Scaling Knowledge Graphs for Automating AI of Digital Twins

Joern Ploennigs(✉), Konstantinos Semertzidis, Fabio Lorenzi, and Nandana Mihindukulasooriya

IBM Research Europe, Dublin, Ireland
Joern.Ploennigs@ie.ibm.com,
{konstantinos.semertzidis1,fabio.lorenzi1,nandana}@ibm.com

Abstract. Digital Twins are digital representations of systems in the Internet of Things (IoT) that are often based on AI models that are trained on data from those systems. Semantic models are used increasingly to link these datasets from different stages of the IoT systems life-cycle together and to automatically configure the AI modelling pipelines. This combination of semantic models with AI pipelines running on external datasets raises unique challenges particular if rolled out at scale. Within this paper we will discuss the unique requirements of applying semantic graphs to automate Digital Twins in different practical use cases. We will introduce the benchmark dataset DTBM that reflects these characteristics and look into the scaling challenges of different knowledge graph technologies. Based on these insights we will propose a reference architecture that is in-use in multiple products in IBM and derive lessons learned for scaling knowledge graphs for configuring AI models for Digital Twins.

Keywords: Knowledge graphs · Semantic models · Scalability · Internet of Things · Machine learning · Digital twins

1 Introduction

Semantic models are establishing across industries in the Internet of Things (IoT) to model and manage domain knowledge. They range from driving the next generation of manufacturing in Industry 4.0 [3,17,19], to explainable transport [29], energy savings in buildings for a sustainable future [5,11]. Their application cumulates in the use of semantic integration of various IoT sensors [28] to automate analytics of the created data [11,37].

Digital Twins are one area of applying semantic models. A *Digital Twin* is a digital representation of an IoT system that is able to continuously learn across the systems life cycle and predict the behaviour of the IoT system [26]. They have multiple uses across the life cycle from providing recommendations in the design of the system, to automating its manufacturing and optimizing its operation by diagnosing anomalies or improving controls with prediction [36]. The core of a Digital Twin is formed by two tightly interacting concepts. First, an AI model, such as a Machine Learning (ML) or simulation model, that is capable

U. Sattler et al. (Eds.): ISWC 2022, LNCS 13489, pp. 810–826, 2022.
https://doi.org/10.1007/978-3-031-19433-7_46

of continuous learning from data and explaining and predicting its behaviour. Second, a *Digital Thread* that is linking these underlying data sources across the systems life cycle [34]. Both approaches interact tightly as the Thread needs to be used to automate the configuration of the AI models to allow to scale their application while results from the AI model should be injected back into the Thread to learn and explain knowledge.

Semantic Knowledge Graph technologies are very well suited for implementing this Digital Thread [1]. They promise to solve several common challenges from normalizing labelling of the various data sources to being flexible enough to be extended over the life cycle when new applications arise [8]. However, scaling knowledge graphs is challenging in its own terms [24] and in our practice we experience multiple issues in scaling Digital Threads. Within this paper we will deep dive into this use case and discuss some of the practical issues. We will follow the typical industry workflow for designing and selecting a solution from collecting requirements and defining a test example, to deriving a reference architecture and evaluating final realization options for some large scale examples. The contributions of the paper are:

- *Requirements for Digital Twins*: We collect the requirements for semantic representation of Digital Twins in Sect. 3.
- *In-Use Experience for Scaling*: We discuss our in-use experience in scaling Digital Twins and propose a reference architecture.
- *Benchmark model for Digital Twins*: We define a benchmark model for semantic Digital Twins for an manufacturing example that tests some of the identified requirements in Sect. 5.
- *Comparison of KG Technologies*: We compare different knowledge graph technologies for managing the semantic models for Digital Twins in Sect. 6 including our own semantic property graph.

2 State of the Art

Knowledge Graphs for Digital Twins: There are several examples of applying semantic models for representing Digital Twins [1,18,20]. Kharlamov et al. [18] argues for the benefits of using semantic models for digital twins e.g. to simplify analytics in Industry 4.0 settings. Similarly, Kalayci et al. [17] shows how to manage industry data with semantic models. Lietaert et al. [21] presents a Digital Twin architecture for Industry 4.0. Chevallier et al. [10] proposes one for Smart Buildings. Akroyd et al. [1] reviews multiple approaches for geospatial knowledge graph for a Digital Twin of the UK. Their work demonstrates the challenges in incorporating data from different domains into one knowledge graph like the heterogenity of data sources. These example represent the use of semantic models for building Digital Twins in different industries that we also see in practise.

Semantic Data Management: A common goal of using knowledge graphs for Digital Twins is to integrate data from various systems. Established solutions exist for doing this with semantic knowledge graphs that also may integrate external

data. Pan et al. [25] presents a survey of semantic data management systems and benchmarks. The authors classify the systems using a taxonomy that includes native RDF stores, RDBMS-based and NoSql-based data management systems. Besta et al. [6] provide a classification based on the database technology. Their analysis shows that the different design have various pros and cons. Some of the widely-used generic triple-stores such as OpenLink Virtuoso [13], Apache Jena, Blazegraph, GraphDB excel on managing RDF data, but, do not scale well in integrating non RDF data. General purpose property graphs like Neo4J or JanusGraph lack intrinsic understanding of semantic models. Multi-modal databases like ArgangoDB or Redis combine a no-sql database with a graph database that allows to manage documents alongside the graph. But, they also suffer from a good understanding of semantic [30]. Entris [12] and Schmidt [31] extend this idea and use semantic models to manage additional data in a data lake. In Sect. 3 we will discuss some unique requirements that create challenges in scaling such knowledge graphs. We derive a reference architecture that separation the semantic graph layer from the data layer to scale better to large volumes of data and have federated access to address the Semantic Digital Threads requirements. As shown by our experiments, such design seems to provide better scalability for our use case compared to the other semantic data management approaches.

Benchmarks for Semantic Data: To validate that the requirements in modelling Digital Twins are unique and evaluate different knowledge graph technologies, we created a new Digital Twin Benchmark Model (DTBM). We compare it against some established benchmarks. The *Berlin SPARQL Benchmark (BSBM)* [7] and *Lehigh University Benchmark (LUBM)* [14] are generic RDF Benchmarks that run a variant of queries on generated datasets. *SP²Bench* [33] is based on DBLP library dataset and reflects the social network characteristics of semantic web data. *DBpedia SPARQL benchmark* [23] uses real queries that were performed by humans and applications against on DBpedia. Additional work reflects the requirements and characteristics of certain domains. *PODiGG* [35] and *GTFS-Madrid-Bench* [9] are examples of benchmarks for public transport domain focused on use cases and requirements on route planning on gespatial and temporal transport data. *LSLOD* [15] contains datasets from the life sciences domain and the Linked Data cloud and 10 simple and 10 complex queries that need query federation. *Fedbench suite* [32] evaluates efficiency and effectiveness of federated SPARQL queries over three different datasets: cross-domain, life science, and SPBenc. We will use BSBM and LUBM in the evaluation in Sect. 6 as they are very well established and tested for many knowledge graphs technologies and address themselves different RDF characteristics. In addition, we will propose a new benchmark focused on our use case.

3 Requirements for Semantic Digital Threads

A Digital Thread is linking data from different life cycle stages of a Digital Twin. This starts from design documents such as textual requirements, test

specifications, to CAD-files and handbooks that may exist in different formats. During production additional properties may be attached to a Digital Twin such as sensor data from machines, materials used, material providers, and test results. During operation the data collected from the final system is also added. It is often related to asset management data such as fault reports, maintenance work-orders, and replacement histories as well as timeseries data collected from IoT sensors embedded in the systems such as temperature measurements, operational states or alarms. The different datasets that are collected across the life cycle are linked together in the Digital Thread and often analyzed by Machine Learning algorithms to discover and explain anomalies, predict the behaviour of the system and advise people in improved manufacturing and operation of the system.

From this description we can synthesize some *characteristics* to a semantic knowledge graph that can be used to implement such a Digital Thread.

- *C1 - Heterogenous Semantic Types:* The connected data is very heterogenous, representing domainspecific semantic types. A domain ontology can contain thousands of types. For example, the BRICK ontology [5] contains ca. 3.000 classes for modelling smart buildings datasets.
- *C2 - Multi-modal Representation:* The data is multi-modal and represented in different formats from timeseries, to binary files, and text documents.
- *C3 - Federated Data:* The data is stored and managed in various systems such as complex Continuous Engineering Systems, Asset Management Systems, or IoT platforms.
- *C4 - Flexible Hierarchies:* Data is often structured in hierarchical models such as location hierarchies (`Country` > `City` > `Factory` > `Production_Line`) and asset hierarchies (`Robot` > `Arm` > `Joint`) that are of flexible depth.
- *C5 - Large size:* We see graph sizes often in the range of 100.000 datasets for a mid-size Digital Twin.
- *C6 - Composability:* Digital Twins often contain other Digital Twins. For example, a factory twin may contain a robot twin.
- *C7 - Lack of semantic knowledge:* We often experience that domain experts do not have deep semantic knowledge. Though, they often understand software engineering concepts like classes and inheritance.
- *C8 - Dynamic:* Digital Twins change over their lifetime and so does the Digital Thread. In consequence, the knowledge graph does change regularly bringing in the need to represent time, states and versioning.

The *goals* for building the Digital Thread are:

- *G1 - Data Linking:* The first goal of the Digital Thread is to link data from various life cycle stages and backend systems together to create an integrated view of the data.
- *G2 - Data Contextualization:* The second goal of building Digital Threads is to contextualize the data and understand spatial and functional context to summarize and explain the data.

- *G3 - Data Model Normalization:* The third goal is to reduce the heterogeneity of the underlying data and normalize it on: common semantics (C1), a common data modality (C2), and common hierarchical model (C4).
- *G4 - Data Access Abstraction:* The next goal is to abstract the access to the underlying data in the federated systems (C3) to allow users to query data by its semantics rather then storage specific IDs like asset or sensor IDs.
- *G5 - AI Automation:* The final goal is to automate lifecycle processes like analytics. This is needed as manual configuration of these analytic processes is not possible due to the data size (C5) and regular changes (C8).

From these characteristics and goals we can derive a set of requirements:

- *R1 - Domain Taxonomies:* From G3 and C1 we can derive the requirement to model both normalized upper ontologies (e.g. `Sensor`) with generic types on the top and more specific types on lower level of the taxonomy (e.g. `Temperature_Sensor` $\sqsubseteq$ `Sensor`).
- *R2 - Subsumption:* From C1 and R1 we can derive the requirement to use subsumption in the taxonomies. Taxonomies may have multiple levels, e.g. most sensor tags in BRICK have about 3–5 levels of parents.
- *R3 - Inheritance:* From C1, C6, C7 we can derive the requirement to support inheritance of properties from concepts to instances. In practise, we use this heavily to propagate for example units of measurement or ML model configurations.
- *R4 - Semantic Data Access:* The solution needs to provide semantic data access according to G4 to the underlying federated data (C3).
- *R5 - Backend agnostic:* The system needs to support various data representation (C2) and federated storage solutions (C3) in a hybrid cloud.
- *R6 - Flexible Depth Queries:* The hierarchies from C4 provide some means of structuring and querying data. However, the lack of defined structures with fixed query depth requires the use of transitive property chains in queries.
- *R7 - Event-based Reasoning:* Reasoning approaches are a good way of automating processes in the knowledge graph for G5 to replicate knowledge for sub-components (C6). The high dynamic of the graph (C8) asks for ways to automate these reasoning steps on graph events, when for example sub-components are added.
- *R8 - Guaranteed Consistency:* C1 and C8 mean also that users regularly change the domain taxonomies and there need to be ways to propagate changes in the subsumption taxonomies or the deletion of concepts that keep the graph consistent and not end up with orphaned elements.
- *R9 - Element-level Access Control:* The Digital Twin is integrating data from various systems (C3, C6) and needs to support different use cases and user roles (C8). In consequence, a fine grained access control on graph element level is needed [4].

Some of these requirements are of common nature for semantic knowledge graphs like R1, R2, R5 and therefore support the applicability of this technology. We consider the requirements R3, R4, R5, R7, R8, R9 more specific for Digital Twins and do not see them in other applications [22,24].

4 Architecture for Semantic Digital Twins

Based on the goals and requirements defined in the last section, we derive a reference architecture for a Semantic Digital Twin in Fig. 1. We keep the reference architecture on purpose generic as we need to support different backends (C3) and want to give readers implementation options. We realized the reference architecture in our own implementation KITT, which is used in different products like IBM Maximo® or IBM TRIRIGA® to integrating data from multiple different solutions in a Digital Threads. The system is deployed and in-use for various customers since multiple years.

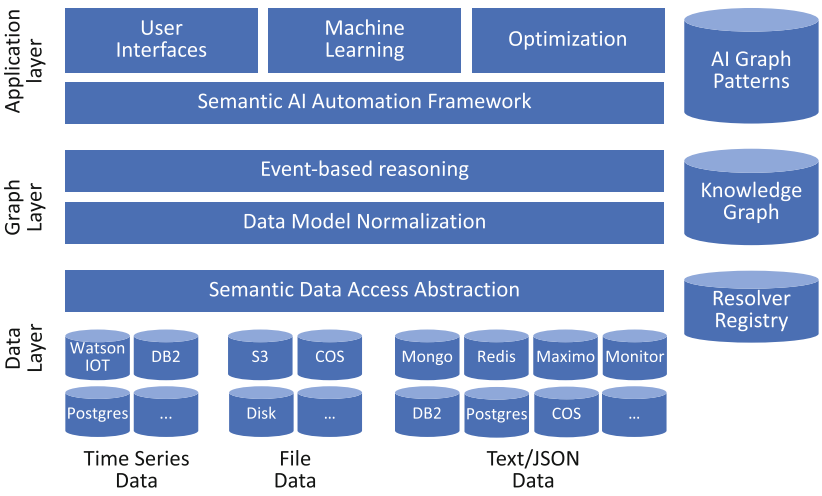

Fig. 1. Reference architecture for semantic Digital Twin

The core design of the architecture is a separation of the data layer from the graph layer. The *data layer* links to data from various federated backend systems. They are integrated by a *Semantic Data Access Abstraction Layer*. This is a microservice environment that is implementing for each backend a resolver microservice that maps the backends onto a key/value interface to query the respective data in a normalized way according to G4. We differentiate three types of data representations to address the multi-modality (C2): (i) Timeseries Data (DB2®, PostgreSQL, Watson IoT®, ...); (ii) Binary File Data (Disk, RocksDB, COS®, ...); (iii) Textual/JSON Data (MongoDB, Redis, Maximo®, ...). A registry allows to dynamically add new resolvers. The mapping to a key/value interface provides a simple interface for implementation of the resolvers and allows us to store the keys as properties in the graph.

The *graph layer* contains the knowledge graph that is linking the meta-data from the various backends (G1). The semantic graph represents the normalized data model (G3) annotated with the associated keys from the data layer.

By storing only the data keys in the graph we get a separation of the metadata graph from the data layer that serves two purposes. First, it keeps the graph manageable in size as the underlying large volume of raw data that often reaches multiple terabytes are kept out of the graph. Second, it helps to solve the federated data problem as clients do not want to move the data out of the master data systems.

We developed our own in-house knowledge graph technology based on our learnings from using over the years different off-the-shelf triple stores and property graphs as backend. Due to the unique requirements, none of them could scale to achieve the desired performance (see Sect. 6) and manageability targets. Based on our experience with triple stores they present two challenges. First, the graph sizes quickly explode when large numbers of properties (like keys) are added, which are secondary for traversals. Although they provide good support for basic RDF semantics most triple stores do not support subsumption inference out-of-the-box (R2) and rely on rule-based materialization. This not only increases the triple number, but, more importantly it is hard to manage consistency in our production context. When users change the structure of taxonomies it is not easy and efficient to update and delete materialized triples such that a consistent state of the graph can be maintained (R8). Property graphs separate the graph from properties and thus could scale better with large number of properties but they do not necessarily are targeting RDF natively resorting to arbitrary conversion tools to support such scenarios. This lack of semantic understanding inflicts subsumption query performance (Sect. 6) and has similar manageability problems of the consistency.

Our knowledge graph is an in-memory semantic property graph called KITT (Keep IoT Trivial). It combines the benefits of semantic understanding of RDF stores and the compact representation of property graphs. It uses an in-memory multigraph representation [2,16] of nodes and edges decorated with properties. Subsumption relationships are directly represented in-memory and allow for guaranteed consistency (R8) such that subconcepts, instances, or relationships cannot exist without the respective parent concepts or property types. The subsumption relationships can also be directly walked by the reasoner. The reasoner is a traversal-based RDFS+ reasoner that supports transitive queries (R6) alongside event-based reasoning (R7) so that whenever new data instances for a concept are added the reasoner will automatically execute relevant update queries. We use this to automatically instantiate e.g. analytic functions as discussed in the next section. Another unique aspect of the graph is its support for property inheritance (R3) that is used to propagate properties, like AI function configurations from concepts. It also supports element-level access control (R9) to homogenize access control across the applications. The graph does not provide a SPARQL interface as user often lack the required semantic knowledge (C7) and provides an easier to use structured YAML/JSON based query format.

The *application layer* contains the various solutions that utilize the Digital Twin. They consist of AI tools for machine learning and optimization that are automatically configured from the knowledge graph and of frontend user

interfaces. To simplify usage of the platform for domain experts (C7) we provide development frameworks in Java and Python that simplify querying and automation of tasks such as AI pipelines.

Through this architecture we support semantic retrieval of data streams which returns with a single call of the in-built reasoner the data alongside it's semantic context in the knowledge graph. For example, the user can ask for all `Power` data attached to a `Robot`, with its `Workorder` history and the last `Image` of a camera monitoring the robot. This query will pull the power consumption data from the IoT platform (Watson IoT®), the maintenance history from an asset management system (Maximo®) and images from an object store (COS®).

5 Digital Twin Benchmark Model

The requirement analysis in Sect. 3 identified the main requirements for designing a knowledge graph for Digital Twins. To test and compare different technologies for scaling knowledge graphs we created the Digital Twin Benchmark Model (DTBM) that shares the identified characteristics. Figure 2 shows the main elements of the model structure.

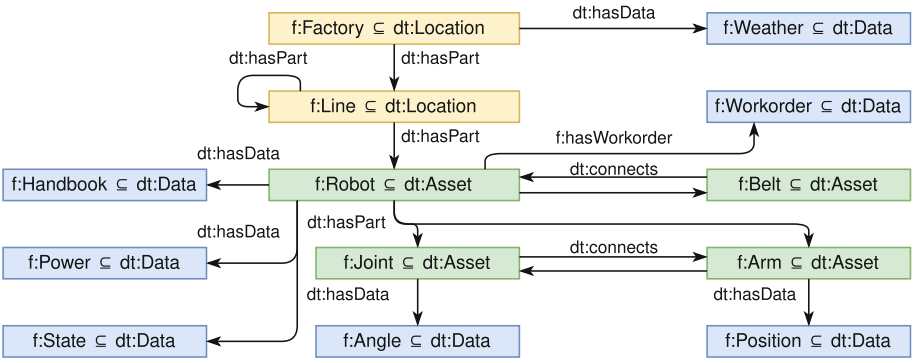

Fig. 2. Digital Twin benchmark model structure

The DTBM benchmark is representing an Industry 4.0 example and creates the knowledge graph for the Digital Twin of a production line. The RDF model is split in a core model (`dt:*`) and a domain taxonomy (`f:*`) according to requirement R1. The core model defines the trinity of `Locations`, `Asset`, and `Data` that is common with other IoT ontologies like BRICK [5]. To keep the benchmark self-contained we refrained from using any upper ontologies. The domain model represents of a location hierarchy in a `Factory` $\sqsubseteq$ `Location` with multiple production lines. The production `Lines` $\sqsubseteq$ `Location` may have a random depth of 1 to 3 levels to represent flexible hierarchies (C4).

Each production line has multiple `Robots` ⊑ `Asset` of different types as example for subsumption (R3) with different machine types. Each robot is composed off two to four `Joint` ⊑ `Asset` and `Arm` ⊑ `Asset` segments determining the degree of freedom to demonstrate composability (C6).

The robots are connected by `Belts` ⊑ `Asset` as example for a logistic transport system. It is to note, that the logistic system is introducing a horizontal element cutting across the hierarchical factory structure. These horizontal structures are very common and allow different views on the represented data.

For simplicity we stick to robots and belts in this example, but, they may be any machine and logistic element in a typical production process where a product passes trough a sequence of production steps.

Each robot has different data types attached (C2): a handbook (`Data_File` ⊑ `Data`), workorder data (`Json` ⊑ `Data`), a power meter (`Data_Series_Numeric` ⊑ `Data_Series` ⊑ `Data`), and on/off state (`Data_Series_Categoric` ⊑ `Data_Series` ⊑ `Data`) coming from different datastorage systems (C3). Each robot joint has angle measurements and each robot arm has a position measurement to illustrate heterogenity (C1). The belt has power meter and state as well to illustrate that data type may repeat across assets. The factory has weather data as example for a dataset at root level. All elements contain additional data properties to store meta-data such as asset ids or data keys linking to the data in the various backend systems.

The benchmark contains 12 queries. They resemble common type of queries that we see in applications run on the knowledge graph from querying information on specific assets to drive user interfaces and automatically configure and execute ML tasks on the data. The queries are heavily relying on subsumption (R2) and are using primarily generic concepts (`Locations`, `Asset`, `Data`). The queries also use transitive relationships (R6) like the `hasPart` relationship that is used to link the factory to lines, down to the robot, and its joints.

```
INSERT {
  ?newfunc rdf:type dt:Function .
  ?newfunc dt:hasInputData ?weather .
  ?newfunc dt:hasInputData ?input .
  ?newfunc dt:hasOutputData ?newout .
  ?newout rdf:type f:Data_Power_Pred .
  ?newout dt:hasDataKeySeries "TBD" .
} WHERE {
  ?loc rdf:type dt:Location .
  ?loc dt:hasSeries ?weather .
  ?weather rdf:type f:Data_Weather .
  ?loc dt:hasPart+ ?asset .
  ?asset rdf:type f:Asset .
  ?asset dt:hasSeries ?input .
  ?input rdf:type f:Data_Power .
  BIND(IRI(CONCAT(STR(?asset),"_Pred")) AS ?newout).
  BIND(IRI(CONCAT(STR(?asset),"_Func")) AS ?newfunc).
}
```

(a) Query 10 to configure an AI function

```
CONSTRUCT {
  ?func rdf:type ?functype .
  ?func dt:hasInputData ?input .
  ?input dt:hasDataKey ?input_key .
  ?func dt:hasOutputData ?output .
  ?output dt:hasDataKey ?output_key .
} WHERE {
  BIND (f:factory1_line_1_robot_1_Func AS ?func) .
  ?func rdf:type dt:Function .
  ?func rdf:type ?functype .
  ?func dt:hasInputData ?input .
  ?input dt:hasDataKey ?input_key .
  ?func dt:hasOutputData ?output .
  ?output dt:hasDataKey ?output_key .
}
```

(b) Query 12 to retrieve the sub-graph of an AI function configuration

Fig. 3. Query example from the Digital Twin benchmark model

Figure 3 shows two example queries from the benchmark. Query 10 in Fig. 3a uses SPARQL update to configures a new analytic AI function for each asset (Robot or Belt) that has an associated power timeseries. It uses this power data as input for the analytic function and also adds the weather data as additional input feature from the closes parent owning one (factory). It then assigns an "TBD" data key to the also newly created output timeseries that later will be replaced with a pointer to the newly computed data. The benchmark contains also other examples for configuring AI functions like: Q9 - to aggregate across a location hierarchy or Q11 - to aggregate all instances by asset type. All queries follow the same template of utilizing a graph pattern (WHERE) to identify deployment locations for the AI functions with relevant inputs and a creation (INSERT) part to materialize the respective AI function. Their configurations could be added to the (INSERT) section [27] or are in our case inherited from their type (R3).

Query 12 in Fig. 3b retrieves the sub-graph for one of these created configurations that contains the inputs and outputs for computing the prediction in an ML job including the data keys. In the given example, this would be a new AI function to compute the prediction of the power meter at robot 1 on line 1 that uses as input the power meter history and the weather data and the newly created output.

6 Experiments

The goal of our experimental evaluation is threefold. First, we want to illustrate the unique challenges that scaling knowledge graphs for digital twins create across various knowledge graph technologies. Second, we want to prove that the proposed Digital Twin Benchmark Model has different behaviour than other benchmarks to verify the need for another model and explore the different characteristics these benchmarks expect of the knowledge graph. Last, we want to validate the benefits of semantic knowledge graphs at the example of our implementation in comparison to established triple stores and property graphs across different benchmarks.

Knowledge Graphs: The first three knowledge graph technologies we evaluate are Blazegraph, GraphDB, and Virtuoso all of which are triple stores that expose SPARQL endpoints. The fourth system is Neo4j as most common property graph. The last one is KITT which is our semantic property graph. The selection is based purely on the availability of information on how to run BSBM and LUBM for these backends.

We used the same workflow to set the systems up, load the data into them, and evaluate their performance. Specifically, we used a containerization version for each graph without following any optimization strategy. The purpose is to replicate a cloud environment where a user will not have access to optimize the backend systems nor the knowledge (C7).

Benchmarks and Datasets: We present the results of two well known RDF benchmarks namely BSBM [7] and LUBM [14] as well as our proposed Digital Twin Benchmark Model (DTBM). Each of them generates RDF datasets which can easily be loaded into the triple stores. For Neo4J we use the neosemantics plugin to import RDF and had to write custom Cypher queries to support all subsumption alternatives. For KITT we use our own RDF loader hat converts RDF in a class and instance model. More details about the benchmarks:

1. The Berlin SPAQRL Benchmark (BSBM) [7] is built around an e-commerce use case and it is scaled via the number of the products. It creates a set of products offered by different vendors and reviewed by consumers. In general, the generated datasets are property heavy and consist of subsumption hierarchies and their depth depends on the chosen scale factor. For evaluating the different systems, BSBM uses a query generator which generates queries that emulate the search and navigation pattern of a consumer looking for a product.
2. The Lehigh University Benchmark (LUBM) [14] was developed to evaluate the performance of the semantic web knowledge base systems with respect to use in large OWL applications. It generates a dataset for the university domain and its query evaluation uses subsumptions and recursive queries such as transitive closure.
3. The Digital Twin Benchmark (DTBM) was introduced in the last section. It is subsumption heavy with transitive queries of flexible depth to query information and configure AI functions.

The characteristics of the datasets generated by the different benchmarks are summarized in Table 1. For the BSBM dataset the real "Dataset scale" factor is the value the reported value multiplied by 1,000. For example, S2 in BSBM refers to 2,000 products. BSBM is evaluated with 16 queries in parallel in Java, while LUBM and DTBM are evaluated with single-threaded sequential queries in Python. We do not run parallel tests to not skew results with Pythons bad parallelization behaviour.

Table 1. Dataset characteristics with their number of elements

Dataset	BSBM				LUBM				DTBM			
scale	triplets	nodes	edges	props	triplets	nodes	edges	props	triplets	nodes	edges	props
S2	377,241	36,871	102,315	37,022	288,894	38,349	113,463	36,834	114,177	35,702	41,698	35,601
S5	1,711,567	152,965	454,334	153,294	781,694	102,383	309,393	100,018	283,785	88,895	103,874	88,713
S10	3,738,188	315,148	982,502	315,477	1,591,643	207,441	630,753	203,625	570,232	178,118	208,224	177,801
S20	7,749,994	632,447	2,018,821	632,776	3,360,686	437,570	1,332,029	430,559	1,136,634	356,087	416,285	355,500
S50	17,571,059	1,593,382	5,132,438	1,594,113	8,317,905	1,082,833	3,298,813	1,067,023	2,844,499	890,624	1,041,300	889,227
S100	35,159,904	3,196,023	10,104,195	3,198,034	16,753,468	2,179,781	6,645,928	2,148,846	5,693,601	1,781,866	2,083,488	1,779,119

Query Configuration: In BSBM, we use the default BSBM test driver [7] which executes sequences of SPARQL queries over the SPARQL protocol against the system under test. In order to emulate a realistic workload, the test driver can simulate multiple clients that concurrently execute query mixes against the test

system. The queries are parameterized with random values from the benchmark dataset, in order to make it more difficult for the test system to apply caching techniques. The test driver executes a series of warm-up query mixes before the actual performance is measured in order to benchmark systems under normal working conditions. To make a fair comparison, we implemented our own test driver that follows a similar design to evaluate Neo4j and KITT. Note, BSBM test driver measures the response time excluding the time needed to deserialize the response and it puts a constraint to the result size.

In LUBM and DTBM, we use our own query executor in Python to evaluate the performance of the various systems. We execute each query 500 times and we report the average response time including the deserialization of the response. We also place a constraint to the result size limiting the result to 1,000 records to put the focus on the query execution time and not the serialization time.

Overall, all benchmarks use SPARQL queries; corresponding Cypher queries for Neo4j or YAML queries for KITT. We optimize the representation for each graph and e.g. directly represent properties in Neo4j and optimize queries accordingly to e.g. activate subsumption support (RDFS+) for all RDF stores. All benchmarks have been executed on a Linux machine with a AMD Ryzen 3959X processor and 64 GB memory and SSDs with the backends running on Docker.

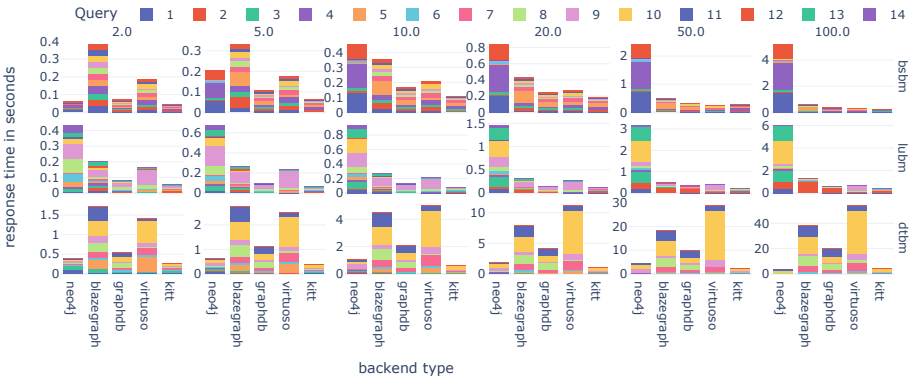

Fig. 4. Query response times for different sizes (columns) and benchmarks (rows)

6.1 BSBM Evaluation

Figure 4 summarizes the average response time for all benchmarks in seconds. We will first discuss the BSBM results in the first row. For this one all queries run in less than a second even for the big graphs. We observe that Neo4j performs well only for small graphs and worse for larger. It seems that it cannot understand the semantics hidden in the dataset and it does not perform well when queries use subsumption. Comparing the triple stores, we find that GraphDB has the lowest response times while Blazegraph has the worst performance. It seems that the queries using filtering by multiple properties such as Query 5 are costly for the

RDF triple stores. KITT outperforms the other knowledge graphs in all datasets slightly. This shows its ability as a semantic property graph to understand the model and the semantics contained in the dataset.

6.2 LUBM Evaluation

The results for LUBM on the second row in Fig. 4 show about the same characteristics as BSBM. However, we observe that Neo4j is not able to perform well for any size. This is due to the fact that LUBM uses many subsumptions and recursive queries and Neo4j seems unable to process them efficiently. Furthermore, we observe for the different graphs that the ratio some queries have in the stacked bars change with the experiment sizes. This hints that the knowledge graphs struggle with different queries characteristics when a graph grows. For example, Query 9 is the most expensive for triple stores up to dataset size S20, and then we see an increase in response time for Query 2. This can be explained by the fact that Query 2 fetches for each department of a university, all the graduate students and searches for those who have obtained their undergraduate degree from the given university. This breadth exploration becomes very costly in larger graphs because the number of students becomes huge and many invalid pathes need to be evaluated before they are dropped from the result set. In addition, Query 9 returns students who have taken a course of their faculty advisor which also leads to multiple traversals of the hierarchy that links advisors to students through a course. Although the query is of a similar structure to Query 2, it is less costly in large graphs due to the smaller search space. Finally, we observe that KITT again surpasses all knowledge graphs and it can produce the result in a few milliseconds. This again shows the benefits of an in-memory semantic property graph that can directly walk the subsumption taxonomy from high-level concepts to instances and their transitive relationships.

6.3 DTBM Evaluation

The results for our proposed benchmark illustrate first the bad scaling behaviour of traditional knowledge graphs in a Digital Twin scenario as discussed in the beginning. While common execution times for BSBM and LUBM were usually under one second for the triple stores they grow to multiple seconds for some queries for larger graphs, which is not acceptable for driving user interfaces. This magnitude differences in response times for DTBM is particularly notable as that the size in triples of the dataset size S100 is only 1/3 of LUBM and 1/6 of BSBM (Table 1).

The second notable aspect is that the performance characteristics between the triple stores change. While Blazegraph was performing worst for BSBM and LUBM it is now Virtuoso that has the largest response times. It is particularly struggling with Query 10 (Fig. 3a) and actually failed out-the-box for S100 due to too large implication space.

Along the same line it is now Neo4J that overtakes the triple stores in the benchmark. It probably benefits from the lower number of nodes and the property representation. It is notable that across all benchmarks it scales constantly along to the number of nodes. However, we have to note that we had to write custom Cypher queries for Neo4J to support all subsumption alternatives for all use cases. In production this is not feasible because taxonomies are domain specific (R1) and change (C8).

KITT outperforms the other graph stores also for this scenario. This demonstrates the generalizable benefit of a semantic property graph across multiple use case from RDF specific benchmarks like BSBM or LUBM to the digital twin use case addressed in DTBM.

7 Summary

In this paper we share our experience in scaling semantic knowledge graphs for automating AI of Digital Twins in the IoT domain. We analyze the unique characteristics, goals and requirements of the use case that uniquely links semantic models to multi-modal data management across different federated data systems.

We derive from this a reference architecture for a Digital Twin knowledge graph solution that we use in multiple products. It separates the knowledge graph from the underlying data in a micro-service environment that is easy to scale and extend.

To enable the community to evaluate different knowledge graph technologies for this architecture we open source a new Digital Twin Benchmark Model (DTBM) that represent the specific requirements of the domain. The DTBM generator creates an Industry 4.0 production line and contains multiple queries that we use in production. We demonstrate in some query examples how AI functions can be automatically configured from the semantic graph. We execute the benchmark for different knowledge graph technologies and compare it to the well established BSBM and LUBM. The result highlight the different behaviour of DTBM in comparison to BSBM and LUBM and substantiate the need of the new benchmark. They also illustrate the challenges in scaling knowledge graphs for Digital Twins that already show by a magnitude larger response times for even smaller graphs. It may be possible to optimize the parameters, indexes and queries for the different knowledge graph technologies, but, given the domain specificity (R1) and dynamic nature (C8) of these models and queries this is not feasible in production for a general purpose cloud service where users have neither access to these configurations nor the expertise (C7).

That our own knowledge graph KITT shows the best performance across all benchmarks demonstrates the advantage of semantic property graphs that combine benefits of RDF and property graphs and support subsumption and transitivity out of the box. We would like to see more graphs that address this need and not specialize on one or the other. For that purpose we open sourced the script for creating the Digital Twin Benchmark Model as well as the used benchmark configuration such that other can replicate and extend our results.

Supplemental Material Availability: The source code and queries for DTBM as well as the used configurations for the benchmark are available at https://github.com/IBM/digital-twin-benchmark-model.

References

1. Akroyd, J., Mosbach, S., Bhave, A., Kraft, M.: Universal digital twin-a dynamic knowledge graph. Data-Centric Engineering, vol. 2 (2021)
2. Ali, W., Saleem, M., Yao, B., Hogan, A., Ngomo, A.C.N.: A survey of RDF stores & SPARQL engines for querying knowledge graphs. VLDB J. **31**, 1–26 (2021)
3. Bader, S.R., Grangel-Gonzalez, I., Nanjappa, P., Vidal, M.E., Maleshkova, M.: A knowledge graph for industry 4.0. In: Extended Semantic Web Conference (ESWC), pp. 465–480 (2020)
4. Bader, S.R., Maleshkova, M., García-Castro, R., Davies, J., Antoniou, G., Fortuna, C.: Towards integrated data control for digital twins in industry 4.0. In: International Workshop on Semantic Digital Twins (SeDiT) at Extended Semantic Web Conference (ESWC) (2020)
5. Balaji, B., et al.: Brick: metadata schema for portable smart building applications. Appl. Energy **226**, 1273–1292 (2018)
6. Besta, M., et al.: Demystifying graph databases: analysis and taxonomy of data organization, system designs, and graph queries. arXiv preprint arXiv:1910.09017 (2019)
7. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. Semant. Web Inf. Syst. (IJSWIS) **5**(2), 1–24 (2009)
8. Bone, M., Blackburn, M., Kruse, B., Dzielski, J., Hagedorn, T., Grosse, I.: Toward an interoperability and integration framework to enable digital thread. Systems **6**(4), 46 (2018)
9. Chaves-Fraga, D., Priyatna, F., Cimmino, A., Toledo, J., Ruckhaus, E., Corcho, O.: GTFS-Madrid-Bench: a benchmark for virtual knowledge graph access in the transport domain. J. Web Semantics **65**, 100596 (2020)
10. Chevallier, Z., Finance, B., Boulakia, B.C.: A reference architecture for smart building digital twin. In: International Workshop on Semantic Digital Twins (SeDiT) at Extended Semantic Web Conference (ESWC) (2020)
11. Dibowski, H., Massa Gray, F.: Applying knowledge graphs as integrated semantic information model for the computerized engineering of building automation systems. In: Extended Semantic Web Conference (ESWC), pp. 616–631 (2020)
12. Endris, K.M., Rohde, P.D., Vidal, M.-E., Auer, S.: Ontario: federated query processing against a semantic data lake. In: Hartmann, S., Küng, J., Chakravarthy, S., Anderst-Kotsis, G., Tjoa, A.M., Khalil, I. (eds.) DEXA 2019. LNCS, vol. 11706, pp. 379–395. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-27615-7_29
13. Erling, O.: Virtuoso, a hybrid RDBMS/graph column store. IEEE Data Eng. Bull. **35**(1), 3–8 (2012)
14. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. J. Web Semant. **3**(2–3), 158–182 (2005)
15. Hasnain, A., et al.: BioFed: federated query processing over life sciences linked open data. J. Biomed. Semant. **8**(1), 1–19 (2017)
16. Ingalalli, V., Ienco, D., Poncelet, P., Villata, S.: Querying RDF data using a multigraph-based approach. In: Extending Database Technology (EDBT), pp. 245–256 (2016)

17. Kalaycı, E.G., Grangel González, I., Lösch, F., Xiao, G., Kharlamov, E., Calvanese, D., et al.: Semantic integration of Bosch manufacturing data using virtual knowledge graphs. In: International Semantic Web Conference (ISWC), pp. 464–481 (2020)
18. Kharlamov, E., Martin-Recuerda, F., Perry, B., Cameron, D., Fjellheim, R., Waaler, A.: Towards semantically enhanced digital twins. In: IEEE International Conference on Big Data, pp. 4189–4193 (2018)
19. Kumar, V.R.S., et al.: Ontologies for industry 4.0. Knowl. Eng. Rev. **34**, e17 (2019)
20. Li, X., Wang, L., Zhu, C., Liu, Z.: Framework for manufacturing-tasks semantic modelling and manufacturing-resource recommendation for digital twin shop-floor. J. Manuf. Syst. **58**, 281–292 (2021)
21. Lietaert, P., Meyers, B., Van Noten, J., Sips, J., Gadeyne, K.: Knowledge graphs in digital twins for AI in production. In: Advances in Production Management Systems (APMS), pp. 249–257 (08 2021)
22. Mihindukulasooriya, N., et al.: Knowledge graph induction enabling recommending and trend analysis: a corporate research community use case. In: International Semantic Web Conference (ISWC) (2022)
23. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.C.: DBpedia SPARQL benchmark-performance assessment with real queries on real data. In: International Semantic Web Conference (ISWC), pp. 454–469 (2011)
24. Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., Taylor, J.: Industry-scale knowledge graphs: lessons and challenges. Commun. ACM **62**(8), 36–43 (2019)
25. Pan, Z., Zhu, T., Liu, H., Ning, H.: A survey of RDF management technologies and benchmark datasets. J. Ambient Intell. Humanized Comput. **9**(5), 1693–1704 (2018)
26. Ploennigs, J., Cohn, J., Stanford-Clark, A.: The future of IoT. IEEE Internet Things Mag. **1**(1), 28–33 (2018)
27. Ploennigs, J., Schumann, A., Lécué, F.: Adapting semantic sensor networks for smart building diagnosis. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8797, pp. 308–323. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11915-1_20
28. Rahman, H., Hussain, M.I.: A comprehensive survey on semantic interoperability for internet of things: state-of-the-art and research challenges. Trans. Emerg. Telecom. Tech. **31**(12), e3902 (2020)
29. Rojas, J.A., et al.: Leveraging semantic technologies for digital interoperability in the European railway domain. In: International Semantic Web Conference (ISWC), pp. 648–664 (2021)
30. Samuelsen, S.D., Nikolov, N., Soylu, A., Roman, D.: An approach for representing and storing RDF data in multi-model databases. In: Research Conference on Metadata and Semantics Research, pp. 47–52 (2020)
31. Schmid, S., Henson, C., Tran, T.: Using knowledge graphs to search an enterprise data lake. In: Extended Semantic Web Conference (ESWC), pp. 262–266 (2019)
32. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: a benchmark suite for federated semantic data query processing. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 585–600. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_37
33. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^{2}Bench: a SPARQL performance benchmark. In: IEEE International Conference on Data Engineering, pp. 222–233 (2009)
34. Singh, V., Willcox, K.E.: Engineering design with digital thread. AIAA J. **56**(11), 4515–4528 (2018)

35. Taelman, R., Colpaert, P., Mannens, E., Verborgh, R.: Generating public transport data based on population distributions for RDF benchmarking. Semant. Web **10**(2), 305–328 (2019)
36. Zheng, X., Lu, J., Kiritsis, D.: The emergence of cognitive digital twin: vision, challenges and opportunities. Int. J. Prod. Res. 1–23 (2021)
37. Zhou, B., Pychynski, T., Reischl, M., Kharlamov, E., Mikut, R.: Machine learning with domain knowledge for predictive quality monitoring in resistance spot welding. J. Intell. Manuf. **33**(4), 1139–1163 (2022). https://doi.org/10.1007/s10845-021-01892-y