Structuring and Accessing Knowledge for Historical and Streaming Digital Twins

Bentley James Oakes[1,2][0000−0001−7558−1434], Bart Meyers[2][0000−0001−9566−8297], Dennis Janssens[2,3][0000−0003−0549−3775], and Hans Vangheluwe[1,2][0000−0003−2079−6643]

[1] AnSyMo Lab, University of Antwerp, Antwerp 2000, Belgium
{Bentley.Oakes,Hans.Vangheluwe}@uantwerpen.be
[2] Flanders Make vzw, Lommel 3920, Belgium Bart.Meyers@flandersmake.be
[3] DMMS Lab, Katholieke Universiteit Leuven, Leuven 3001, Belgium
Dennis.Janssens@kuleuven.be

Abstract. Organisations are intensely developing Digital Twins (DTs) to correctly and efficiently answer questions about the history and behaviour of physical systems. However, it is not clear how to construct these DTs starting from the data, information, knowledge, and wisdom available in the organisation. In this work, we present our approach to DT construction which involves a layered knowledge graph (KG) architecture communicating with the organisation's data repositories. We explain the components and timelines for using the KG to build both historical and streaming DTs, and what kinds of questions can be answered for our drivetrain use case.

Keywords: Digital Twins · Knowledge Graph · Knowledge Representation.

1 Introduction

The rise of the Internet of Things (IoT) and Industry 4.0 means that today's industries are faced with a deluge of data. Multitudes of data points (sales, sensor values, maintenance records, ...) are generated each second and poured into various data repositories, each with implicit connections and relations between organisational resources (processes, machines, environmental conditions, etc.). The lack of semantics and explicit relations can hinder actionable insights [2] such that a data scientist investigating the data spends considerable time to understand the context of the data before it can be used. This slows down the process of discovering trends, influencing new designs, using modelling results, and discovering past experiments and lessons learned.

For example, a simple query such as *what experiment produced these results?* can often become an laborious endeavour to answer. As a first step, one must find

the data and then understand the format and units, which may involve countless emails to find the one person in the organisation who understands the data. This inefficient process can consume the time of multiple people on coordination, which is further delayed if anyone involved has left the organisation.

Report Context This paper presents our current insights and vision developed as part of a joint academic-industrial project led by Flanders Make, the strategic research center for the Flemish manufacturing industry. The *Framework for Systematic Design of Digital Twins* (DTDesign) project brings together three universities and five industrial partners to focus on determining the most effective technologies and methods for how to build a *Digital Twin* (see Section 3) from a representation of an organisation's past and current data, information, knowledge, and wisdom (DIKW) [15]. The insights presented in this report are gained from our collaborations, discussions, and creation of initial prototypes. Throughout the lifetime of the project, these results are and will be iteratively applied and validated on the industrial use cases, beyond the academic drivetrain use case presented in Section 3.2.

Approach In the DTDesign approach, the representation of an organisation's DIKW is stored as a *knowledge graph* (KG) which "acquires and integrates information into an ontology and applies a reasoner to derive new knowledge"[5]. It contains relevant DIKW which is consistent and provides value to the organisation through its connections, and has been successfully applied in many domains [1, 4]. The KG is built, maintained, and queried by the user (possibly through applications). The results of these queries give insights into the system, or are used to build applications or code to run on the machines. For example, anomaly detection and optimisation algorithms are trained and executed using the input conditions of particular procedures [11].

While KGs and DTs are each relatively old concepts, only recently has there been work on their interactions, such as [16]. This is due to the increasing implementation of DTs within industry and academia, the rise of IoT gathering an enormous amount of data, and the computing resources offered by cheap servers locally or in the cloud. However, experience reports on the development of DTs and connected systems can lack crucial information [13], especially on the DT components and their developmental timelines. In this paper, we describe the processes of building up such a KG, building DTs from that KG, and their relationships to each other and the physical system under study. In particular, we discuss how the combination of KGs with the collection of industrial data at scale leads to effective DTs able to provide actionable answers to questions. An example timeline is also presented to illustrate the steps for building DTs from a KG.

Contributions and Structure This paper presents these specific contributions:

- Section 2: A high-level architecture relating a KG to DTs.
- Section 3: Definition of *historical* and *streaming* DTs, and a connection to the answers they provide for an example drivetrain use case.

– Section 4: A discussion of the structure and purpose of this KG including how to access, update, and compartmentalise the data/information/knowledge/wisdom within to serve large industrial organisations.
– Section 5: A presentation of an example timeline for creating DTs and their relations from a KG.

2 Conceptual Architecture

This section provides a brief overview of our approach to using a knowledge graph (KG) to assist with the construction of digital twins (DTs). As discussed in Section 3, these DTs are a virtual representation of a physical system.

Figure 1 shows the conceptual architecture of our solution. Our approach relates four main components: 1) the KG containing information, which rests on the data repositories of the organisation and acts as a *historical DT*, 2) the users and applications who access information from the KG and DT, 3) a physical system under study or control, and 4) the (streaming) DT of that system.

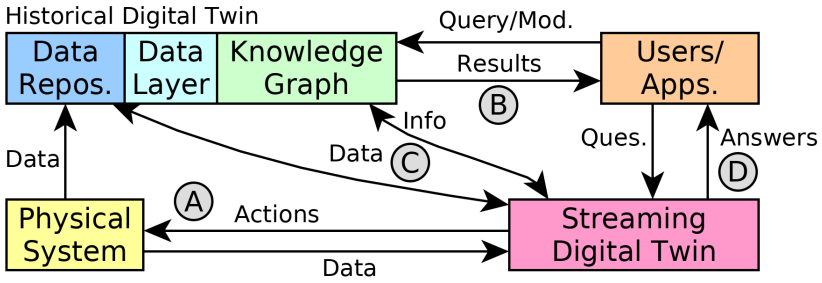

Fig. 1: Conceptual architecture of our proposed approach.

Four relations are also specified: A) the relation between the physical system and the DT, B) information added to or retrieved from the KG, C) the connection between the KG, data repositories, and the digital twin, and D) the questions and answers between the user and the DT. Of course, the users may also manipulate the physical system, but we omit this interaction from our architecture.

The remaining sections of this paper describe these components and relations in detail. Also note that Figure 1 does not represent the *time dimension* of the creation of the historical and streaming DTs. This is discussed in Section 5.

3 Running Example: Digital Twin(s) for a Drivetrain

This section provides background on digital twins (DTs) and introduces our notions of *historical* and *streaming* DTs. Our drivetrain use case is also briefly presented through a discussion of how these DTs can provide actionable answers about the drivetrain.

3.1 Digital Twin Background

As sensors and computing power become ever cheaper each year, it becomes easier to combine real-time data with high-fidelity models to create a virtual representation of a system to answer questions about that system. The umbrella term *digital twin* captures this relationship between a physical system and its digital counterpart [6], where "a DT is a virtual instance of a physical system (twin) that is continually updated with the latter's performance, maintenance, and health status data throughout the physical system's life-cycle" [12].

Kritzinger *et al.* classify the term *digital twin* into three categories depending on the relation between the physical object and the digital object [10]. This relation is labelled *A* in Figure 1. A *digital model* is where there is no automatic transfer of data between the digital and physical objects. A *digital shadow* is where data from the physical object is sent to the digital object automatically, as in a tracking simulator. Finally, a *digital twin* has an automatic connection both to and from the physical system. Further details about this separation and its application to DT experience reports are found in our previous work [13].

3.2 Addressing Drivetrain Questions

One of the use cases within the DTDesign project (see Section 1) is to represent and answer questions about an generic *drivetrain* representing drive-load interaction, which composes the components that deliver power from a motor to the wheels. In our conceptual architecture, these questions are answered by software/hardware components implementing a DT. Three question sets are presented here, as they showcase how the DTs built are at different scales and relate to the other components in Figure 1 differently.

The first question set is, "what is the correlation between motor current and the drivetrain torque, and which experiments have been conducted to investigate this?". The answer is therefore based on *historical data*, which implies that the DT is of the *history of the drivetrain*. We therefore term this a *historical digital twin* (Section 3.3).

The second question set is "What is the best sensor to add to my drivetrain to measure torque? Should this be a virtual sensor, or one relying on a non-intrusive sensor?". This question also relies on historical data for sensor validation as well as for physical modelling of the drivetrain components.

The third question set asks, "What is an algorithm to detect anomalies in the torque of the drivetrain, such that a warning signal can be raised?" This algorithm can be trained on historical data, but will be *deployed* into a *streaming digital twin* (Section 3.4) which is directly connected to the physical system.

3.3 Historical Digital Twin

In the *historical* DT case, the objective is to utilise the past knowledge of an organisation to determine which correlations are present in data, or to create and optimise algorithms to deploy in a *streaming* DT.

A traditional process may involve a data scientist mining experimental results and databases to locate such correlations. This can be a labour-intensive process if the information is not readily available, such as the data scientist having to contact a database owner to understand the meaning of the stored data.

In our approach, the historical DT created is a combination of software and hardware which is able to access the historical information available in the knowledge graph, or up-to-date information from the physical drivetrain system through the data lake. The data scientist can then ask relevant questions to the DT to obtain the required information in a structured and effective manner.

The historical DT is also relevant for questions where the user wishes to model and simulate the drivetrain itself in high-fidelity. As discussed in Section 3.2, a relevant question is whether a sensor can be found that is not too physically intrusive to be mounted on the drivetrain, or whether a virtual sensor must be created through combination of readings from other sensors. This relies on an accurate three-dimensional representation of the drivetrain geometry and of the considered sensors, such that the models can be combined with expert knowledge to understand if a particular sensor can be physically inserted into the drivetrain.

Note that in the historical DT case, the physical system does not need to be directly connected to the DT. Instead, it is the historical data that is of interest to the data scientist, and the DT represents the *history of the physical system* or the *product plus production* instead of the current behaviour.

3.4 Streaming Digital Twin

In contrast, the second DT type created for the drivetrain use case is focused on the actual "live" behaviour of the drivetrain itself. Data flows from the physical system to the streaming DT in near real-time. That is, the streaming DT is running alongside the physical system. For example, the drivetrain question about anomaly detection requires a constantly updated stream of data being fed as input into a device running alongside the drivetrain. This anomaly detection model must accurately represent the behaviour of the drivetrain such that all relevant anomalies are detected with few false positives. A future enhancement for this use case is also to directly control the motors on the drivetrain using the streaming DT, such that anomalies are handled appropriately in real-time.

This type of DT focuses more on the relationship between the digital twin and the physical system. Thus the streaming DT is closer to those DT envisioned for optimisation and control of a system, where the DT has an automatic connection to the physical system (see [10, 13]). Note that this implies more technical constraints, such as Internet of Things (IoT) issues (power consumption, real-time constraints, local computing, etc.) and intrusivity of sensors.

3.5 Digital Twin Classifications

Note there may not be a hard division between *historical* and *streaming* for a DT, as both relate to information about a physical system. However, we state that in a historical DT, the "twinning" of the DT focuses on the past data

and organisation's knowledge about a product or process. In other words, the physical system being represented is the *past history of the object in question, and the organisation's knowledge about it.* In the drivetrain example, this historical DT focuses on the correlation between the current provided to the motors and the resulting drivetrain torque, offering answers to questions about *properties*, *correlations*, and *experiments*. As such, more information from the KG may be accessed or present within the historical DT. Indeed, a portion of the KG may be deployed as the historical DT itself. In contrast, a *streaming* DT is more focused on the "live" information from the physical system itself in (semi-) real-time.

Section 5 also discusses how a DT may change classification between historical and streaming through its development, as the relevant questions change and data from the physical system becomes accessible.

Finally, we wish to emphasise that this classification of *historical* and *streaming* DTs is (mostly) orthogonal to the classification of Kritzinger *et al.* [10] of *digital model/shadow/twin*. Recall that Kritzinger *et al.* makes a classification on whether the connection between the physical and virtual systems are manual or automatic. That is, whether the DT can automatically sense or actuate the physical system. In contrast, our classification is based on whether a DT answers questions based on data and information from the physical system's *past* (including development information), or on the physical system's *current* state.

Therefore, it is possible to have any combination of *historical/streaming* and *digital model/shadow/twin*, except for a *streaming digital model*.

That is, a DT could be created focusing on historical data which automatically (or not) receives data and automatically (or not) controls the physical system. For example, a historical DT may be constructed to periodically examine the experimental data collected and automatically perform new experiments on the physical system to better understand the behaviour of the system.

As a streaming DT is concerned with the current behaviour of the system, it must be able to automatically access the state of the physical system. Thus, a streaming DT must be a *digital shadow* or *digital twin* as defined by Kritzinger *et al.* [10], depending on whether there is the possibility to perform automatic actions on the physical system.

4 Knowledge Graph

This section will discuss the *knowledge graph* in our approach. The concepts of information hierarchy, heterogeneity, and compartmentalisation will be discussed. The relation between the user and the knowledge graph as displayed in Figure 1 is also detailed.

4.1 Knowledge Graph Motivation

As mentioned in Section 1 there is a deluge of heterogeneous data present in today's large organisations. There has been significant work in the past on *understanding data* through the use of *ontologies* and *semantic reasoning* to provide

context to this data [3]. This allows data to be augmented with and transformed into *information*, *knowledge*, and *wisdom* (DIKW) [15], in a (semi-) automated process [14]. This can be visualised as a pyramid, with a wide base of data, then a layer of information, then knowledge, and a peak of wisdom. This represents how it becomes more difficult to extract meaning from the lower layers, and how much more data there will be in an organisation as compared to the forward-facing wisdom. As a rough mapping of this hierarchy to an industrial context, *data* is kept in databases, *information* is the linking between elements in a database, *knowledge* is the meta-data (data about data), and *wisdom* is any planning which takes into account the other levels.

One approach to storing this DIKW is with a *knowledge graph*, which stores this information as typed and attributed nodes with typed and attributed edges between them. The regular structure is well-suited to query approaches, and a great deal of flexibility is provided to the organisation to store DIKW their way.

In particular, the knowledge graph should conceptually support storage of all relevant DIKW from the organisation, and the connections between. For example, the knowledge graph may contain formalisation of the assets, experiments, and conclusions performed by an organisation. This allows for its reuse in current and future activities.

Once this knowledge graph is (partially) filled, then it can start to answer meaningful questions about the people, processes, and things present in the organisation. An connection to data lakes expands the DIKW and connections available. New DIKW must also be added over time to support new uses, in an incremental and consistent manner.

4.2 Heterogeneity

One challenge of combining a knowledge graph with industrial digital twins is that the knowledge graph must handle DIKW which is highly heterogeneous. For example, the Reference Architectural Model Industrie 4.0 (RAMI 4.0) [7] contains six distinct layers: *business*, *functional*, *information*, *communication*, *integration*, and *asset*. The knowledge graph could contain DIKW at all six of these layers and allow for answering questions that span multiple layers.

The DIKW within or referred to by the knowledge graph is also highly heterogeneous. For example, there may be time series, relational information, graph-based networks, images, documents, and any other information found in a large organisation. Another industrially relevant division is whether a piece of DIKW details the *structure* of an object, or instead details the *behaviour* of that object. For example, an architectural decomposition of a drivetrain refers to the mechanical components, while the behaviour is given through simulations or experimental results.

4.3 Type Hierarchies

Conceptually, the knowledge graph contains all DIKW and connections in an organisation. As this is not feasible, it must be possible for the knowledge graph

to be easily modified and evolved, and for connections between concepts to be very flexible. As the knowledge graph grows in richness, the number of concepts contained must also expand for flexibility and increased expressiveness.

In our knowledge graph approach, we offer the user a *typing system* that defines attributes and relationships for concepts. This allows the user to specify their own *types* and *instances*, and to form typing relationships within the knowledge graph. An example subset of a knowledge graph is pictured in Figure 2. The upper box of the figure shows the types, while the lower box displays the instances. Note the vertical typing relations between these two boxes which provide the typing information for the instances.

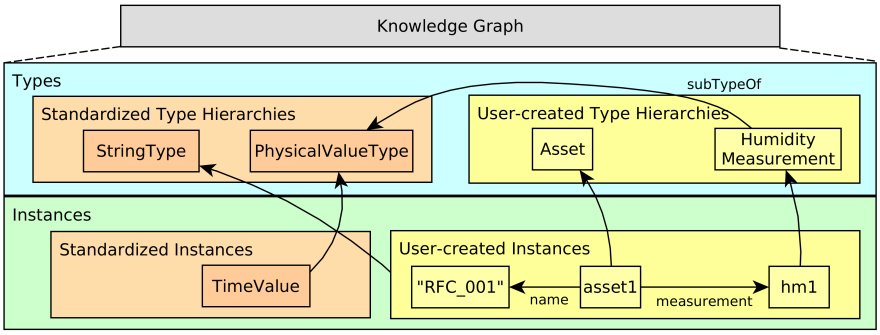

Fig. 2: Example types, instances, and connections within a Knowledge Graph.

However, it is also important to ensure consistency within the knowledge graph and to promote reusability and inter-operation with other sources of information. Therefore, the use of standardised concepts is suggested wherever possible. Also present in Figure 2 is the use of standardised and user-created types. In our approach, we define these explicit *type hierarchies* to provide a consistent and unified way to access and reason about the information contained.

In our approach, the knowledge graph is divided into built-in/standardised types and instances, and user-defined ones. This encourages levels of consensus, where the upper levels have (potentially) an international consensus, and the user can define organisation-wide types and instances as required.

For example, the current prototype defines built-in types similar to those defined in the Sensor-Observation-Sampling-Actuator ontology (SOSA) [9]. This ontology is a W3.org standard suitable to representing sensors, observations, and experiments in a cyber-physical system. Selecting a subset of this ontology allows the user to exploit the person-hours taken by others to construct a consistent and expressive ontology, while retaining the flexibility to define their own types.

Figure 3 shows an example of type hierarchies implemented within our prototype knowledge graph. The upper levels are type hierarchies which form the structural backbone of knowledge graphs. The lower levels are type hierarchies in the knowledge graph prototype representing useful formalisms. These hierar-

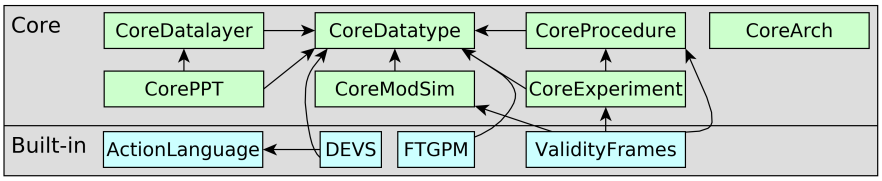

Fig. 3: Layered type hierarchies.

chies are not fixed, but instead represent that layered approach needed to have a consensus-based yet flexible type system. Of course, the user is able (and encouraged) to develop their own layers beneath (or above, or alongside...) those layers shown here to ensure flexibility and relevance to the concepts required by the organisations.

For example, the *ValidityFrames* type hierarchy in Figure 3 includes types relating to our previous research in defining the conditions and degree in which a model is valid with respect to a real-world system [17, 18]. This could be the temperature in which a model is valid (such as it is invalid below freezing), or define the set of training data for a model (such that a neural net has only been trained on daytime images and not night-time images). A type *VFModel* is defined which denotes the model of interest. As this model should also be contextualised through connection to other modelling and simulation concepts, the *ValidityFrames* type hierarchy imports the *Modelling and Simulation* type hierarchy (*CoreModSim*), and the *VFModel* type is denoted as a sub-type of the *CoreModel* type in that type hierarchy.

4.4 Data Layer

For practical reasons, in our approach the knowledge graph cannot contain all DIKW. This is due to the massive amounts of data incoming from assets in the organisation, which could quickly overwhelm the storage, reasoning, or querying capacities of the knowledge graph. Instead, our approach proposes a dedicated data layer between the knowledge graph and the underlying data stores. Thus this knowledge graph could be termed as a 'virtual' knowledge graph.

Figure 4 shows our layered knowledge graph approach. At the bottom are the sources of raw data, such the sensors of machines and organisation databases. Unique identifiers stored in the knowledge graph point to these sources of information. For example, the location of a data series may be stored in the knowledge graph, but not the individual data points themselves. The adapter layer is then able to resolve both the element type and the unique identifier to determine how and where to access the underlying data.

The knowledge graph requires neutral access to these data sources through a *data adapter layer*. This layer is intended to shield the knowledge graph from knowing the particulars of the data storage, which is quite technical and subject to constant revision. Examples of this include the database API, SQL queries,

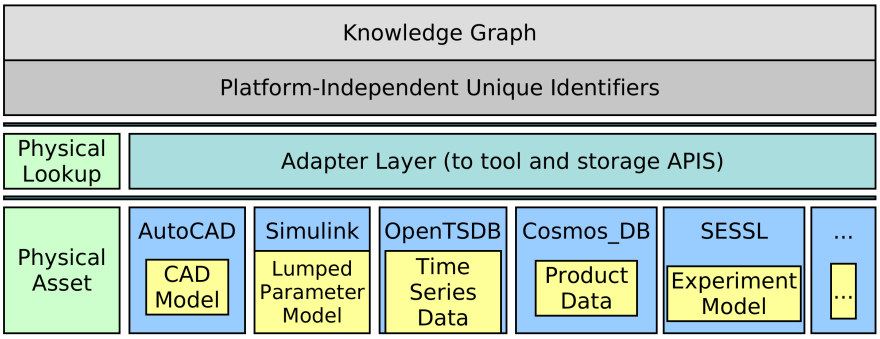

Fig. 4: The layered approach of the knowledge graph upon the data layer.

or OPC-UA (https://opcfoundation.org) commands necessary to access information, or the login information for repositories. Separating these more technical details out from the knowledge graph allows for greater scalability through the removal of transitory information and better handling of heterogeneous machine configurations. Thus, the data layer forms the abstracting barrier between the virtual knowledge graph (dealing with reasoning concerns) and the technical data sources (dealing with scalability and storage concerns).

For example, the knowledge graph may indicate that humidity information for a particular machine is stored as a *HumidityMeasurement* instance stored as a *TimeSeries* at a particular location. Typing this information allows for the knowledge graph to indicate constraints on the data, and precisely specify the type of information stored, while leaving the actual values to be stored in an appropriate way until explicitly requested by a user or application. This *TimeSeries* can be explicitly linked to the *Humidity* data type, allowing for a user to directly query, "where is humidity information for this machine?". Further information and knowledge can also be inferred with further connections, to indicate the experimental conditions a *TimeSeries* was produced from or the *Observation* procedure used.

4.5 Accessing the Knowledge Graph

In the overview shown in Figure 1, there is a component which represents the user accessing the knowledge graph, or an application on their behalf. This application could be a dashboard, reports, or any other visualisation. Once the information is placed in the knowledge graph, there must be an easy-to-use method for users and applications to extract this information. An example could be to ask for the set of all physical quantities in the knowledge graph, or which data stores contain temperature information.

As with any API, there are many challenges to provide a scalable, expressive query interface to the knowledge graph. Currently, we consider the API specification language GraphQL (https://graphql.org) to be a feasible approach. This query language takes queries defined in a JSON-like format, and evaluates

the fields requested using defined *resolvers* [8]. This approach again shields the user from understanding the underlying technology used to store the knowledge graph.

5 Constructing Digital Twins

This section will discuss the contribution of this paper concerning the construction of digital twins (DTs) using a knowledge graph (KG). In particular, an example timeline is shown noting how the historical and streaming DTs are created from the KG, and in connection with the KG.

Figure 1 relates the different components of our approach. However, the *time dimension* is not represented in that figure and will instead be discussed here. In particular, this section deals with relation C in Figure 1.

A portion of our current research is to create a workflow for how the KG, physical system, and DTs could be created either sequentially or in parallel. However, for brevity, this work will instead present an example timeline, showing one possible path for creation of all components. Figure 5 demonstrates this timeline as the construction of a KG, and then different types of DTs being built from it. In this figure, time increases to the right and the arrows between boxes represent a manual or automatic transfer of DIKW between the KG and the DTs.

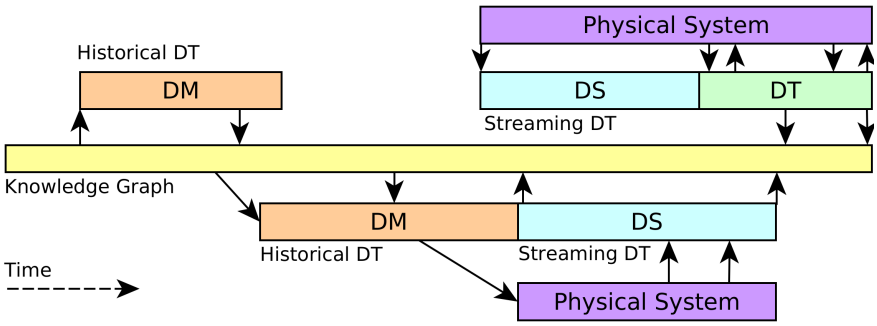

Fig. 5: Example timeline for Digital Twin creation.

5.1 Building the Knowledge Graph

The first step in the construction of the DTs is the building up of the KG itself. In Figure 5, this happens before any digital twins are built, where the user imports data, information, knowledge, and wisdom (DIKW) from existing repositories and creates types and instances. Conceptually, this KG exists throughout the lifetime of the organisation. As DTs are built, they are built using the knowledge in the KG, as well as connect to the KG to deposit new DIKW.

5.2 Digital Twin Types

Figure 5 has boxes representing the DTs which contain 'DM', 'DS', and 'DT'. These refer to the *digital model/shadow/twin* classifications from Kritzinger *et al.* which classify based on the manual or automatic connection with the physical system [10]. Viewed as a timeline, it is clear that the type of DT can change over time. In particular, it is likely that each DT is 'promoted' with more automation, as this allows for more insight and control of the physical system.

In particular, digital models are relevant for questions such as in design-space exploration [17]. In the lower-right part of Figure 5 a historical DT is a DM and is used to create the physical system. DSs can be used for anomaly detection and correlation questions, as found in the drivetrain use case. Finally, DTs can be used for adaptation and real-time control.

5.3 Building the Digital Twins

Once the KG is created, a few different paths exist for the creation of the DTs. In the lower-right of Figure 5, a historical DT is used to gather DIKW about the system, such as designing the most effective sensor for the drivetrain use case. Then, a *deployment* occurs, where the models and sensors are deployed onto hardware and connected to the physical system.

On the top-left of Figure 5, a DM is created from the KG to gain insights about the system. Insights are then relayed back to the KG, with the communication interleaved with the creation of the DT on the bottom-right. This represents how the queries to the KG are "snapshots" in time, and how the KG evolves as information comes back from the DTs.

Finally, in the top-right of Figure 5, a DS is created which is in connection with the physical system, and is not reliant on information from the KG. It is then promoted to a DT which exports DIKW to the KG.

6 Conclusion

This paper has presented our insights on the issue of combining a knowledge graph (KG) layered upon an organisation's data repositories with the process of constructing Digital Twins (DTs). In particular, the components of this approach are seen in Figure 1, with an example timeline in Figure 5. These DTs, separated into *historical* and *streaming*, allow answering different types of queries such as about correlations, optimisations, and anomaly detection.

Our future research involves clarifying the possible workflows for creating these DTs, their possible architectures, and a classification of their features.

Acknowledgments

This research was supported by Flanders Make, the strategic research center for the Flemish manufacturing industry, and partially funded by the DTDesign ICON (Flanders Innovation & Entrepreneurship FM/ICON::HBC.2019.0079) project.

References

1. Abu-Salih, B.: Domain-specific knowledge graphs: A survey. Journal of Network and Computer Applications **185**, 103076 (2021)
2. Banerjee, A., Dalal, R., et al.: Generating digital twin models using knowledge graphs for industrial production lines. In: Workshop on Industrial Knowledge Graphs (2017)
3. Buchgeher, G., Gabauer, D., et al.: Knowledge graphs in manufacturing and production: A systematic literature review. IEEE Access **9**, 55537–55554 (2021)
4. Chukkapalli, S.S.L., Mittal, S., et al.: Ontologies and artificial intelligence systems for the cooperative smart farming ecosystem. IEEE Access **8**, 164045–164064 (2020)
5. Ehrlinger, L., Wöß, W.: Towards a definition of knowledge graphs. SEMANTiCS (Posters, Demos, SuCCESS) **48**, 1–4 (2016)
6. Grieves, M., Vickers, J.: Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In: Transdisciplinary perspectives on complex systems, pp. 85–113. Springer (aug 2017)
7. Hankel, M., Rexroth, B.: The Reference Architectural Model Industrie 4.0. ZVEI: Die Elektroindustrie (2015)
8. Hartig, O., Pérez, J.: Semantics and complexity of GraphQL. In: Proceedings of the 2018 World Wide Web Conference. pp. 1155–1164 (2018)
9. Janowicz, K., Haller, A., et al.: SOSA: A lightweight ontology for sensors, observations, samples, and actuators. Journal of Web Semantics **56**, 1–10 (2019)
10. Kritzinger, W., Karner, M., et al.: Digital twin in manufacturing: A categorical literature review and classification. IFAC-PapersOnLine **51**(11), 1016–1022 (2018). https://doi.org/10.1016/j.ifacol.2018.08.474
11. Li, X., Wang, L., et al.: Framework for manufacturing-tasks semantic modelling and manufacturing-resource recommendation for digital twin shop-floor. Journal of Manufacturing Systems **58**, 281–292 (2021)
12. Madni, A.M., Madni, C.C., Lucero, S.D.: Leveraging digital twin technology in model-based systems engineering. Systems **7**(1), 7 (jan 2019). https://doi.org/10.3390/systems7010007
13. Oakes, B.J., Parsai, A., et al.: Improving digital twin experience reports. In: Proceedings of the 9th International Conference MODELSWARD. pp. 179–190. INSTICC, SciTePress (2021). https://doi.org/10.5220/0010236101790190
14. Opdahl, A.L., Tessem, B.: Ontologies for finding journalistic angles. Software and Systems Modeling **20**(1), 71–87 (2021)
15. Rowley, J.: The wisdom hierarchy: representations of the DIKW hierarchy. Journal of Information Science **33**(2), 163–180 (2007)
16. Rozanec, J.M., Jinzhi, L.: Towards actionable cognitive digital twins for manufacturing. In: Second International Workshop on Semantic Digital Twins (2020)
17. Van Acker, B., Oakes, B.J., et al.: Validity frame concept as effort-cutting technique within the verification and validation of complex cyber-physical systems. In: Proceedings of the 23rd ACM/IEEE International Conference MODELS-C. pp. 1–10 (2020)
18. Van Mierlo, S., Oakes, B.J., et al.: Exploring validity frames in practice. In: International Conference on Systems Modelling and Management. pp. 131–148. Springer (2020)