CIRP Annals - Manufacturing Technology 00 (2020) 1–4

Contents lists available at ScienceDirect

CIRP Annals - Manufacturing Technology

journal homepage: https://www.editorialmanager.com/CIRP/default.aspx

A design framework for adaptive digital twins

John Ahmet Erkoyuncu[a,*], Iñigo Fernández del Amo[a], Dedy Ariansyah[a], Dominik Bulka[a], Rok Vrabič (2)[b], Rajkumar Roy (1)[c]

[a] School of Aerospace, Transport and Manufacturing, Cranfield University, United Kingdom
[b] Faculty of Mechanical Engineering, University of Ljubljana, Ljubljana, Slovenia
[c] School of Mathematics, Computer Science and Engineering, City, University of London, United Kingdom

ARTICLE INFO

Article history:
Available online xxx

Keywords:
Digital twins
Design method
Ontology

ABSTRACT

Digital Twin (DT) is a 'living' entity that offers potential with monitoring and improving functionality of interconnected complex engineering systems (CESs). However, lack of approaches for adaptively connecting the existing brownfield systems and their data limits the use of DTs. This paper develops a new DT design framework that uses ontologies to enable co-evolution with the CES by capturing data in terms of variety, velocity, and volume across the asset life-cycle. The framework has been tested successfully on a helicopter gearbox demonstrator and a mobile robotic system across their life cycles, illustrating DT adaptiveness without the data architecture needing to be modified.

1. Introduction and state of the art

The digital twin (DT) is a digital representation of a physical asset that can be used to describe its properties, condition, and behaviour through modelling, analysis, and simulation [1]. The vision for DTs is to reach a fully autonomous system that can collect information about its operation (e.g. manufacturing) and simulate its effects; so the DT can support any decision to be made about the asset [2]. There have been a range of applications of DTs. Bilberg and Malik [3] present a DT driven human-robot assembly system. Tao et al. [4] offer DT driven health management for wind turbines. Stark et al. [5] use the test environment of smart factory cells to research key factors to develop and operate DTs. Schleich et al. [6] describe a reference model based on the concept of Skin Model Shapes for DTs.

A DT captures design, manufacturing and operational information, aiming to represent with minimal differences the actual asset. This contributes to various types of decision making [2]. DTs heavily rely on collecting and storing vast life-cycle related data about products using IoT technologies. This is often enabled through PLM or ERP platforms, which can cover the entire product life-cycle [7]. However, structuring the data and enabling its use across the asset life-cycle is still a major challenge, where ontologies are attracting interest. According to Zhu et al. [8], ontologies have been widely used in context modelling, as they are independent of programming languages and enable context reasoning. Erkoyuncu et al. [9] present adaptive operational support using a context aware Augmented Reality (AR) technique. Stark et al. [10] use ontology in the context of PLM to improve sustainable product development. Abramovici et al. [7] offer a semantic data management platform for the development and continuous reconfiguration of smart products and systems. There is also a rich literature in computational ontologies, which cover Description Logic used broadly for defining ontologies such as the Web Ontology Language [11].

Stark et al. [5] highlight the need for further development in the linkage between existing DTs' brownfield systems and their data. Tomiyama et al. [2] state that the feedback mechanism, particularly the one from the as-is model to the product model, is not yet sufficiently understood. Bilberg and Malik [3] also raise future work needs in seamless integration of different modules with minimum manual intervention. Even though DTs are used in different contexts, limited research evaluates how a DT and its architecture can accommodate changes occurred to the asset during its life-cycle. Lack of approaches for adaptiveness is one of the main reasons preventing industrial adoption of DTs.

Software integration in DTs is currently achieved using standard formats for data exchange [5]. This paper argues that this may not be feasible for evolving DTs, which require software changes due to asset modifications across its life-cycle. Hence, several research challenges remain to be addressed: (1) DT comprises multiple models of varying granularity, describing various aspects across the life-cycle, (2) the models need to be interlinked and synchronised with the asset, and (3) presents challenges in data and models in terms how we manage big data in DTs in terms of variety, volume, and velocity. This paper contributes with an ontology-based approach for designing DT data architectures to semantically link data and models to represent an asset. The research question is: How to design a DT data architecture adaptable to changes in its software and the asset over its life-cycle?

* Corresponding author.
E-mail address: j.a.erkoyuncu@cranfield.ac.uk (J.A. Erkoyuncu).

https://doi.org/10.1016/j.cirp.2020.04.086

2. System overview for adaptive DTs

An asset evolves over its life-cycle through modifications. These can be due to changes of its systems/components (e.g. retrofitting), or modifications in the processes that manage the asset (e.g. monitoring). From a DT perspective, these changes have various implications: (1) data representing the asset should reflect the modifications made, (2) software using such data should be able to manage the changes in data, and (3) new software included in the DT over the asset evolution should be able to manage existing data and add new. The consequences of the abovementioned implications are varied for distinct DT data management options. Decentralised approaches (Fig. 1.a) include data repositories for each software system. Hence, they are more redundant and secure but require much more effort to spread changes. Instead, centralised approaches (Fig. 1.b) are better at managing data changes, though they can be less redundant. They use a central repository to manage and serve data to each software on demand. Centralised repositories require less effort when a new software is introduced (only one interface to be created). Although, if a new software implies changes to the repository's data architecture (e.g. a new column in an existing relational database), then it would still require all existing interfaces to be modified. That is solved by shared language approaches (Fig. 1.c) that also maintain centralisation advantages.

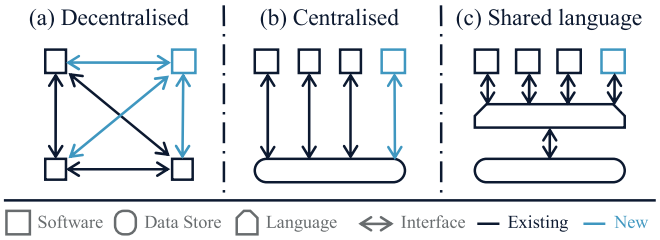

Fig. 1. DT data architures: Decentralised, centralised and ontology-based.

Compared to other shared languages (e.g. structured query language - SQL), ontologies offer additional advantages. Firstly, they organise data semantically according to knowledge domains (KD). So that DT data can be shared with the same meaning for both software systems and users. Secondly, ontologies are based on the 'open-world' assumption. Since data not declared is not implied to not exist, data changes can spread easier as interfaces are prepared to receive new data schemas. Thirdly, ontologies can provide inferencing capabilities to the data stored, offering additional capabilities to reason over existing data. These inferencing capabilities can be integrated through the use of interfaces between different DT software. Such interfaces can be either generic (e.g. standard inferencing engines) or ad-hoc for specific issues (e.g. high-frequency big data storages).

3. Ontology based design framework for adaptive DTs

The design framework determines the steps to generate or update ontologies, so that diverse DT software can communicate with each other using a common language. Such language should describe every KD to be considered in an asset's life-cycle. Besides, it should also include unique references for every object to be part of an asset. These are why this design framework consists of two main stages (Fig. 2). The first one, only to be conducted once, enables to uniquely declare any possible asset's part, including those to be modified over the asset's life-cycle. The second stage enables to generate interfaces for each new DT software to be incorporated. These may include new information regarding existing or new assets' life-cycle KD. This stage is where the design change requirements are captured, and a suitable DT data architecture is developed. In order to quantitatively evaluate modifications of ontology-based data architectures, the following definitions are used: (1) variety (number of attributes and/or relationships to modify/update), (2) volume (number of individuals to modify or update when assigning new attributes/relations), and (3) velocity (number of interfaces to update due to asset's evolution).

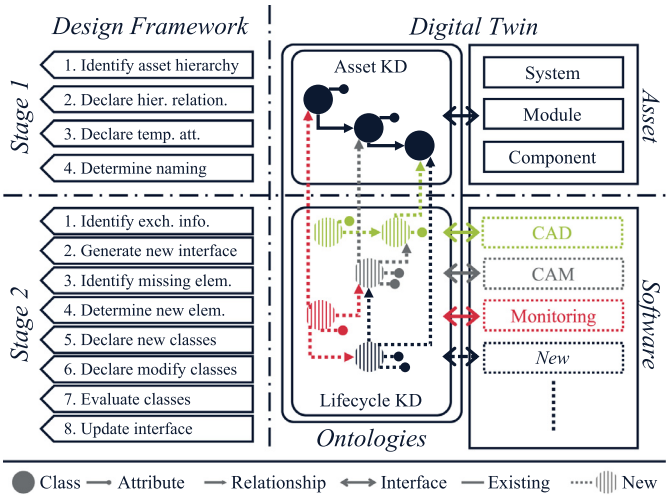

Fig. 2. Ontology Design Framework: Ontologies in asset KDs over time.

3.1. Stage 1: asset description for adaptive DT

In order for diverse DT software to exchange data consistently, it is necessary to declare unique references for every possible object (e.g. component or sensor) that is part of an asset. So, the changes experienced (e.g. replacement) over the asset's life-cycle can be tracked. This stage includes the following steps: **Step 1:** Identify all possible hierarchical levels the asset can consist of and declare these as classes. **Step 2:** Declare all necessary hierarchical relationships to correlate these classes. **Step 3:** Declare all necessary attributes to identify assets' parts temporality (e.g. disposal). **Step 4:** Determine a standard to name every object, so they can have unique references (e.g. spare part). Although, the ontology elements may be subject to change, the number of attributes and relationships assigned for each class should not vary. If so, every time an asset's part is modified, any existing or new DT software can keep track of it. Consequently, additional information regarding an asset and its parts should be allocated in separate ontologies as described in *Stage 2*.

3.2. Stage 2: designing dynamic behaviour of DT

In *Stage 2*, the focus is on offering design flexibility and choices to introduce adaptiveness in DT data architecture deriving from changes in the asset. In order to maintain the ontologies' 'open-world' assumption, there are two aspects of ontology-based data to consider: temporality and hierarchy. A **temporal** attribute/relationship is such that includes temporal information regarding the asset (e.g. manufacturing date). A **hierarchical** attribute/relationship is such that refers to a specific asset's element category (e.g. component or system) as declared in *Stage 1*. If every attribute or relationship that is temporal or hierarchical is maintained in a separate ontology class, then these classes can be kept separated to each other while maintaining their semantic connections. So, different interfaces can treat ontology classes differently while maintaining data consistency. The following steps assume these aspects to explain a method to generate mutable ontologies, so data can be shared and updated consistently over the asset's life. These steps are triggered every time a DT software is introduced or renewed:

Please cite this article as: J.A. Erkoyuncu et al., A design framework for adaptive digital twins, CIRP Annals - Manufacturing Technology (2020), https://doi.org/10.1016/j.cirp.2020.04.086

Step 1: Declare new/updated software's exchangeable data as ontology attributes and relationships. This captures the design requirement for digital twin adaptiveness based on the data to be exchanged. **Step 2:** If they are all declared in the existing ontologies, generate a new interface straightaway. Otherwise continue to the following step. **Step 3:** Identify all 'missing' attributes and relationships from the existing ontologies and list the classes containing those that are not missing. **Step 4:** If all attributes and relationships are 'missing', then jump to Step 6. Otherwise, continue to Step 5. **Step 5:** For each 'missing' attribute and/or relationship: ***Step 5.1:*** List all possible classes from Step 3 to which the 'missing' attribute/relationship can be assigned to. ***Step 5.2:*** If the 'missing' attribute/relationship is temporal, then generate a new class, assign it the new class, and a relation to the classes listed in Step 5.1. ***Step 5.3:*** If the 'missing' attribute/relationship is hierarchical, then assign it to all the possible classes listed in Step 5.1. If these classes already include hierarchical attributes or relationships, duplicate the classes by splitting them. ***Step 5.4:*** If the 'missing' attribute/relationship is none of the above, assign it to all the possible classes listed in Step 5.1. ***Step 5.5:*** If as a result of previous steps there is more than one potential class to be declared/modified, then continue to Step 7. Otherwise continue to Step 8. **Step 6:** For all 'missing' attributes and/or relationships: ***Step 6.1:*** Create new classes by grouping attributes and relationships so each temporal and hierarchical aspect is separated. ***Step 6.2:*** Ensure each new class includes a relationship to a class from the hierarchical ontology schema (*Stage 1*). ***Step 6.3:*** Ensure that each new class whose individuals can be modified without modifying its reference to the hierarchical individual (*Stage 1*) have a temporal attribute or relationship. ***Step 6.4:*** If as a result of previous steps there is more than one potential class to be declared, then continue to Step 7. Otherwise continue to Step 8. **Step 7:** Existing options of class declarations should be evaluated based on their impact to the data architecture: ***Step 7.1:*** For each new/modified class declaration, calculate its changes in variety, velocity and volume. ***Step 7.2:*** Select the class declaration that better matches the DT's data architecture in terms of variety, velocity and volume. **Step 8:** Declare all new/modified ontologies classes, update their knowledge bases and generate the new software interface.

4. Case studies: helicopter gearbox and robotic system

4.1. Helicopter gearbox demonstrator

This case study focuses on retrofitting a helicopter's gearbox. It consisted of adding a proximity sensor to measure the gearbox' performance through the shaft's rotational speed. To do so, a ring shaft was retrofitted by adding a new screw to it. So, the proximity sensor can detect that new screw to measure the shaft's rotational speed for improving the gearbox' availability.

The implementation of the framework began by identifying the common language to describe an asset and its composition in a hierarchical way. Fig. 3(a) shows the asset hierarchy, classes, and relations for *Stage* 1. *Stage* 2 focused on a new hole that needed to be manufactured on a ring and the KD of the hole was formalised. Step 1 defined the properties of the hole including the object (Fig. 3(a)). Since no existing schema can represent the new KD of a hole (Step 2), all the missing classes and properties were identified (Step 3). Since neither existing classes nor attributes were present to describe a hole (Step 4), the design process proceeded to Step 6, which is to create new classes (e.g. GeometricalHole) with the required attributes including temporal attributes (e.g. hasDesigned-DateTimeUpdated) and relation to the existing class. Since no other plausible declaration of class with its attributes was identified (Step 7), the existing ontology was updated followed by the implementation of software interface (Step 8). A web-based software that records manufacturing activities done on the asset was created to update the component (e.g. adding a hole). This action created a new individual for a hole on a specific component and communicated to the design software to update its 3D model through a plug-in.

Fig. 4 shows the final result, working on a plug-in for the design software that understands changes in DT data made by other software using inferencing and automatically generates a new version of the component representation to reflect co-evolution with the asset. This ensures that the latest change (e.g. 3D model) has been automatically incorporated. The result is that the updated DT is readily available to other software systems e.g. condition monitoring, which results in feedback to the physical asset. From a variety perspective, the ontology-based architecture requires two new relationships and six new attributes (variety = 8). The SQL approach would require two new tables to add the foreign keys (2 relationships) and 6 new columns (attributes). Although, the 'variety' number is similar, the ontology-based approach would require less effort to apply these changes because it does not require to modify pre-existing schemas. From a volume perspective, the amount of data to be modified is similar in both cases. However, when the number of individuals to modify due to updates on a class declaration (a table in SQL) increase, the ontology-based approach would simplify data consistency. For

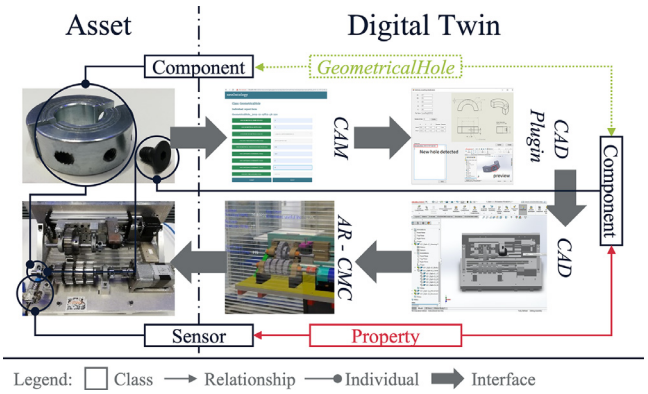

Fig. 4. Co-evolution of DT with its physical asset in the gearbox example.

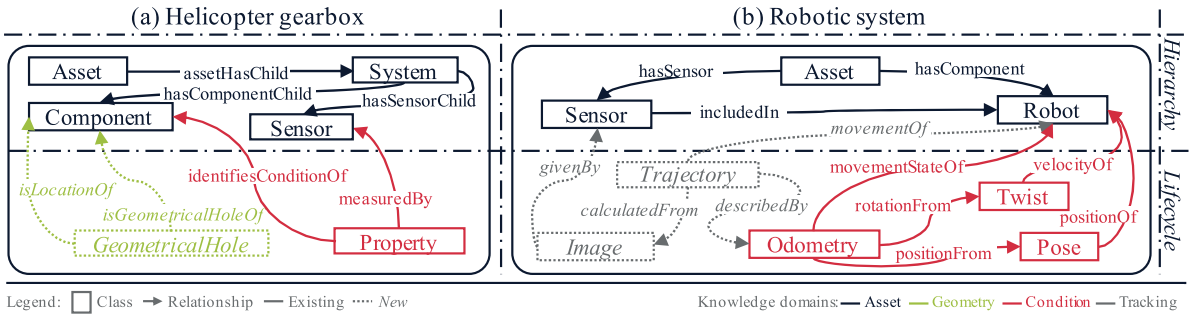

Fig. 3. Ontology Design Framework: Cases of study.

Please cite this article as: J.A. Erkoyuncu et al., A design framework for adaptive digital twins, CIRP Annals - Manufacturing Technology (2020), https://doi.org/10.1016/j.cirp.2020.04.086

velocity, the number of interfaces to be changed (speed to spread changes) is lesser (faster), since it will only require to update the CAD interface (velocity = 1), rather than the CAM and CAD interfaces (as in the SQL approach, velocity = 2).

4.2. Robotic system

The second case study demonstrates (Fig. 3b) how the existing DT of a laboratory-scale mobile robot system is adapted using the developed framework. The final result (Fig. 5), shows how the proposed framework improves tracking, monitoring, and navigating the existing DT. The existing system consists of real robots and a DT that comprises a simulator and navigation software that works with the simulator. The goal of the adaptation is to design and implement a camera-based tracking system that will enable the navigation software to work with real robots by providing the transformations between the robots' coordinate systems required by the navigation software. Also, a monitoring system is envisioned that enables real-time tracking of robot trajectories over time. The adaptation therefore includes installing an overhead camera, developing a tracking software that detects robot markers and calculates their global poses, and developing a monitoring module for visualising trajectory.

This case study shows how the steps of *Stage 2* can be used to evolve the DT. The existing ontology already contains classes that describe the robots for the purposes of the simulation, e.g. classes describing locations, velocities, and coordinate system transformations of robots and their joints: Odometry, Twist, and Pose. The tracking system can reuse these interfaces (Steps 1 and 2). However, there are neither classes that can be used to describe the images generated by the camera, nor are there classes that provide a suitable interface for trajectory tracking, required by the monitoring system (identified in Steps 3 and 4). New classes, Image and Trajectory, are added (Step 6) and evaluated (Step 7).

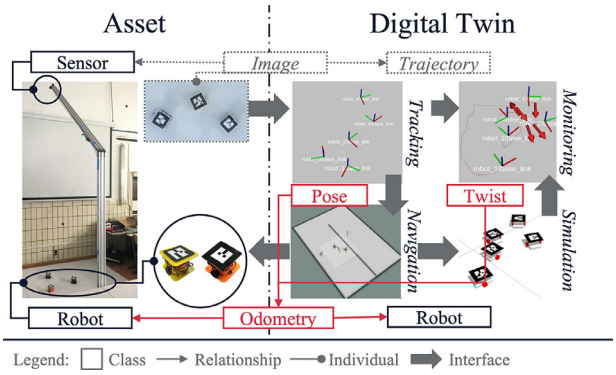

Fig. 5. Ontology-based communication in the mobile robotic case study.

Similar to the helicopter gearbox case, there are advantages in variety, volume and velocity in the ontology based approach. In total, two classes, six attributes, and four relationships are created (variety = 10). One interface is modified due to the change to the Odometry class (velocity = 1). If, on the other hand, the system was implemented using a relational database, a new table would have to be created for each new class and each new relationship, which would result in a total of six new tables and one modified table.

5. Conclusions and future work

This paper offers a design framework to capture changes in the asset across its life-cycle within a DT from the perspective of data architecture. This contributes to research with the ability for synchronization between the physical and the digital world to establish closed loops. The developed ontology-based approach offers opportunities for model and model-traceability to apply model-based systems engineering development approaches for DT solutions. It also offers a unified single object model, which enables to replace a group of translational interface standards. Future work is needed in applying the developed framework through a simulation study in a controlled experiment to test the life-cycle and in giving feedback from DT to the physical asset.

Acknowledgements

The research was partially supported by the EPSRC funded project DigiTOP (EP/R032718/1), and the Slovenian Research Agency (P2-0270). The work was cooperated between Cranfield University, City, University of London and Ljubljana University. The authors also acknowledge Babcock International for their support with this work. Data underlying this paper can be accessed at https://doi.org/10.17862/cranfield.rd.12136074.

References

[1] Helu M, Joseph A, Hedberg T (2018) A Standards-Based Approach for Linking as-Planned to as-Fabricated Product Data. *CIRP Annals - Manufacturing Technology* 67:487–490.
[2] Tomiyama T, Lutters E, Stark R, Abramovici M (2019) Development Capabilities for Smart Products. *CIRP Annals - Manufacturing Technology* 68:727–750.
[3] Bilberg A, Malik A (2019) Digital Twin Driven Human-Robot Collaborative Assembly. *CIRP Annals - Manufacturing Technology* 68:499–502.
[4] Tao F, Zhang M, Liu Y, Nee AYC (2018) Digital Twin Driven Prognostics and Health Management for Complex Equipment. *CIRP Annals - Manufacturing Technology* 67:169–172.
[5] Stark R, Fresemann C, Lindow K (2019) Development and Operation of Digital Twins for Technical Systems and Services. *CIRP Annals - Manufacturing Technology* 68:129–132.
[6] Schleich B, Anwer N, Mathieu L, Wartzack S (2017) Shaping the Digital Twin for Design and Production Engineering. *CIRP Annals - Manufacturing Technology* 66:141–144.
[7] Abramovici M, Göbel J, Dang H (2016) Semantic Data Management for the Development and Continuous Reconfiguration of Smart Products and Systems. *CIRP Annals - Manufacturing Technology* 65:185–188.
[8] Zhu E, Ong P, Nee AYC (2015) A Context-Aware Augmented Reality System to Assist the Maintenance Operators. *International Journal of Computer Integrated Manufacturing* 28(2):213–225.
[9] Erkoyuncu JA, Fernández I, Mura MD, Roy R, Dini G (2017) Improving Efficiency of Industrial Maintenance With Context Aware Adaptive Authoring in Augmented Reality. *CIRP Annals - Manufacturing Technology* 66(1):465–468.
[10] Stark R, Pfortner A (2015) Integrating Ontology Into PLM- Tools to Improve Sustainable Product Development. *CIRP Annals - Manufacturing Technology* 64 (1):157–160.
[11] Krötzsch M, Frantisek S, Horrocks I (2015) Description Logics. *IEEE Intelligent Systems* 29(1):12–19.