpoint/download/owl/NoiPA.owl) - Specifically tailored for the MEF use case, the NoiPA Ontology addresses aspects related to the management of public employee payroll data. It is utilized in the MEF context to enhance data interoperability and understanding.

3.1.2 Dell use case

- *Foundational ontologies* - Essential ontologies providing a conceptual foundation for knowledge representation. They include the Time Ontology in OWL, defining temporal concepts, and the Vocabulary of Interlinked Datasets (VoID) for describing linked datasets.
  - Time Ontology in OWL (https://www.w3.org/TR/owl-time/)
  - Vocabulary of Interlinked Datasets (https://www.w3.org/TR/void/)
- *KG ontologies* - Key ontologies supporting the representation of knowledge graphs. This category includes the Resource Description Framework (RDF), a foundational language for expressing triples in the Semantic Web.
  - Resource Description Framework (RDF) (https://www.w3.org/TR/rdf11-concepts/)
- *Sensor/Telemetry ontologies* - Ontologies addressing sensor networks and telemetry data. Examples include the Semantic Sensor Network Ontology (SSN) for describing sensors and observations and the Robot ontology for robotic systems.
  - Semantic Sensor Network Ontology (https://www.w3.org/TR/vocab-ssn/)
  - Robot ontology [BCM17]
- *Data representation ontologies* - Ontologies focusing on the representation of data. Notable examples are the Data Catalog Vocabulary (DCAT) for describing datasets and the RDF Data Cube Vocabulary for representing multi-dimensional data.
  - Data Catalog Vocabulary (https://www.w3.org/TR/vocab-dcat/)
  - RDF Data Cube Vocabulary (https://www.w3.org/TR/vocab-data-cube/)
- *Workload representation ontologies* - Ontologies related to representing workloads, services, and APIs. This category includes Lightweight Semantic Descriptions for Services on the Web (WSMO-Lite), OWL-S for semantic web services, and the OpenAPI Specification for defining RESTful APIs.
  - Lightweight Semantic Descriptions for Services on the Web (https://www.w3.org/Submission/WSMO-Lite/)
  - OWL-S: Semantic Markup for Web Services (https://www.w3.org/Submission/OWL-S/)
  - OpenAPI Specification (https://swagger.io/specification/)

3.1.3 SAP use case

- Data Privacy Vocabulary (https://www.w3.org/community/dpvcg/2022/12/05/dpv-v1-release/) - This vocabulary addresses data privacy concerns, providing a standardized way to express terms and concepts related to data protection. It supports SAP in ensuring compliance with data privacy regulations.

After identifying the ontologies, the next step in the schema definition involves the establishment of relationships and mappings between the identified ontologies. This process is crucial for integrating and aligning diverse ontological structures to ensure a coherent and interoperable knowledge representation.

3.2 Implemented Scripts & Microservices

3.2.1 Metadata service

Is in the very early stage of development. We determine what will exactly constitute the metadata and which functions will be included to manipulate it. The usability of Python-based software for this purpose is discussed.

3.2.2 Prediction service

Provides predictions for time-series data on the platform. The prediction service is implemented using data on energy consumption. SPARQL queries are used to retrieve such data from a snippet of knowledge graph, which is used as illustration of implementation in each example. The forecasting is done using different methods such as AutoARIMA, Hybrid Bayesian Neural Networks, DeepAR, Temporal Fusion Transformer. Different methods are used to determine the conditions under which each of them shows the best performance to use optimized solutions in the future.

3.2.3 Data Storage Service

Main focus of the development now includes efforts to integrate SHACL (Shapes Constraint Language) validation within the data storage workflow. SHACL is a language for validating RDF (Resource Description Framework) graphs against a set of conditions.

3.2.4 Trade-off Service

The trade-off service will interact with the DKG to provide the workload placement with what it needs in terms of metadata for latency, resource usage, energy, privacy, data locality requirements and security. Its definition is in progress aligning with other technical components. Once the design is complete, development begins.

3.2.5 DKG Management Operations

This paragraph examines the essential processes required to build and maintain a knowledge graph model. We focus mainly on the creation, reading, writing, and algebraic operations (union, difference, intersection) in RDF. The method of managing the graph will be demonstrated here in a programmatic manner, with a focus on the Apache Jena framework, the core tool utilized in our project.

The goal of this mission is to provide a thorough understanding of the underlying processes while simplifying the complexity involved in creating RDF models through programming. Also, we include concise examples that show how the models are built directly in RDF/XML format as well, showcasing the adaptability and versatility of this data representation and of the Apache Jena tool. Although this presentation focuses on Apache Jena, it's crucial to remember that the basic functionality for managing a Distributed Knowledge Graph is the same for many graph database systems. As one considers building an RDF model, it becomes evident that the ideas mentioned above go beyond just one framework. Because these functions are widely used, they may be easily applied in a variety of graph database setups, which supports the flexible nature of knowledge graph management features.

3.3 Resource, Properties, Literals: Graphical, Programmatical and RDF/XML representation

Considered a major standard, the Resource Description Framework (RDF) is formally acknowledged as a W3C recommendation. RDF is fundamentally a basis for resource descriptions. A **Resource** is any entity we want to represent, which is uniquely recognized by a URI and represented in the graph as an ellipse. A URI is a sequence of characters that identifies a logical (abstract) or physical resource; it uniquely distinguishes one resource from another.

Resources in the RDF paradigm is linked to **properties**, which are represented by arcs labelled with the name of a certain property. Values/**Literals** are then connected to these attributes. The next example in Figure 3 Graphical representation of a simple RDF model., shows the representation of a person *Full Name*, where the value is represented as a text literal (a string enclosed in a rectangle).

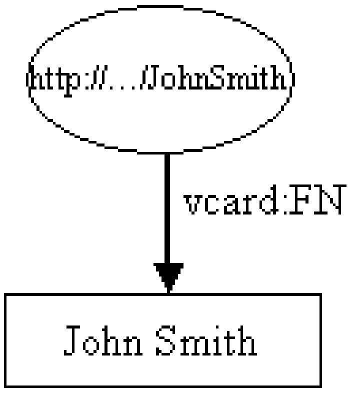


Figure 3 Graphical representation of a simple RDF model.

RDF graphs, such as the one above, can be created and altered using the Jena Java API. Specialized object classes designed for the representation of fundamental elements including graphs, resources, attributes, and literals are included in the tool and those are the main bread for developers that want to interact with and change the RDF structure. These interfaces are also referred to, for simplicity, as *Resource*, *Property*, and *Literal*. Rather, the term "model" refers to a graph, which is represented by the *Model* interface. Jena effectively makes the DKG navigation, query, and transformation possible through the Model interface. The same graphical example above could be represented programmatically as in Figure 4.

```
// some definitions
static String personURI    = "http://somewhere/JohnSmith";
static String fullName     = "John Smith";

// create an empty Model
Model model = ModelFactory.createDefaultModel();

// create the resource
Resource johnSmith = model.createResource(personURI);

// add the property
johnSmith.addProperty(VCARD.FN, fullName);
```

Figure 4 JAVA representation of a simple RDF model.

Jena has methods for reading and writing RDF as XML. These can be used to save an RDF model to a file and later read it back in again. The very similar model created programmatically above, is now showed here in XML, with the addition of the resource Jhon Smith having a Full Name and its division between family name and given name.

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
 >
  <rdf:Description rdf:about='http://somewhere/JohnSmith'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID="A0"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A0">
    <vcard:Given>John</vcard:Given>
    <vcard:Family>Smith</vcard:Family>
  </rdf:Description>
</rdf:RDF>
```

Figure 5 XML representation of a simple RDF model.

The syntax used in Figure 5 represents:

- The <rdf:Description> element describes the resource whose URI is http://somewhere/JohnSmith.
- The <vcard:FN> element describes a property of the resource.
- The value of the property is the literal "John Smith".
- The <vcard:N> element is a resource. In this case the resource is represented by a relative URI reference.

The three representations listed above are the current methods for creating a knowledge graph.

3.4 Access Information included in a Model

In an RDF model, every arc is referred to as a statement. A fact regarding a resource is asserted in each statement. A statement is sometimes called a triple because it has three parts:

- The *subject* is the resource from which the arc originates.
- The *predicate* is the property that labels the arc.
- The *object* is the resource or literal to which the arc points.

This format enables knowledge to be represented in a machine-readable way and with this precise representation, semantic data can be queried and reasoned about without ambiguity. To make an example, a triple could be "John is tall 45 cm", which comprises a subject "John", a predicate ("is tall"), and an object ("45 cm").

In the end an RDF model is represented as a set of statements. The Jena model interface defines a *listStatements()* method which returns an *StmtIterator*, a subtype of Java's Iterator over all the statements in a Model. StmtIterator has a method *nextStatement()* which returns the next statement from the iterator. Finally, from the *Statement* you could use methods to access the subject, the predicate and the object of a statement, as visualized in Figure 6.

```java
// list the statements in the Model
StmtIterator iter = model.listStatements();

// print out the predicate, subject and object of each statement
while (iter.hasNext()) {
    Statement stmt      = iter.nextStatement();  // get next statement
    Resource  subject   = stmt.getSubject();     // get the subject
    Property  predicate = stmt.getPredicate();   // get the predicate
    RDFNode   object    = stmt.getObject();      // get the object
```

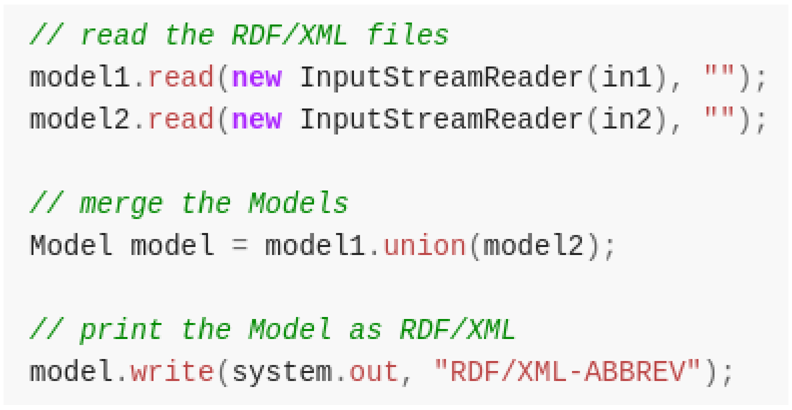
Figure 6 Retrieve statements.

Above we are making use of *listStatements()* which lists all the statements in a model, is perhaps the longest way of querying a model and its use is not recommended on very large Models. Fortunately, is possible to search based on specific property, querying it. The core Jena API supports only a limited query primitive, but more powerful query facilities are applied thanks to SPARQL that is not described here.

3.4.1 Operations on Models

Jena expands its capabilities by providing three fundamental model manipulation procedures. These operations, like set operations, include union, intersection, and difference.

Jena's union approach, which reflects the algebraic union operation, facilitates the smooth merging of two models. When the two models have identical nodes, they will be combined to form a single object that unifies both. Properties, the edges, encounters the same destiny since the union will eliminate superfluity and provides a concise representation in a single merged model. Figure 7 Multiple graphs union operation.shows how to read the two models and how to merge them in one.



```
// read the RDF/XML files
model1.read(new InputStreamReader(in1), "");
model2.read(new InputStreamReader(in2), "");

// merge the Models
Model model = model1.union(model2);

// print the Model as RDF/XML
model.write(system.out, "RDF/XML-ABBREV");
```

Figure 7 Multiple graphs union operation.

The intersection and difference of the models can be calculated in a similar way using the methods *.intersection(Model)* and *.difference(Model)*.

More information could be retrieved in the Apache Jena Documentation website [Jen15].

3.5 Designing a Microservice

Developing a microservice for script and microservices development involves creating a service that facilitates the execution of scripts and the deployment of microservices. Below are some important considerations and steps to take when designing and implementing such a microservice:

Define Requirements:

Clearly define the requirements for your microservice. What scripting languages do you want to support? What capabilities should it have for microservices development?

Choose Technology Stack:

Select a technology stack suitable for your microservice. Consider the programming language, framework, and any additional tools you may need.

Script Execution:

Implement the ability to execute scripts. Depending on your requirements, you might need to support various scripting languages (Python, JavaScript, etc.).

Input and Output Handling:

Design a mechanism to handle input parameters for scripts and capture their output. This may include specifying input parameters via an API or configuration file and capturing script output for further processing.

Error Handling:

Implement robust error handling to deal with issues that may arise during script execution. This includes catching syntax errors, runtime errors, and providing meaningful error messages.

Security:

Ensure security measures are in place. Limit script access to necessary resources, and consider implementing features such as sandboxing to prevent malicious activities.

Microservices Deployment:

If your microservice is also responsible for deploying other microservices, implement features for packaging, deploying, and managing microservices. This might involve containerization (using Docker) or other deployment strategies.

Version Control:

Consider incorporating version control features for scripts and microservices to track changes, rollback, and manage updates effectively.

API Design:

Design a clear and consistent API for your microservice. This API should cover script execution, microservices deployment, and any other functionalities your service provides.

Documentation:

Create comprehensive documentation for your microservice. Include details on how to use the API, sample requests and responses, and any other relevant information.

Testing:

Implement thorough testing, including unit tests, integration tests, and end-to-end tests, to ensure the reliability and correctness of your microservice.

Logging and Monitoring:

Integrate logging to capture relevant information during script execution and microservices deployment. Implement monitoring to track the health and performance of your microservice.

Scalability:

Design your microservice to be scalable. Consider factors such as load balancing and the ability to scale horizontally to handle increased demand.

Continuous Integration/Continuous Deployment (CI/CD):

Set up CI/CD pipelines to automate the testing and deployment process, ensuring a smooth and efficient development lifecycle.

User Authentication and Authorization:

Implement user authentication and authorization mechanisms to control access to your microservice.

Compliance and Governance:

If applicable, ensure your microservice complies with any industry or organizational governance standards.

3.6 Prediction Microservice

The prediction microservice aims to adaptively provide forecasting functionality for time series data on the GLACIATION platform such as the ones related to energy consumption or cloud workload. It is particularly useful for providing insights into the energy consumption or workload trends of the platform and can be used for other downstream tasks, such as resource allocation. Formally, the forecasting problem can be described as follows:

$$\widehat{y_{t+1}} = f(y_1,\ y_2,\ y_3,\ \ldots,\ y_t),$$

where $\widehat{y_{t+1}}$ refers to the predicted value at time step $t+1$ based on previous t observations in the time series data, and $f(\cdot)$ indicates a forecasting method to be used to map those historical observations to the predicted value [JA18].

The set of values that the random variable y_{t+1} could take, along with their relative probabilities, is known as the probability distribution of y_{t+1} given those previous observations. In forecasting, we refer to this distribution as the forecast distribution. This distribution with its confidence intervals is particularly useful in certain applications such as workload prediction for resource management in cloud computing, where overpredicting resource demand is more desirable than underpredicting the demand, as the latter can have a critical impact on Quality of Service (QoS).

In the following, we describe the input, output, a set of methods investigated and implemented, as well as key observations of the prediction microservice so far. At the end, we provide a summary and future work for this microservice.

Input

The input of the service should be an array of data representing a time series such as the historical energy consumption data or resource utilization history.

This can be obtained from DKG using a SPARQL query based on the metadata model defined in Task 6.1 in the future. For illustration purposes, we use a DKG snippet as an example. The snippet utilizes the PROV ontology [LSM+13] and the ontology of units of measure (OM) [RAT13] to provide relevant metadata, including the unit of a measurement, timestamp, and the measurement (value) itself.

In particular, the PROV ontology provides a set of classes, properties, and restrictions that can be used to represent and interchange provenance information generated in different systems and under different contexts. The OM ontology includes classes, instances, and properties that represent different concepts used for defining and using measures and units.

For instance, the snippet below shows a set of measurements in megawatt with respect to an energy consumption time series data aggregated at hourly intervals[2].

[2] https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption/

```
@prefix om: <http://www.ontology-of-units-of-
measure.org/resource/om-2/> .

@prefix prov: <http://www.w3.org/ns/prov#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .


[] a om:Measure,

        prov:Entity ;

    om:hasNumericalValue "1784.0"^^xsd:float ;

    om:hasUnit om:megawatt ;

    prov:generatedAtTime "2010-11-26T15:00:00"^^xsd:dateTime .


[] a om:Measure,

        prov:Entity ;

    om:hasNumericalValue "2694.0"^^xsd:float ;

    om:hasUnit om:megawatt ;

    prov:generatedAtTime "2009-01-21T13:00:00"^^xsd:dateTime .

…

…
```

Here, each data point or measurement in a time series is an instance of both `om:Measure` and `prov:Entity` classes, containing relevant information such as the value, unit, and generated time of the measurement.

Given such a time series data in DKG, we can use a SPARQL query to retrieve the data and use it as an input to the prediction service to forecast the next value. For instance, the following SPARQL query shows an example of obtaining the set of measurements stored in the above-mentioned DKG ordered by the generated time of each measurement. The retrieved set of measurements can be provided as an input to a forecasting method implemented within the prediction microservice.

```
   query = """

    prefix om: <http://www.ontology-of-units-of-
measure.org/resource/om-2/>

    prefix prov: <http://www.w3.org/ns/prov#>

    prefix xsd: <http://www.w3.org/2001/XMLSchema#>
    SELECT ?measure
    WHERE {
      ?subject rdf:type om:Measure .
      ?subject om:hasNumericalValue ?measure .
      ?subject prov:generatedAtTime ?date .
    }
    ORDER BY ASC(?date)
    """
```

Output

The primary output of the prediction microservice is the predicted value for the next time step based on the given time series data as input. In addition to the predicted value, the output also includes other useful metadata regarding the trained forecasting model and the prediction. This information offers an overview of the trained model, the evaluation results of the model, and the upper and lower bounds of the prediction using the confidence intervals.

More specifically, the metadata includes key elements such as:

- history: the input time series for training a forecasting model
- n_periods: the number of time steps to be forecasted
- evaluate: the number of time steps of the input to be used as testing examples for evaluation
- model: information about the trained forecasting model
- eval_results: the evaluation results in terms of Mean Absolute Percentage Error (MAPE) and Mean Squared Error (MSE)
- prediction: predicted value of the next time step
- upper_bound: the upper bound of 95% confidence interval of the prediction
- lower_bound: the lower bound of 95% confidence interval of the prediction
- fit_predict_time: the time spent on training and prediction of the model
- test_pred: the predicted values for testing examples
- test_upper_bounds: the upper bounds of test predictions
- test_lower_bounds: the lower bounds of test predictions

Without loss of generality, the output in a JSON format is illustrated as below.

```
{
  "history": [2219, 2096, …],
  "n_periods": 1,
  "evaluate": 10,
  "dump_path": "/forecasting/20231024-113258216461",
  "model": " ARIMA(3,1,4)(0,0,0)[0] intercept",
  "test_upper_bounds": [2661.3216631203404, …],
  "test_lower_bounds": [2485.2606919759073, …],
  "test_pred": [2573.291177548124, …],
  "test": [2576, …],
  "eval_results": {
    "mape": 0.01,
    "mse": 1132.7
  },
  "prediction": 1879.0052656737,
  "upper_bound": 1967.0357035331722,
  "lower_bound": 1790.9748278142279,
  "fit_predict_time": 106.79
}
```

In addition to the metadata of the output, a visualization can provide an intuitive insight on the performance of a trained forecasting model for a given input. In this regard, the microservice also provides auxiliary plots, such as an evaluation plot illustrating the testing examples and the forecasting results. For example, the testing examples can be the last 10 examples in the input data, and they are used to evaluate a forecasting model trained based on the input data, excluding those 10 examples.

Figure 8 below shows an example of such plot. In the plot, the set of “x” indicates the last 10 examples in the input time series data, while the solid line represents the prediction of the forecasting model. In addition, the shaded area of the plot represents the 95% confidence intervals, which are based on the upper and lower bounds of the prediction.

From the figure, we can observe that the trained model predicts the set of testing examples reasonably well, and all those examples fall within the 95% confidence intervals of the prediction. On one hand, those useful insights can provide confidence in our prediction microservice. On the other hand, we can also diagnose and improve a model in the cases where it does not perform well.

One thing to note is that those examples used for testing can also be used for updating the model after evaluation. That is, the prediction can still incorporate all the examples from the input data.

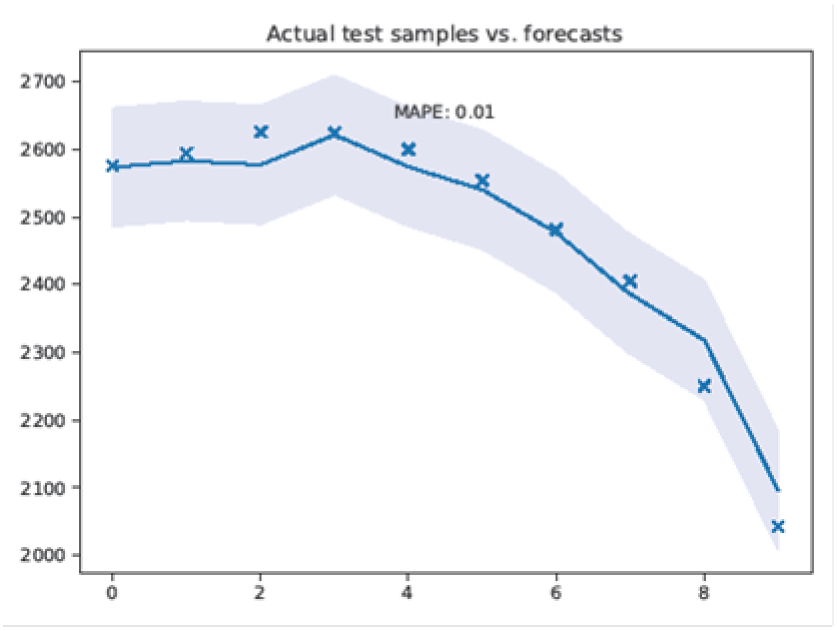


Figure 8 Example of forecasting results with 95% confidence intervals.

Methods

AutoARIMA [Smi17]: ARIMA, which stands for AutoRegressive Integrated Moving Average, is a well-established time series forecasting model. Due to its simplicity and powerfulness, it has been widely adopted in a wide range of applications such as predicting stock prices [AAA14], cloud resource [CMR14], and temperature trends [LD20].

ARIMA combines three key components:

- AutoRegressive (AR): The AR component models the relationship between a data point and its lagged (previous) values. The intuition behind this component is that the current data point depends on previous ones.
- Integrated (I): This component represents the number of differences required to make the time series stationary. Stationary means that the statistical properties of the data such as mean and variance, do not change over time.
- Moving Average (MA): The MA component models the relationship between the current data point and past white noise error terms. It aims to capture the short-term fluctuations or noise in the data.

AutoARIMA, as the name suggests, aims to automatically selecting best parameters in AR, I and MA components to find the best-performing ARIMA model for a given time series data.

HBNN (Hybrid Bayesian Neural Networks) [RVP+22]: A Bayesian Neural Network (BNN) is a type of neural network in which the weights are modelled as probabilistic variables and optimized via Bayes-by-backprop approach [BCK+15]. HBNN is a variation of BNN which employs a Bayesian approach in its last layer. This variation is made with the consideration of trade-off between training time and epistemic uncertainty modelling. Epistemic uncertainty here refers to the uncertainty arising from the limited dataset size or incomplete knowledge of model parameters [KHH20]. HBNN has been applied in workload prediction tasks in the context of cloud computing [RVP+22, RVP+23a, RVP+23b], and has shown its effectiveness of capturing the uncertainty of predictions.

The overall model architecture is illustrated in Figure 9 HBNN architecture.The input sequence is processed by 1-dimensional Convolutional Neural Networks (1DConv), LSTM (Long Short-Term Memory), and Dense layers, followed by the Bayesian layer [LBH15]. The final dense layer is a Gaussian distribution layer which contains two neurons representing the mean and variance of the distribution of the prediction.

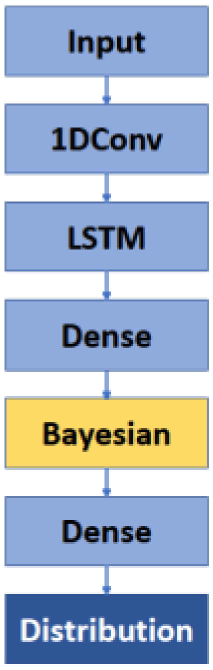


Figure 9 HBNN architecture.

DeepAR is a probabilistic forecasting model for time series data introduced by [SFG+20] in 2020. It leverages auto-regressive Recurrent Neural Networks (RNNs) and can be trained on one or multiple time series produced by the same or similar processes. To generate probabilistic forecasts, the model learns the parameters of a Gaussian distribution with the objective of minimizing the log-likelihood function using the Adam optimizer [KB14].

Temporal Fusion Transformer (TFT) [LAL+21] is an attention-based Deep Neural Network for time series forecasting. The model uses a multi-head attention mechanism to capture the most relevant segments of the input window for forecasting and utilizes a quantile loss as the loss function to optimize the network. Furthermore, multiple time series from diverse distributions have been used together to jointly optimize the network. The output of the network consists of the seven quantiles of a Gaussian distribution.

Although the accuracy of forecasting the future time series is important, some applications, such as workload prediction in the context of cloud computing, require a conservative prediction strategy. Underpredicting future workload can have a critical impact on QoS, which is critical for customer satisfaction. In our recent study [RVP+23b], we have also compared three deep learning models such as HBNN, DeepAR, and TFT for workload forecasting in cloud computing, as shown in Table 2. We also utilized five clustering approaches to explore the selection of similar time series datasets for training. For more details regarding these clustering approaches, refer to [RVP+23b].

- **Random** method chooses a set of training datasets randomly without using any clustering approaches.
- **PCA (Principal Component Analysis)** [WEG87] reduces the dimension of data into two with the first two principal components, and then cluster the data using a K-Means approach.
- **VRAE (Variational Recurrent Auto-Encoder)** [FA14] is a generative model for unsupervised learning for time series data in which an encoder is used to reduce the dataset dimensionality. Afterwards, K-Means clustering is applied to obtain clusters.
- **SOM (Self-Organizing Maps)** [Koh82] clusters data into a dimensionality-reduced map where each cell in the map represents a cluster, and the weight vector of a cell is trained using the data.
- **DTW (Dynamic Time Warping)** is an algorithm for measuring the similarity between time series datasets, and the similarities are used for clustering.

The set of metrics with respect to QoS are as follows:

- **Success Rate (SR):** the percentage of correctly predicted requests, i.e., the percentage of actual requests that are within the upper bound of the prediction.
- **Overprediction (OP):** the percentage of overprediction computed as the difference between the upper bound (UB) of the confidence interval and ground truth for those predictions which exceed the real demand of resources.
- **Underprediction (UP):** the percentage of underprediction computed as the difference between ground truth and the UB of the confidence interval for those predictions which are below the real demand of resources.

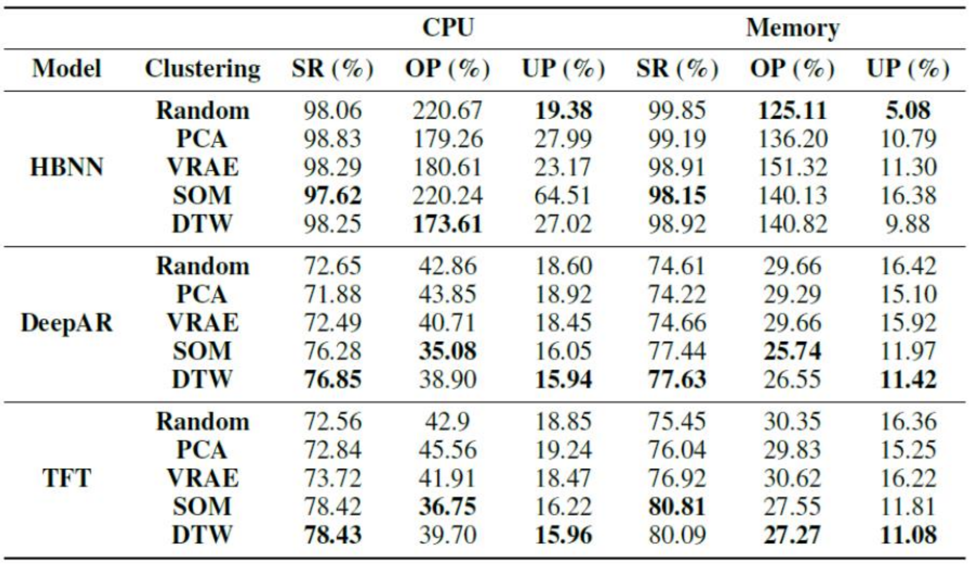
Ideally, a forecasting method should aim to achieve a high success rate while minimizing both overprediction and underprediction. Nevertheless, there are some trade-offs between these metrics. In cases with strict QoS requirements, achieving a high success rate is critical, even if it requires some compromise in terms of overprediction.

Table 2 shows the results of CPU and memory demand prediction regarding SR, OP, and UP, using 500 datasets for training. A clustering method is applied to choose relevant datasets. We can observe from the table that the SR values are higher for HBNN compared to DeepAR and TFT, regardless of clustering method applied. This indicates that the Bayesian layer in HBNN can effectively capture the epistemic uncertainty associated with predictions, as reflected by the magnitude of the standard deviation.

In contrast, DeepAR and TFT tend to exhibit overconfidence in their predictions, resulting in notably lower standard deviation values. For example, with the DTW as the clustering method, HBNN can achieve 98.25% of SR, while DeepAR and TFT achieve 76.85% and 78.43% of SR respectively. Regarding the UP metric which relates to the percentage of unmatched requests, the HBNN method exhibits a notably lower percentage of unmatched requests compared to the other two methods, making it a more reliable choice for applications that require high QoS. However, it is worth noting that these benefits of HBNN in terms of higher SR and lower UP come with a compromise of higher OP.

In summary, DeepAR and TFT tend to be overconfident in the prediction, while HBNN can provide a trade-off between accuracy and successful resource requests at the cost of allocating more resources.

Table 2 CPU and Memory demand prediction statistics.



		CPU			Memory		
Model	**Clustering**	**SR (%)**	**OP (%)**	**UP (%)**	**SR (%)**	**OP (%)**	**UP (%)**
HBNN	**Random**	98.06	220.67	**19.38**	99.85	**125.11**	**5.08**
	PCA	98.83	179.26	27.99	99.19	136.20	10.79
	VRAE	98.29	180.61	23.17	98.91	151.32	11.30
	SOM	**97.62**	220.24	64.51	**98.15**	140.13	16.38
	DTW	98.25	**173.61**	27.02	98.92	140.82	9.88
DeepAR	**Random**	72.65	42.86	18.60	74.61	29.66	16.42
	PCA	71.88	43.85	18.92	74.22	29.29	15.10
	VRAE	72.49	40.71	18.45	74.66	29.66	15.92
	SOM	76.28	**35.08**	16.05	77.44	**25.74**	11.97
	DTW	**76.85**	38.90	**15.94**	**77.63**	26.55	**11.42**
TFT	**Random**	72.56	42.9	18.85	75.45	30.35	16.36
	PCA	72.84	45.56	19.24	76.04	29.83	15.25
	VRAE	73.72	41.91	18.47	76.92	30.62	16.22
	SOM	78.42	**36.75**	16.22	**80.81**	27.55	11.81
	DTW	**78.43**	39.70	**15.96**	80.09	**27.27**	**11.08**

Required Python Packages

The prediction microservice has been developed using the Python programming language, along with relevant packages such as `pmdarima` and `Tensorflow` [Aba16] for implementing forecasting methods to be integrated into the service.

The required packages and their versions required for the prediction microservice include, but are not limited to:

- python 3.9.18
- arch 5.1.0
- keras 2.8.0
- matplotlib 3.8.0
- numpy 1.26.0
- pandas 2.1.1
- talos 1.0.2
- tensorflow 2.8.0
- tensorflow-gpu 2.8.0
- tensorflow-probability 0.14.0
- pmdarima 2.0.3
- scikit-learn 1.3.1
- scipy 1.11.3
- statsmodels 0.14.0

Summary

In this subsection, we introduced the prediction microservice with respect to its input, output, and a set of methods have been investigated. In addition to the prediction, the output of the service also contains relevant metadata for evaluating, diagnosing, and trouble-shooting the microservice. Since there is no one-fits-all solution for all time series data, the choice of a forecasting method to be used for a given time series data should be determined by considering different criteria. For instance, these criteria can include the performance of the forecasting model with respect to the time series data, the characteristics of time series data, and the set of service requirements in the GLACIATION platform and its use cases. Furthermore, the prediction microservice has been created as a standalone service until the point in time under discussion. In the future, it will be integrated into the metadata fabric to enable interaction with the distributed knowledge graph and other services, e.g., retrieving energy consumption time series data from DKG and providing forecasting results in the user interface of DKG.

3.7 Data Storage Service

The ongoing development of the Data Storage Service (DSS) includes efforts to integrate SHACL validation within the data storage workflow. This intermediate release reflects a snapshot of the continuous development process, with a particular focus on the SHACL validation component:

SHACL Validation Upon Data Storage

- Triggering Mechanism: The service is exploring methods for the automatic initiation of SHACL validation when RDF data is uploaded. This feature is in a developmental stage, with the goal of achieving a balance between performance and precision.
- Progressive SHACL Shapes Library: A preliminary collection of SHACL shapes has been initiated, serving as the basis for the validation routines. This library is under progressive elaboration, with plans to incorporate a wider array of shapes in line with evolving data specifications.
- Dynamic Shape Selection: Work is underway to develop capabilities that allow the DSS to intelligently determine the relevant SHACL shapes applicable to the uploaded data. This initiative is aimed at simplifying the validation process for the end-user by minimizing the manual intervention required.

Use-case for SHACL

SHACL (Shapes Constraint Language) is a language for validating RDF (Resource Description Framework) graphs against a set of conditions. These conditions are provided as shapes and other constructs in a SHACL shapes graph. SHACL is used in various contexts and scenarios, particularly where RDF data needs to be validated for consistency, correctness, and compliance with certain standards or expectations. Here are some of the use cases for SHACL:

1. **Data Quality Assurance:**
    a. Ensuring data adheres to defined quality criteria before it is used in applications.
    b. Detecting data anomalies, inconsistencies, and missing values.
2. **Data Modelling and Ontology Enforcement:**
    a. Enforcing domain-specific constraints in ontologies.

   b. Validating RDF data against ontology constraints to ensure that the data is a correct instance of the given ontology.

3. **Schema Validation:**
   a. Checking that RDF data conforms to a schema, similar to XML Schema or JSON Schema validation.
   b. Ensuring that relationships and properties within RDF data are used correctly according to a schema.
4. **Data Integration:**
   a. Validating integrated data from multiple sources to ensure it meets a common set of constraints.
   b. Helping with data alignment and transformation by highlighting discrepancies.
5. **Form Input Validation:**
   a. Validating user input in web forms that result in RDF data, ensuring that submitted data conforms to business rules.
6. **API Contract Validation:**
   a. Enforcing constraints on data that is consumed or produced by web services.
   b. Validating RDF payloads in RESTful services or SPARQL endpoints.
7. **Regulatory Compliance:**
   a. Checking data against regulatory requirements or industry standards.
   b. Using SHACL to demonstrate and ensure compliance with various regulations.
8. **Interoperability Assurance:**
   a. Ensuring that data shared between systems adheres to agreed-upon shapes, promoting interoperability.
   b. Validating standardized metadata for content like bibliographic records, scientific data, etc.
9. **Knowledge Graph Maintenance:**
   a. Periodic validation of knowledge graphs to ensure data quality and consistency over time.
   b. Assisting in knowledge graph curation by identifying data that does not meet certain conditions.
10. **Data Publishing:**
    a. Providing guarantees about the structure and quality of RDF data before it is published.
    b. Ensuring that published datasets meet the standards required by data consumers.
11. **Data Transformation and Mapping:**
    a. Verifying the results of data transformation processes.
    b. Ensuring that data mappings preserve semantic constraints.
12. **Ontology Evolution and Versioning:**
    a. Validating datasets against evolving ontologies.
    b. Ensuring backward compatibility of ontologies with existing datasets.

Our use-case for the data storage service is schema validation, data integration, and ontology enforcement. Future uses which the project will investigate include knowledge graph maintenance and ontology evolution.

Apache Jena SHACL

Apache Jena SHACL is an implementation of the W3C Shapes Constraint Language, designed for validating RDF graphs against a set of conditions. It supports both SHACL Core and SHACL SPARQL Constraints, which are used to define and enforce rules on the structure of RDF data.

One of the features of Jena SHACL is the SHACL Compact Syntax, which offers a more concise way of writing constraints compared to the standard SHACL syntax.

The tool is versatile, with command-line operations for validating RDF data, parsing SHACL files, and integrating with Apache Jena Fuseki, a SPARQL server. For instance, the command `shacl validate` can be used to check if an RDF graph meets the specified conditions using a SHACL shapes file. It is flexible enough to allow both the data and shapes files to be the same. Furthermore, the system can output validation results in various formats, including a compact text representation. For integration with Fuseki, one can configure a dataset to use SHACL validation by defining a service operation within the server's configuration.

Apache Jena SHACL also comes with an API, found in the package `org.apache.jena.shacl`, which provides classes for validating RDF graphs and updating them based on validation results. The API also supports the SHACL Compact Syntax for reading and writing, though it's worth noting that this syntax might not cover all shape possibilities and can be lossy for RDF data. Additionally, Jena SHACL includes SPARQL-based targets, which allow for dynamic determination of the nodes to be validated using SPARQL queries.

Finally, a ValidationListener can be used to monitor the validation process, receiving events throughout the validation lifecycle, such as when a shape starts or finishes validating, when focus nodes are identified, and when individual constraints yield results. This functionality allows developers to track the validation process programmatically and respond to events as needed.

Utilizing SHACL for Validating Intel RealSense Camera Data with ROS

Contextual Overview

The Intel RealSense camera is a sophisticated piece of technology that captures not just visual data in the form of images and videos but also depth information, collectively known as RGBD (Red, Green, Blue, Depth) data. This depth-sensing capability is critical for various applications, including robotics, augmented reality, and 3D scanning. We employ the Robot Operating System (ROS), a flexible framework for writing robot software, which serves as a conduit for processing the data captured by the RealSense camera. Within GLACIATION, the RealSense Camera Data will be generated by the cobots that are a part of Use Case 2 which is the manufacturing use case.

Metadata Representation with RDF

Each RGBD dataset generated by the Intel RealSense camera through ROS is accompanied by metadata that provides essential details about the data capture process. We represent this metadata using the Resource Description Framework (RDF), a standard model for data interchange on the web. RDF effectively encodes information about resources in the form of subject-predicate-object expressions, known as triples.

Here is an example of how the metadata for an RGBD file captured from the Intel RealSense camera is structured in RDF:

@prefix dc: <http://purl.org/dc/elements/1.1/> .

@prefix ex: <http://example.org/properties/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

```
ex:RGBDFile001
    a ex:RGBDFile ;
    dc:format "RGBD" ;
    dc:title "RealSense RGBD capture" ;
    dc:description "Depth and visual data captured using Intel RealSense camera." ;
    ex:width "1280"^^xsd:int ;
    ex:height "720"^^xsd:int ;
    ex:depthFormat "Z16" ;
    ex:colorFormat "RGB8" ;
    ex:timestamp "2023-11-03T10:00:00Z"^^xsd:dateTime ;
    ex:frame_id "camera_frame" ;
    dc:source <http://example.org/data/rgbd-file001> .
```

SHACL Validation

To ensure the integrity and correctness of the metadata, we implement SHACL (Shapes Constraint Language) for validation. SHACL allows us to define shapes or templates that describe the structure and constraints of RDF data. When RDF data is validated against SHACL shapes, it ensures compliance with the defined standards and requirements.

Below is a SHACL shape designed to validate the RDF metadata of the RGBD files:

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/properties/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .


ex:RGBDFileShape
    a sh:NodeShape ;
    sh:targetClass ex:RGBDFile ;
    sh:property [
        sh:path dc:format ;
        sh:hasValue "RGBD" ;
    ];
```

```
sh:property [
    sh:path dc:title ;
    sh:datatype xsd:string ;
    sh:minLength 1 ;
];
sh:property [
    sh:path dc:description ;
    sh:datatype xsd:string ;
    sh:minLength 1 ;
];
sh:property [
    sh:path ex:width ;
    sh:datatype xsd:int ;
    sh:minInclusive 1 ;
];
sh:property [
    sh:path ex:height ;
    sh:datatype xsd:int ;
    sh:minInclusive 1 ;
];
sh:property [
    sh:path ex:depthFormat ;
    sh:hasValue "Z16" ;
];
sh:property [
    sh:path ex:colorFormat ;
    sh:hasValue "RGB8" ;
];
sh:property [
    sh:path ex:timestamp ;
```

```
        sh:datatype xsd:dateTime ;
    ];
    sh:property [
        sh:path ex:frame_id ;
        sh:datatype xsd:string ;
        sh:minLength 1 ;
    ];
    sh:property [
        sh:path dc:source ;
        sh:nodeKind sh:IRI ;
    ];
```

The SHACL shape specifies constraints such as the required format for the depth and colour data, the minimum length for strings, and the requirement for certain properties to have specific values or data types. It serves as a blueprint to validate each RDF data entry against our expected standards, ensuring that any data ingested from the Intel RealSense camera via ROS conforms to our system requirements and is thus reliable for downstream processing and analysis.

3.8 Metadata Services

Metadata Services will be providing an API to interact with the Metadata of the model. Here, we understand Metadata in a broad sense, such as “data about data”, for example information about data creation: author, timestamp, source, external parameters (possibly including measured parameters of environment which could be relevant regarding estimation of energy costs) or data modification. Such metadata could be stored on each edge device and reflect the state of a part of the knowledge graph stored there. We will work with different meta-data types reflected in Metadata Architecture.

A database with all metadata entries will be the core of the service. Currently, we assess possible technological solution that would suit this role in the best way. Python scripts, based on, e.g., Flask framework, will work around the database providing easy ways to access data, write it, and remove it when necessary. Validation scripts will be added to check the compliance of entered metadata with standards which we are developing in close collaboration with our partners.

Metadata Services will be fully integrated to work with distributed knowledge graph and with other microservices.

More specifically, we are going to store such metadata (tentatively):

- Unique Entry ID
- Creation time
- Last edit time

- Last access time
- Author
- Source
- Associated energy costs/External energy data
- Work capacity of edge device
- References to scheme/ontologies relevant for the data
- Privacy and access rights information
- Environmental data (e.g., weather report)
- Data movement metadata
- Kubernetes specific metadata

We will provide tools for automatic and manual processing of entries. This would include generating of some information automatically on creation of a new entry, such as time stamps, as well as possibility to manually add Author and Source information. Some information will be obtained from external sources, such Energy data and Environmental data.

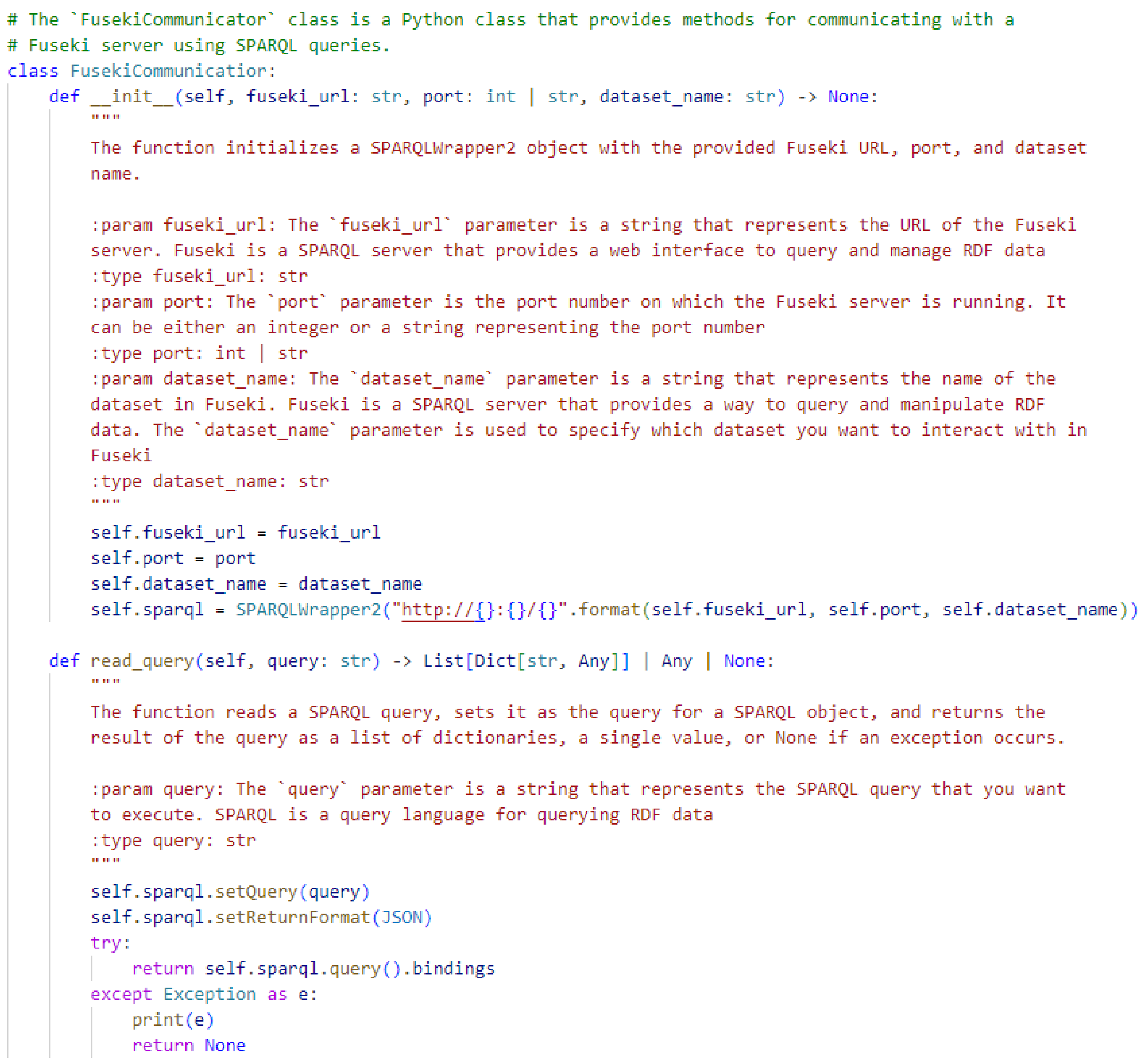
The list of foreseen API functions (not exclusive, tentative, list):

- **Create new entry.** Will accept entry text and target database as input. Will produce a new entry using entered data and state of the environment. It will record timestamp and measurements of energy and environment characteristics and assign a unique entry ID. Also, the function will check permissions of a user to write to a database.
- **List entries.** The function will list all the available entries on a device giving only short description.
- **Read entry.** Show detailed information about an entry.
- **Remove entry.** Delete an entry from the database.

The functions can also be more specific, when they will address different types of metadata, for example:

- Access to External Energy data
- Access to Environmental data (e.g., weather report)
- Policy broadly defined.
- Access to metadata for Data movement
- Access to Kubernetes-specific metadata

We should add that the search engine communicates with the Distributed Knowledge Graph (particularly, with the local Knowledge Graph stored on a device) through Metadata Services. The FusekiCommunicator framework is used to access the data and read it from Python, see Figure 10 Code snippet of the Fuseki Communicator class in Python.



```python
# The `FusekiCommunicator` class is a Python class that provides methods for communicating with a
# Fuseki server using SPARQL queries.
class FusekiCommunicator:
    def __init__(self, fuseki_url: str, port: int | str, dataset_name: str) -> None:
        """
        The function initializes a SPARQLWrapper2 object with the provided Fuseki URL, port, and dataset
        name.

        :param fuseki_url: The `fuseki_url` parameter is a string that represents the URL of the Fuseki
        server. Fuseki is a SPARQL server that provides a web interface to query and manage RDF data
        :type fuseki_url: str
        :param port: The `port` parameter is the port number on which the Fuseki server is running. It
        can be either an integer or a string representing the port number
        :type port: int | str
        :param dataset_name: The `dataset_name` parameter is a string that represents the name of the
        dataset in Fuseki. Fuseki is a SPARQL server that provides a way to query and manipulate RDF
        data. The `dataset_name` parameter is used to specify which dataset you want to interact with in
        Fuseki
        :type dataset_name: str
        """
        self.fuseki_url = fuseki_url
        self.port = port
        self.dataset_name = dataset_name
        self.sparql = SPARQLWrapper2("http://{}:{}/{}".format(self.fuseki_url, self.port, self.dataset_name))

    def read_query(self, query: str) -> List[Dict[str, Any]] | Any | None:
        """
        The function reads a SPARQL query, sets it as the query for a SPARQL object, and returns the
        result of the query as a list of dictionaries, a single value, or None if an exception occurs.

        :param query: The `query` parameter is a string that represents the SPARQL query that you want
        to execute. SPARQL is a query language for querying RDF data
        :type query: str
        """
        self.sparql.setQuery(query)
        self.sparql.setReturnFormat(JSON)
        try:
            return self.sparql.query().bindings
        except Exception as e:
            print(e)
            return None
```

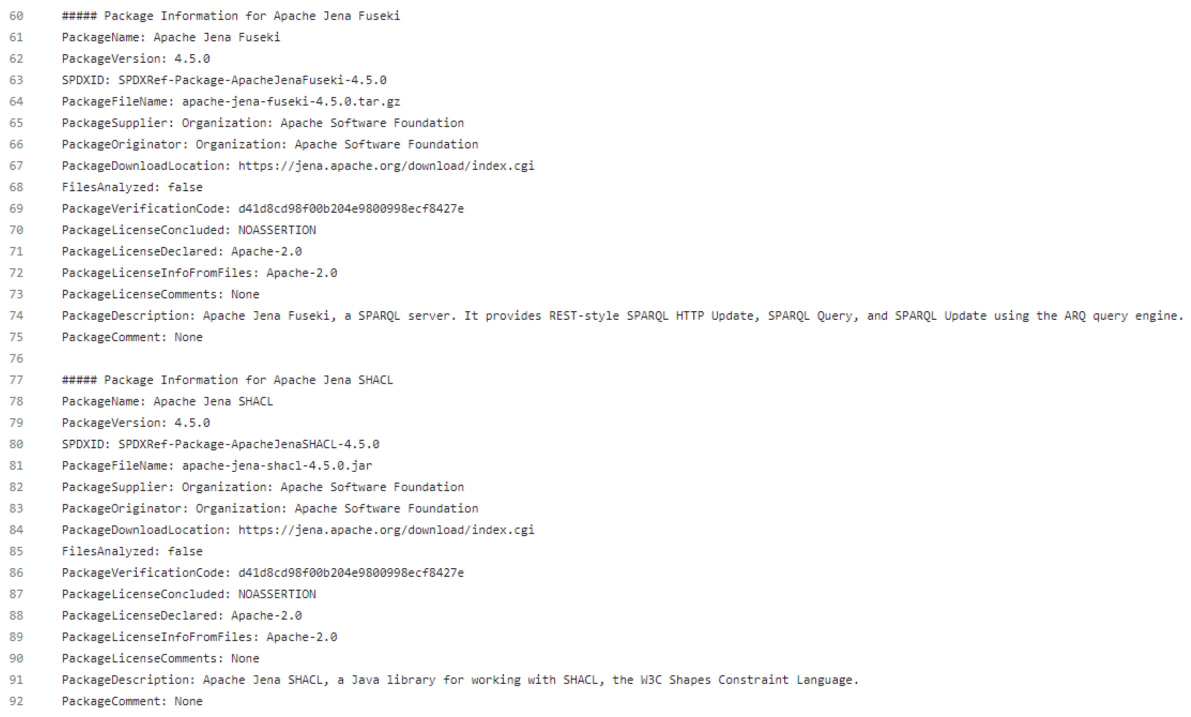
Figure 10 Code snippet of the Fuseki Communicator class in Python.

Metadata services are under active development and are planned for subsequent releases.

3.9 IceStream Software Release Description

Software Bill of Materials

In our Novel Metadata Fabric, IceStream, we utilize the Software Package Data Exchange (SPDX) format as a systematic approach to document the software components and their associated metadata. This standardized method is crucial for accurately cataloguing various aspects of software, including licenses, origins, and dependencies. Our software stack, incorporating tools like Apache Jena Fuseki, Apache Jena SHACL, and other integral software artifacts, is meticulously detailed in an SBOM using SPDX. This approach ensures thorough documentation of all components within our framework, aiding in adherence to open-source license requirements and efficient management of software bills of materials (SBOMs). Such detailed documentation is pivotal for our project's objective to measure software energy consumption transparently and responsibly, particularly given the complex interplay of multiple software layers. A section of the SBOM for IceStream, exemplifying this practice, is presented in Figure 11 SBOM snippet for IceStream software.

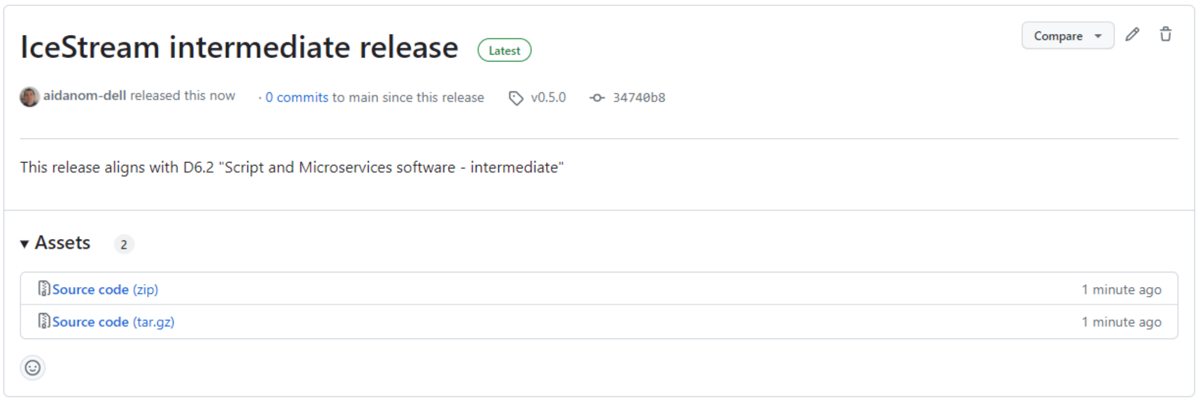


```
##### Package Information for Apache Jena Fuseki
PackageName: Apache Jena Fuseki
PackageVersion: 4.5.0
SPDXID: SPDXRef-Package-ApacheJenaFuseki-4.5.0
PackageFileName: apache-jena-fuseki-4.5.0.tar.gz
PackageSupplier: Organization: Apache Software Foundation
PackageOriginator: Organization: Apache Software Foundation
PackageDownloadLocation: https://jena.apache.org/download/index.cgi
FilesAnalyzed: false
PackageVerificationCode: d41d8cd98f00b204e9800998ecf8427e
PackageLicenseConcluded: NOASSERTION
PackageLicenseDeclared: Apache-2.0
PackageLicenseInfoFromFiles: Apache-2.0
PackageLicenseComments: None
PackageDescription: Apache Jena Fuseki, a SPARQL server. It provides REST-style SPARQL HTTP Update, SPARQL Query, and SPARQL Update using the ARQ query engine.
PackageComment: None

##### Package Information for Apache Jena SHACL
PackageName: Apache Jena SHACL
PackageVersion: 4.5.0
SPDXID: SPDXRef-Package-ApacheJenaSHACL-4.5.0
PackageFileName: apache-jena-shacl-4.5.0.jar
PackageSupplier: Organization: Apache Software Foundation
PackageOriginator: Organization: Apache Software Foundation
PackageDownloadLocation: https://jena.apache.org/download/index.cgi
FilesAnalyzed: false
PackageVerificationCode: d41d8cd98f00b204e9800998ecf8427e
PackageLicenseConcluded: NOASSERTION
PackageLicenseDeclared: Apache-2.0
PackageLicenseInfoFromFiles: Apache-2.0
PackageLicenseComments: None
PackageDescription: Apache Jena SHACL, a Java library for working with SHACL, the W3C Shapes Constraint Language.
PackageComment: None
```

Figure 11 SBOM snippet for IceStream software

Software Release Information

The IceStream software aligning with D6.2 is available on the GLACIATION GitHub repository as illustrated in Figure 12 IceStream Software Release.



IceStream intermediate release Latest

aidanom-dell released this now · 0 commits to main since this release v0.5.0 34740b8

This release aligns with D6.2 "Script and Microservices software - intermediate"

Assets 2

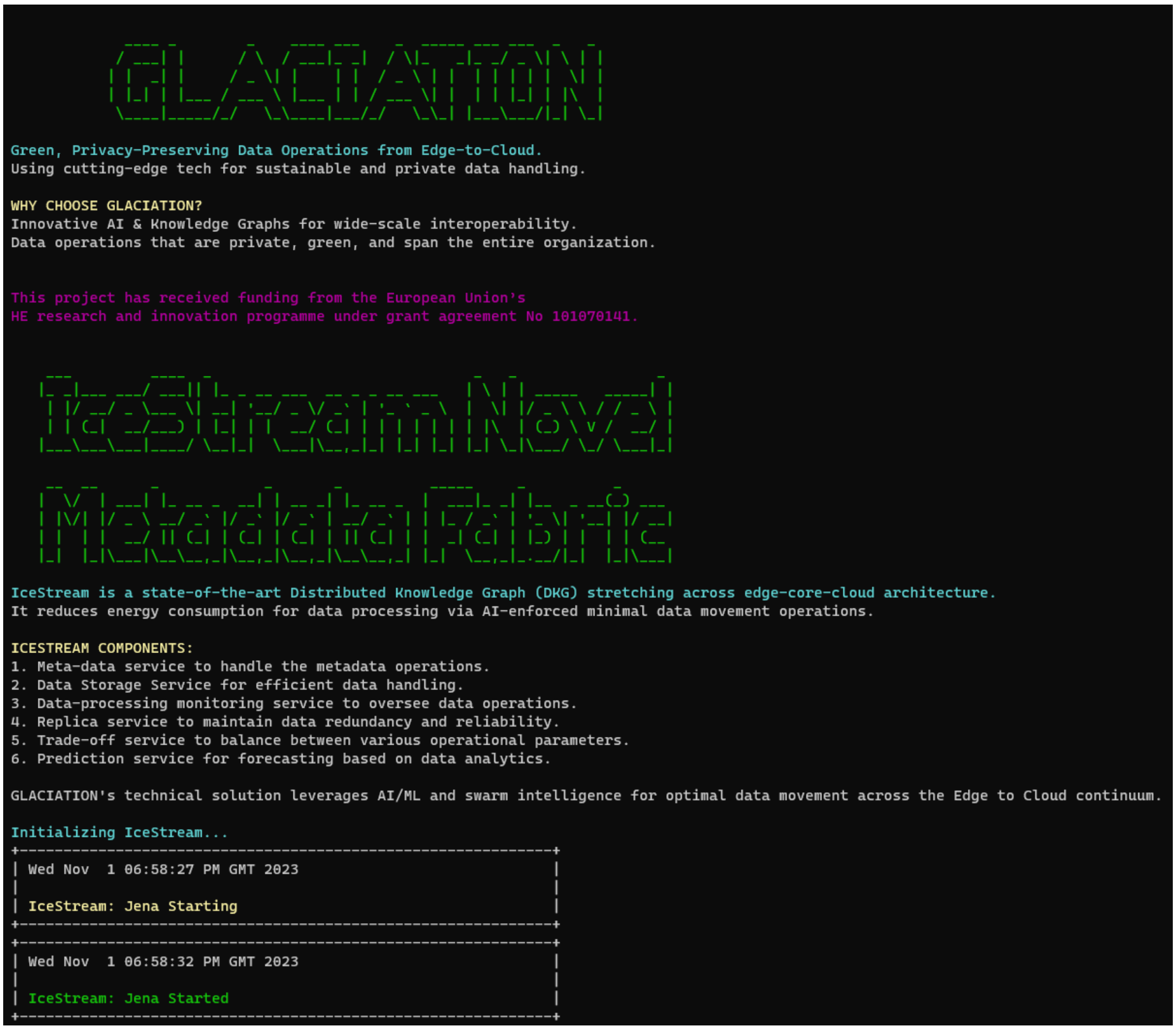
Source code (zip) 1 minute ago

Source code (tar.gz) 1 minute ago

Figure 12 IceStream Software Release

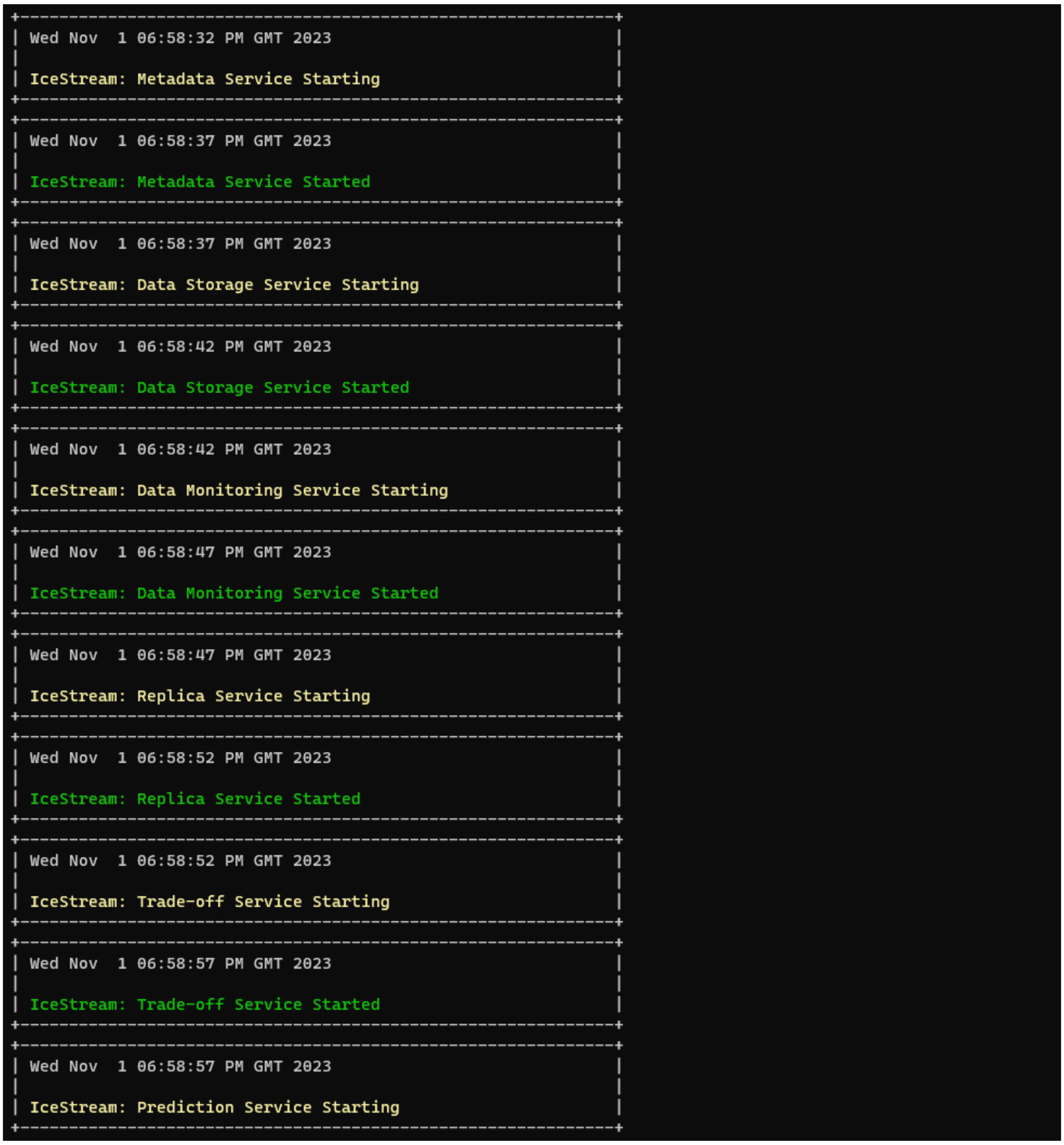
IceStream Deployment

As the GLACIATION platform is Kubernetes based, we need a consistent manner to deploy the various pods and daemonsets to the cluster. To do this we currently use a bash script. Each major component within GLACIATION has it's own bash script and are started in sequence. This is illustrated in Figure 13 IceStream Deployment Script Part 1 and Figure 14 IceStream Deployment Script Part 2.



```
          ____ _        _    ____ ___    _  _____ ___ ___  _   _
         / ___| |      / \  / ___|_ _|  / \|_   _|_ _/ _ \| \ | |
        | |  _| |     / _ \| |    | |  / _ \ | |  | | | | |  \| |
        | |_| | |___ / ___ \ |___ | | / ___ \| |  | | |_| | |\  |
         \____|_____/_/   \_\____|___/_/   \_\_| |___\___/|_| \_|

Green, Privacy-Preserving Data Operations from Edge-to-Cloud.
Using cutting-edge tech for sustainable and private data handling.

WHY CHOOSE GLACIATION?
Innovative AI & Knowledge Graphs for wide-scale interoperability.
Data operations that are private, green, and span the entire organization.


This project has received funding from the European Union’s
HE research and innovation programme under grant agreement No 101070141.


   ___          ____  _                              _   _                _
  |_ _|___ ___ / ___|| |_ _ __ ___  __ _ _ __ ___   | \ | | _____   _____| |
   | |/ __/ _ \\___ \| __| '__/ _ \/ _` | '_ ` _ \  |  \| |/ _ \ \ / / _ \ |
   | | (_|  __/ ___) | |_| | |  __/ (_| | | | | | | | |\  | (_) \ V /  __/ |
  |___\___\___||____/ \__|_|  \___|\__,_|_| |_| |_| |_| \_|\___/ \_/ \___|_|

   __  __      _            _       _          _____     _          _
  |  \/  | ___| |_ __ _  __| | __ _| |_ __ _  |  ___|_ _| |__  _ __(_) ___
  | |\/| |/ _ \ __/ _` |/ _` |/ _` | __/ _` | | |_ / _` | '_ \| '__| |/ __|
  | |  | |  __/ || (_| | (_| | (_| | || (_| | |  _| (_| | |_) | |  | | (__
  |_|  |_|\___|\__\__,_|\__,_|\__,_|\__\__,_| |_|  \__,_|_.__/|_|  |_|\___|

IceStream is a state-of-the-art Distributed Knowledge Graph (DKG) stretching across edge-core-cloud architecture.
It reduces energy consumption for data processing via AI-enforced minimal data movement operations.

ICESTREAM COMPONENTS:
1. Meta-data service to handle the metadata operations.
2. Data Storage Service for efficient data handling.
3. Data-processing monitoring service to oversee data operations.
4. Replica service to maintain data redundancy and reliability.
5. Trade-off service to balance between various operational parameters.
6. Prediction service for forecasting based on data analytics.

GLACIATION's technical solution leverages AI/ML and swarm intelligence for optimal data movement across the Edge to Cloud continuum.

Initializing IceStream...
+--------------------------------------------------------------+
| Wed Nov  1 06:58:27 PM GMT 2023                              |
|                                                              |
| IceStream: Jena Starting                                     |
+--------------------------------------------------------------+
+--------------------------------------------------------------+
| Wed Nov  1 06:58:32 PM GMT 2023                              |
|                                                              |
| IceStream: Jena Started                                      |
+--------------------------------------------------------------+
```

Figure 13 IceStream Deployment Script Part 1



```
+------------------------------------------------------------+
| Wed Nov  1 06:58:32 PM GMT 2023                            |
|                                                            |
| IceStream: Metadata Service Starting                       |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:37 PM GMT 2023                            |
|                                                            |
| IceStream: Metadata Service Started                        |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:37 PM GMT 2023                            |
|                                                            |
| IceStream: Data Storage Service Starting                   |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:42 PM GMT 2023                            |
|                                                            |
| IceStream: Data Storage Service Started                    |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:42 PM GMT 2023                            |
|                                                            |
| IceStream: Data Monitoring Service Starting                |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:47 PM GMT 2023                            |
|                                                            |
| IceStream: Data Monitoring Service Started                 |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:47 PM GMT 2023                            |
|                                                            |
| IceStream: Replica Service Starting                        |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:52 PM GMT 2023                            |
|                                                            |
| IceStream: Replica Service Started                         |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:52 PM GMT 2023                            |
|                                                            |
| IceStream: Trade-off Service Starting                      |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:57 PM GMT 2023                            |
|                                                            |
| IceStream: Trade-off Service Started                       |
+------------------------------------------------------------+
+------------------------------------------------------------+
| Wed Nov  1 06:58:57 PM GMT 2023                            |
|                                                            |
| IceStream: Prediction Service Starting                     |
+------------------------------------------------------------+
```

Figure 14 IceStream Deployment Script Part 2

4 Future Work

As we advance into an increasingly digital epoch, the roles of scripts and microservices in software development are pivotal. Their evolution is a testament to the industry's need for agility, scalability, and modularity. While the current state of script and microservices development has brought transformative changes to applications and systems, the horizon promises even more innovative shifts.

- **Metadata Service**

The conceptualization and development of a "Metadata Service" represent a pivotal aspect of system enhancement and refinement. The Metadata Service is an integral component designed to manage, organize, and provide access to metadata within a software ecosystem. Metadata, which encompasses descriptive information about data, processes, or resources, plays a critical role in contextualizing and optimizing the utilization of these elements within a system.

The development of a Metadata Service involves a multifaceted approach, addressing various aspects to ensure its effectiveness and seamless integration into the overall software architecture.

In essence, the conceptualization and development of a Metadata Service for future work embody a commitment to advancing data management capabilities within a software ecosystem. It serves as a linchpin for improved organization, accessibility, and utilization of metadata, thereby contributing to the overall efficiency and adaptability of the system. Through careful consideration of these diverse aspects, the Metadata Service becomes a catalyst for enhanced data governance and optimization in the evolving landscape of software development.

- **Data Processing and Monitoring Service**

Development and enhancement of a "Data Processing and Monitoring Service" represent a strategic initiative to elevate the capabilities of a software ecosystem. This service is designed to manage, process, and monitor data streams, providing critical insights and real-time analytics for informed decision-making.

Development of a Data Processing and Monitoring Service for future work is a strategic investment in elevating data-driven decision-making within a software ecosystem. By focusing on real-time processing, scalability, advanced analytics, security, and adaptability, this service becomes a cornerstone for organizations seeking to harness the full potential of their data assets in a dynamic and ever-evolving digital landscape.

- **Replica Service**

Conceptualization and development of a "Replica Service" represent a strategic endeavour to enhance data resilience, availability, and distribution. The Replica Service is designed to manage and replicate data across distributed nodes, ensuring redundancy, fault tolerance, and efficient data access.

- **Prediction service**

The "Prediction Service" stands as a forward-looking initiative aimed at enhancing the platform's capabilities in forecasting, anticipating, and proactively responding to dynamic changes within the computing environment. The Prediction Service is designed to harness advanced analytics and machine learning techniques to provide valuable insights, enabling informed decision-making and optimized resource allocation.

Prediction Service is a strategic initiative to elevate the platform's intelligence and responsiveness. By leveraging advanced analytics and machine learning, this service enables organizations to not only anticipate future states but also proactively shape the computing environment for optimal efficiency and performance.

- **Data Storage Service**

Represents an innovative initiative aimed at advancing data storage capabilities with the incorporation of SHACL (Shapes Constraint Language). The primary focus is on enhancing data validation, schema definition, and overall data quality within the storage service.

Shows a strategic move towards elevating data quality, schema flexibility, and overall storage capabilities. By harnessing the expressive power of SHACL, this service ensures that data stored within the Glaciation ecosystem is not only accurate and consistent but also adaptable to evolving data requirements.

- **Trade-off Service**

The "Trade-off Service" stands as an innovative initiative aimed at providing a systematic approach for evaluating and navigating trade-offs among various parameters within the system. This service is designed to empower users with the ability to make informed decisions by considering the interdependencies and consequences of different choices.

Strategic initiative to empower users with a systematic and informed approach to decision making. By considering multi-dimensional trade-offs, providing optimization capabilities, and fostering continuous improvement, this service becomes a cornerstone in achieving a balanced and adaptable computing environment within the GLACIATION ecosystem.

5 Conclusion

The objective of the intermediate deliverable is to present the progress of the schema and microservices development in Task 6.2 of Work Package 6 within the initial 14 months of the GLACIATION project. We have explored and investigated a set of ontologies for the schema regarding the three GLACIATION use cases and implemented several microservices.

In Chapter 2, we introduced the concept of a Distributed Knowledge Graph, its core building blocks which include various data models and representation techniques for organizing knowledge, and the key technologies for implementing it.

Chapter 3 first provided an overview of schema and a set of ontologies regarding three use cases, and then presented implemented scripts and microservices. These include the prediction microservice, data storage service, and metadata service developed to date. These microservices enable several functionalities, such as forecasting data popularity with different algorithms, SHACL validation within the data storage workflow, and metadata operations in the platform.

Finally, Chapter 4 discussed a list of future tasks that require improvement and development, building on top of the current progress of the schema, scripts and microservices. The ongoing development of schema and microservices, along with other planned microservices, will be integrated into the GLACIATION platform to fulfil the requirements for operating the novel metadata fabric by the end of the project.

References

[Aba16] M. Abadi. "TensorFlow: learning functions at scale." In *Proceedings of the 21st ACM SIGPLAN international conference on functional programming*, pp. 1-1. 2016.

[Jen15] Apache Jena. A free and open source java framework for building semantic web and linked data applications, 2015. jena.apache.org.

[RVP+22] A. Rossi, A. Visentin, S. Prestwich, and K. N. Brown. "Bayesian uncertainty modelling for cloud workload prediction." In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 19-29. IEEE, 2022.

[RVP+23a] A. Rossi, A. Visentin, S. Prestwich, and K. N. Brown. "Uncertainty-Aware Workload Prediction in Cloud Computing." *arXiv preprint arXiv:2303.13525*, 2023.

[Smi17] T. G. Smith, et al. pmdarima: ARIMA estimators for Python, 2017-, http://www.alkaline-ml.com/pmdarima [Online; accessed 2023-10-24].

[RAT13] H. Rijgersberg, M. V. Assem, and J. Top. "Ontology of units of measure and related concepts." *Semantic Web* 4, no. 1: 3-13, 2013.

[LSM+13] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao. "Prov-o: The prov ontology." *W3C recommendation* 30, 2013.

[RVP+23b] A. Rossi, A. Visentin, S. Prestwich, and K. N. Brown. "Clustering based numerosity reduction for cloud workload prediction." *AI4TS at IJCAI,* 2023

[BCK+15] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. "Weight uncertainty in neural network." In *International conference on machine learning*, pp. 1613-1622. PMLR, 2015.

[KHH20] A. Kristiadi, M. Hein, and P. Hennig. "Being bayesian, even just a bit, fixes overconfidence in relu networks." In *International conference on machine learning*, pp. 5436-5446. PMLR, 2020.

[LAL+21] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister. "Temporal fusion transformers for interpretable multi-horizon time series forecasting." *International Journal of Forecasting* 37, no. 4: 1748-1764, 2021.

[KB14] D. P. Kingma, and J. Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*, 2014.

[SFG+20] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. "DeepAR: Probabilistic forecasting with autoregressive recurrent networks." *International Journal of Forecasting* 36, no. 3: 1181-1191, 2020.

[JA18] R. J. Hyndman, and G. Athanasopoulos. Forecasting: principles and practice. *OTexts*, 2018.

[FA14] O. Fabius, and J. R. Van Amersfoort. "Variational recurrent auto-encoders." *arXiv preprint arXiv:1412.6581*, 2014.

[Koh82] T. Kohonen. "Self-organized formation of topologically correct feature maps." *Biological cybernetics* 43, no. 1: 59-69, 1982.

[WEG87] S. Wold, K. Esbensen, and P. Geladi. "Principal component analysis." *Chemometrics and intelligent laboratory systems* 2, no. 1-3: 37-52, 1987.

[LBH15] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning." *Nature* 521, no. 7553: 436-444, 2015.

[HB21] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. *ACM Comput. Surv.* 54, 4, Article 71 (May 2022), 37 pages. https://doi.org/10.1145/3447772

[CO23] COST (European Cooperation in Science and Technology) is a funding agency for research and innovation networks. https://cost-dkg.eu/

[BEP+08] Bollacker, Kurt, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. "Freebase: a collaboratively created graph database for structuring human knowledge." In Proceedings of the 2008 *ACM SIGMOD international conference on Management of data*, pp. 1247-1250. 2008.

[LDY+23] Lu, Yuhuan, Bangchao Deng, Weijian Yu, and Dingqi Yang. "HELIOS: Hyper-Relational Schema Modeling from Knowledge Graphs." In *Proceedings of the 31st ACM International Conference on Multimedia*, pp. 4053-4064. 2023.

[ZM20] Zouaq, Amal, and Felix Martel. "What is the schema of your knowledge graph? leveraging knowledge graph embeddings and clustering for expressive taxonomy learning." In *Proceedings of the international workshop on semantic big data*, pp. 1-6. 2020.

[BCM17] Buoncompagni, Luca, Alessio Capitanelli, and Fulvio Mastrogiovanni. "A ROS multi-ontology references services: OWL reasoners and application prototyping issues." *arXiv preprint arXiv:1706.10151* (2017).

[LD20] Y. Lai, and D. A. Dzombak. "Use of the autoregressive integrated moving average (ARIMA) model to forecast near-term regional temperature and precipitation." *Weather and Forecasting* 35, no. 3: 959-976, 2020.

[AAA14] A. A. Ariyo, A. O. Adewumi, and C. K. Ayo. "Stock price prediction using the ARIMA model." In 2014 *UKSim-AMSS 16th international conference on computer modelling and simulation*, pp. 106-112. IEEE, 2014.

[CMR14] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. "Workload prediction using ARIMA model and its impact on cloud applications' QoS." *IEEE transactions on cloud computing* 3, no. 4: 449-458