Author preprint. Official publication may differ.
Accepted and best paper receipient at SEAA 2024 conference (https://dsd-seaa.com/seaa2024/).

Maintaining consistency of digital twin models: exploring the potential of graph-based approaches

Hossain Muhammad Muctadir, Loek Cleophas, Mark van den Brand
Software Engineering & Technology cluster, Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
Email: {h.m.muctadir, l.g.w.a.cleophas, m.g.j.v.d.brand}@tue.nl

***Abstract*—Digital twins (DTs) encapsulate the concept of a real-world entity (RE) and corresponding bidirectionally connected virtual one (VE) mimicking certain aspects of the former in order to facilitate various use-cases such as predictive maintenance. DTs typically encompass various models that are often developed by experts from different domains using diverse tools. To maintain consistency among these models and ensure the continued functioning of the system, effective identification of any consistency issues and addressing them whenever necessary is imperative. In this paper, we investigate the concept of consistency management and propose a consistency management framework that addresses various characteristics of DT models. Subsequently, we present two case-studies that implement the proposed framework with graph-based techniques. Taking into account both case-studies, we argue that the graph-based approaches have significant potential and invite further exploration.**

***Index Terms*—digital twin, consistency management, model consistency, model management, model inconsistency**

I. Introduction

A DT is a software intensive system where certain aspects of a real-world entity (RE) are mimicked in one or more connected virtual entities (VE) in order to facilitate various advanced use-cases such as predictive maintenance, optimization, virtual training, and so on. Recently, the concept of DTs has been integrated and used in an increasing number of domains [6]. A significant number of these DTs are related to complex industrial systems where various heterogeneous and cross-domain models and model driven engineering (MDE) play a central role [2], [9], [10], [36]. VEs are abstractions of real-world concepts i.e. models [37]. As a result, VEs can be characterized as model-based [33], and hence the benefits of MDE [9] can be applied to their construction.

Since the concept of DT and its extensive adoption are rather recent, it remains unclear whether DT models exhibit any distinguishing characteristics. Recent studies [7], [42], [43] and our interaction with DT practitioners [33] provided us with early indications about particularities of DT models, especially the ones used in developing VEs. We found that these models are highly multi-domain, heterogeneous, often physics-based, evolve rather frequently, and must maintain sufficient accuracy, trustworthiness, efficiency, and speed when executed. Moreover, we observe that these models, intended to be used collectively, are frequently developed by engineers from different disciplines using various tools. Despite sharing some characteristics with traditional Model-based Systems (MBSs), DT models are potentially more complex to manage due to factors such as higher heterogeneity, frequent evolution, and multi-domain multi-tool nature. Furthermore, managing VE models that are reused or repurposed from their corresponding RE can become even more complex, as they operate beyond their original context.

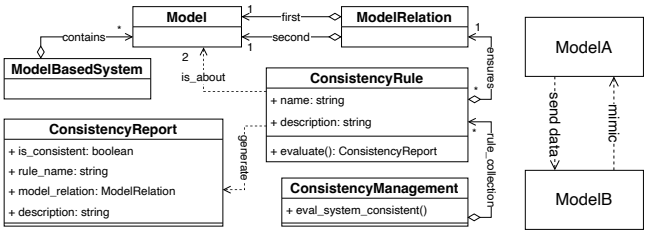
Maintaining consistency among diverse models within an MBS is challenging, especially in the context of DTs, where it is crucial [41]. This is mainly due to varied formalisms and languages used for multi-tool heterogeneous DT models. This heterogeneity is the result of using various commercial and in-house tools in the design and development of these models, with domain specific languages (DSL) playing an increasingly significant role [1], [27], [33]. We identify this heterogeneity as a key challenge for seamless consistency analysis. Our work aims to address such challenges for effectively managing consistency among heterogeneous DT models. Furthermore, we see system-level consistency as an aggregation of model-level consistency, managing which is the focus of our research.

This work is divided into three parts. Firstly, we define the concept of consistency management in the context of DTs. Based on this concept and current literature we investigate the characteristics of DT models and their impact on consistency management of these models. Secondly, we present our consistency management framework for DT models, which considers both the definition of consistency management and the model characteristics. Finally, we present two case-studies with graph-based implementations of our framework.

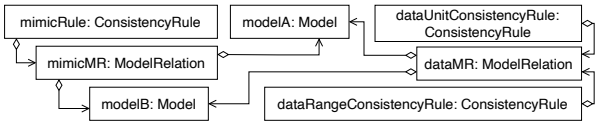
The paper is structured as follows. The definition of consistency management and the characteristics of DT models are discussed in Sections II and III respectively. We present our consistency management framework and its implementation in two case-studies in Sections IV and V respectively. In Section VI we explore related work. The results of the case-studies, related discussions, and future work possibilities are presented in Section VII. Section VIII concludes the paper.

II. Model consistency and its management

In this section we discuss our conceptualisation of consistency management and related notions, which are shown in Figure 1a. Our conceptualisation, which is influenced by the work of Feldmann et al. [17], focuses on capturing the relations among models in an MBS, identifying potential violations of these relations, and recording information necessary



(a) Consistency management meta-model (b) Example models



(c) Example entities needed for consistency management of (b).

Fig. 1. The concept of consistency management and a corresponding example.

to address them. We believe subsequent actions (i.e., confirming identified issues by investigating the MBS and resolving them) are highly dependent on the implementation, potentially complex, and often require a human in the loop [33].

We define model consistency management as the process of maintaining consistency among interrelated models within a DT. The notion of interrelation encapsulates various relationships that can be established between two models. The actual nature and characteristics of this relationship depend on the implementation of the corresponding models and on the use-case. In our meta-model, a relationship is expressed with the abstract concept `ModelRelation`, which establishes a relationship between two models. One or more `ConsistencyRules` ensure the validity of a `ModelRelation`. The validity check is performed using the `evaluate` function, which generates a `ConsistencyReport` consisting of the details of the corresponding `ModelRelation`, `ConsistencyRule`, and indicates any potential consistency issue with the boolean flag `is_consistent`. Additionally, the `description` property provides a comprehensible explanation of the aforementioned details. A DT may encounter consistency issues if any of the consistency rules are violated (i.e., `is_consistent` is set to `false`). Finally, we define the concept of `ConsistencyManagement` as the aggregated evaluation of all `ConsistencyRules` existing within an MBS and, if needed, acting upon the generated `ConsistencyReports`.

We further explain the concept of consistency management using the example in Figures 1b and 1c. Figure 1b shows a simplified example of two related models: `ModelA` and `ModelB`, which respectively belong to an imaginary RE and the corresponding VE. Figure 1c presents the entities needed for the consistency management of these models. They are related in two ways: `ModelB` (1) receives data from `ModelA` and (2) virtually mimics its behaviours. These relations are represented in Figure 1c with `dataMR` and `mimicMR` respectively, which are instances of `ModelRelation`. The consistency of these relations are maintained by three instances of `ConsistencyRule`: `mimicRule`, `dataUnitConsistencyRule`, and `dataRangeConsistencyRule`, where the former is related to `mimicMR` and the latter two to `dataMR`. The `mimicRule` ensures the behavior of `ModelA` is correctly mimicked by `ModelB`. The other two rules ensure the consistency of the data exchanged between the corresponding models. The first one ensures the data is in the correct unit, while the second one verifies the value falls within a predefined acceptable range. Here the consistency management involves verifying the integrity of these consistency rules and promptly reporting any violations.

In [17] Feldmann et al. separately defined the concept of link, linking model elements, and inconsistency, which are later utilised for inconsistency management. In contrast, we explicitly connect the concept of `ModelRelation`, which is similar to the concept of link, and `ConsistencyRule`. This emphasizes the connection between dependent models.

III. Impact of DT Model Characteristics on Consistency Management

As indicated earlier, the notion of DT and its large-scale adoption is a recent phenomenon. Therefore, the known characteristics of DTs and involved models are not yet mature enough. This deficiency is also reflected in the current literature, where our search for characteristics of DT or VE models yielded fewer than 50 publications. We analysed these publications with keyword search and manual skimming. Thereafter, we were left with 15 papers that explicitly mentioned various characteristics of the aforementioned models or indirectly indicated them based on corresponding use-cases. Table I lists these identified characteristics and the related literature. In this section, we discuss these characteristics based on the literature mentioning them. Afterwards, we explore the impacts of these characteristics on the consistency management of DT models.

A. Characteristics of DT models

In this section, we discuss the characteristics from Table I.

TABLE I
Characteristics of DT models identified from the literature

Characteristics	References
Frequent evolution	[20], [28], [29], [34]
Heterogeneity	[4], [7], [8], [20], [32], [40]
Data-intensive	[4], [6]–[8], [20], [24], [29], [34], [39]
Physics-based	[21], [32], [40], [43]
Optimal accuracy or fidelity	[4], [21], [24], [25], [29], [43]
Fast	[8], [43]
Trustworthy	[43]
Real-time	[34]

Frequent evolution. As the real-world counterpart evolves, the VE must co-evolve to stay aligned with it. In a DT context, the evolution of RE can be instigated by, among other factors, enhanced insights gained from simulations within VE. Moreover, meeting new requirements, bug fixes, feature enhancements also drive VE evolution. Therefore, evolution

is one of the key characteristic of DT models. According to Newrzella et al. [34], this evolution lasts the whole lifecycle of the DT in order to hold the current information at all times. Evolution is not unique to DTs as other MBSs can also evolve over time. However, for VE models within a DT this occurs more frequently. Furthermore, we also notice evolution when existing models, primarily developed for real-world entities, are reused for VE development (e.g., Walravens et al. [42]).

Heterogeneity. This is an obvious characteristic of DT models. Due to the complexity and provided functionalities, VEs can contain multiple models encompassing various aspects of the corresponding RE. For example, Tao et al. [40] mentioned four aspects: geometry, physics, behavior, and rules. These models are often cross-domain and multi-tool with divergent syntax and semantics. Therefore, heterogeneity can be recognised as a key characteristic of DT models.

Data-intensive. Data is playing an increasingly important and multifaceted role in the construction and functioning of DTs [41]. We observe data being used for optimisation, developing data-driven models, data streams being used for synchronising RE and corresponding VE, and numerous other key DT functions. Bordeleau et al. [7] emphasized the collection of data and its use through the lifecycle of a DT and demonstrated the role of data with three examples. Barricelli et al. [6] discussed the usage of big data and artificial intelligence (AI) within a DT. Gargalo et al. [20] emphasized that data collection is paramount for the successful development and implementation of DTs in the bio-manufacturing industry.

Physics-based. Such models are used for twinning the physics of the corresponding RE. Wright et al. [43] discussed the importance of physics models within a VE and suggest against using non-physical models as the core of a VE. Glaessgen et al. [21] explained the use of ultra-high fidelity physics models as the backbone of the DT of air force vehicles.

Optimal accuracy or fidelity. Zhang et al. [44] listed several DT definitions ranging from cradle-to-grave model with extremely high fidelity to simplistic data-driven simulation executing in isolation. This shows the disagreement among researchers on how closely a VE should mimic the corresponding RE. However, besides differing opinions on the degree of fidelity, we observe, in most cases, a broad consensus regarding the role of fidelity within VE models [25].

Other characteristics. While a limited group of researchers mentioned speed, trustworthy, and real-time characteristics, we consider these highly relevant for VE models. Wright et al. [43] highlighted high-value and safety-critical systems as the most recognized DT application areas, stressing the importance of trustworthy and fast predictions. These characteristics are also important for other use-cases (i.e., predictive maintenance). Newrzella et al. [34] emphasized the (near) real-time capability of VE models for tracking of the RE.

We acknowledge that this list of characteristics is not complete and more may emerge with further adoption of DTs. Moreover, a DT can be composed of models of multiple types having only a subset of characteristics from Table I. However, we argue that these characteristics form a good enough representation of the known characteristics of DT models and play a crucial role for inter-model consistency management. We base this on the following facts: firstly, these characteristics are extracted from highly relevant, recent, and well-cited publications; secondly, the characteristics are associated with several domains and use-cases signifying their generic and inclusive nature. Therefore, to facilitate a unified, reusable, and comprehensive consistency management among DT models, these characteristics must be taken into account.

B. Impact on consistency management

Based on the identified characteristics, we derive a set of requirements for the consistency management of DT models.

a) REQ01: Uniform representation of heterogeneous data: The *heterogeneous* and *data-intensive* nature of DT models points towards the need for a general enough representation capable of representing syntactically diverse model information. Such a representation is essential for implementing a singular manner of analysing the consistency of the corresponding cross-domain multi-tool models. This requires the transformation of models from their original format to the common representation, which adds additional overhead of this transformation and maintenance of the common representation. However, we strongly believe that this overhead is significantly cost-effective compared to the effort and resources needed for identifying and resolving any consistency issues without an appropriate consistency management system.

b) REQ02: Flexible and efficient data storage: The characteristics discussed in Section III-A suggest that throughout the lifespan of a DT the associated *heterogeneous data-intensive* models frequently undergo *evolution*. In this context, choosing an appropriate storage method for the uniformly represented data, discussed in *REQ01*, is crucial. Furthermore, transforming models from their original format to a unified representation has its challenges (e.g., mismatching or missing attributes, semantical issues). These factors must be taken into account while choosing the storage method. Additionally, since DTs are *real-time data-intensive* systems, the data storage chosen for the corresponding consistency management must be efficient in terms of space and read-write performance.

c) REQ03: Management of consistency rules: As explained in Section II, the consistency management is done by defining consistency rules and checking them each time the DT or part of it evolves. Therefore, these consistency rules play a central role in consistency management, forming the set of requirements a DT must fulfill at all times for its proper functioning. As the DT *evolves*, the consistency rules must also co-evolve in order to keep them relevant and meaningful. Therefore, the management of consistency rules must be an integral part of the model consistency management without introducing significant overhead. The creation and modification of consistency rules should be simplified and supported by suitable tooling. Additionally, automatically identifying potentially outdated consistency rules, often resulting from model evolution, can further minimize the overhead.

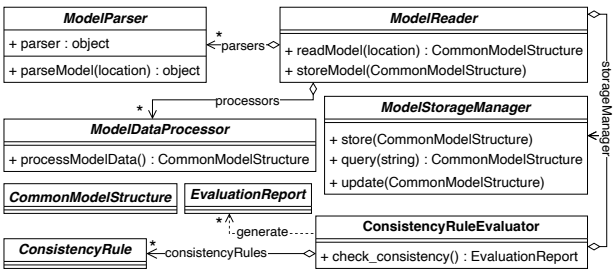


Fig. 2. Consistency management framework for DT models.

We understand that these requirements may extend beyond DTs as other systems might have similar characteristics and, therefore, similar requirements for consistency management. However, we argue that these are the essential prerequisites for any system aimed at consistency management of DT models.

IV. Consistency Management Framework for DTs

We present our consistency management framework in this section and depict it in Figure 2. It has been designed considering the heterogeneity and diversity in terms of the ideation and implementation of DTs, defining entities and their intended functionalities essential for effective consistency management. The actual implementation of the framework is DT specific and may differ significantly across DTs. The framework covers three core functionalities: read and extract model data; store these extracted data; and evaluate consistency rules.

A. Read and extract data from models

This functionality is fulfilled by three classes described below. We define these as abstract classes requiring a modeling environment specific implementation. These environments are used for constructing models contained within a DT. Since the selection of modeling environments may vary for each Digital Twin, the corresponding implementations can also differ.

`ModelParser` is responsible for parsing and extracting raw data from a model. As this class is the only entity directly interacting with the models, it represents the connection between the models and our framework. The parsing is done with the `parser` object, which is specific for and utilises the interface provided by a modeling tool.

`ModelDataProcessor` further processes the parsed raw data and transforms it to `CommonModelStructure`, which is the common representation (cf., *REQ01* in Section III-B) of the heterogeneous models related to the corresponding DT. `ModelDataProcessor` is also responsible for any other post-processing of the raw model data.

`ModelReader` manages the whole process of reading the models using appropriate `ModelDataProcessor` and `ModelParser` classes, and storing the resulting `CommonModelStructure` using the `ModelStorageManager`. For this purpose, it maintains the knowledge about the models used in a DT, the modeling environments used for constructing these, and appropriate implementations of the parser and processor classes capable of interacting with these environments.

B. Store extracted data

The storage of the extracted model data is managed by the `ModelStorageManager`. Additionally, this class is responsible for two very important functions: (1) interfacing with the lower level storage (e.g., Neo4j[1]); and (2) ensuring adherence to the `CommonModelStructure` for the stored data. As an interface to the underlying storage, `ModelStorageManager` decouples the storage from the consistency management. Therefore, switching between storage solutions is possible by choosing the appropriate implementation of the `ModelStorageManager`.

We discussed the importance of uniform representation of heterogeneous data (cf., *REQ01* in Section III-B). Defining a global unified representation supporting all major modeling formalisms is a challenging task, but is not needed either, as any particular DT only uses a limited set of formalisms. Therefore, our consistency management framework is designed around a local unified representation, denoted as `CommonModelStructure`, only unifying the formalisms used within a DT. This representation forms the schema of the stored model data and establishes the basis for defining the `ConsistencyRule`.

C. Define and evaluate consistency rules

The definition of consistency management presented in Section II includes the concept of `ConsistencyRule`, which is reused within the consistency management framework (cf. Figure 2). `ConsistencyRuleEvaluator` is responsible for evaluating all such rules and generating `EvaluationReports`, which is analogous to the `ConsistencyReport` from Section II.

V. Case-studies

In this section we present two case-studies in which we implement our consistency management framework introduced in Section IV. The first one is about analyzing the consistency among models of a DT of an autonomous truck capable of docking itself at a distribution center [5]. The second one analyses consistency of textual grammar-based DSL definitions (i.e., Xtext[2]). For this purpose, we use the Xtext-based pilot implementation of the second version of the Systems Modeling Language (SysML-V2) [38].

The implementations of both case-studies share numerous similarities in terms of activities and used technologies. Figure 3 shows an overview of performed activities within the case-studies and their relation with the key functionalities of our consistency management framework (cf. Section IV). We start with extracting information from the models and storing them as JSON files. These files allow cross programming language interfacing as the programming interface language varies across modeling tools. Afterwards, we further process this raw data to (1) convert them to a `CommonModelStructure`, which is explained in Section IV-A; (2) establish connections between different model

[1]Property graph database https://neo4j.com/product/neo4j-graph-database
[2]Language workbench Xtext https://eclipse.dev/Xtext/

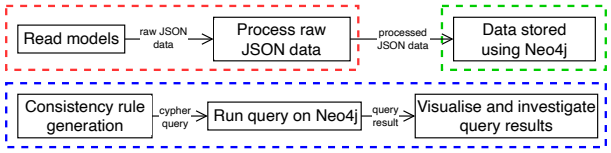
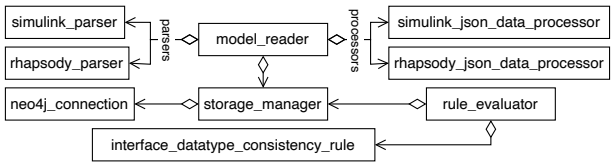
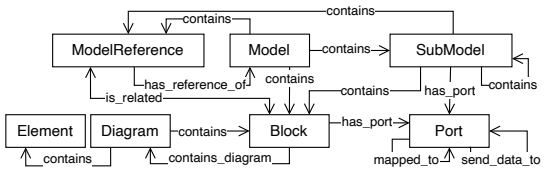


Fig. 3. Overview of performed activities in the case-studies for implementing our consistency management framework, as explained in Section IV. The colored dashed boxes indicate the relation with the key functionalities of the framework. The red, green, and blue highlighted activities are related to functionalities explained in Sections IV-A, IV-B, and IV-C respectively. Furthermore, due to data and process dependencies, these activities must occur in the same order (i.e., first red, then green, and finally the blue activities).



(a) Implemented framework entities for truck DT case-study



(b) Part of the unified data model for Simulink and Rhapsody

Fig. 4. Overview of the implementation of the consistency management framework for Simulink and Rhapsody model from autonomous truck DT [5].

entities, especially in cross-tool scenarios, where relationships may not be explicit; (3) investigate and fix any faulty data. In the following step, we store the processed data using Neo4j, a labelled property graph database. We use the nodes and edges in the graph data to respectively represent various modeling entities and relations among them. This structure is the manifestation of the concept of `CommonModelStructure` and varies across case-studies (i.e., Figures 4b and 5b). Once stored, this graph data can be queried for patterns that are useful for ensuring consistency between the corresponding models. In the case-studies we generate Cypher [19] queries with different implementations of `ConsistencyRule` (cf. Figure 2), where each implementation represents specific consistency management scenarios. In the next sections, we present examples of these generated queries for the corresponding case-studies. Finally, we investigate the query results and related model entities to identify any consistency issues.

A. Consistency management of delivery truck docking DT

In this case-study we perform consistency management of models contained in the DT of an autonomous truck, which is capable of autonomously driving and docking itself within a virtual distribution center [5]. The VE is created by integrating Simulilnk (a simulation environment), IBM Rhapsody (a SysML modeling tool), and Unity (a 3D game engine) models. The behavior and structure of the distribution center and the autonomous truck are modeled in Rhapsody. The Simulink model handles the truck's path planning, kinematics, and collision detection. The Simulink models are imported into Rhapsody as `Blocks`, with their `Ports` mapped to corresponding `Ports` in the Simulink model. The 3D environment including the physics and interactivity is implemented in a Unity model. This case-study focuses on the consistency management between the Simulink and Rhapsody models.

Figure 4a shows an overview of the implementation of our consistency management framework for this case-study. We implemented two sets of parsers and data processors for reading information from the Simulink and Rhapsody models. The data processors ensure the conformity of the processed data with the meta-model presented in Figure 4b, which is created by merging similar concepts from both of the modeling tools. For example, in both modeling environments a model can refer to external models and we encapsulate this concept as `ModelReference`. Here the concept of similarity is determined by the modeler implementing the framework depending on the context. Furthermore, we show an example of a consistency rule titled `interface_datatype_consistency_rule`. It checks if the data type of a `Port` in a Rhapsody model matches the corresponding `Port` in Simulink. This rule is formulated as a Cypher query, which is shown in Listing 1. Running it on the data stored in Neo4j retrieves pairs of mapped ports with mismatching data types, which might cause runtime issues.

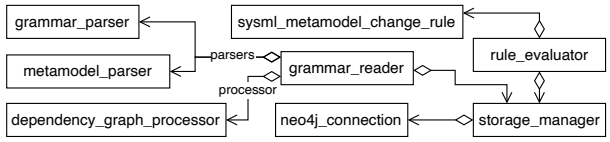
```
match (rpsd_prt:Port)-[:MAPPED_TO]->(smln_prt:Port)
where rpsd_prt.data_type is null or smln_prt.
    data_type is null
or rpsd_prt.data_type <> smln_prt.data_type
return rpsd_prt, smln_prt
```

Listing 1. Cypher query for checking datatype similarity

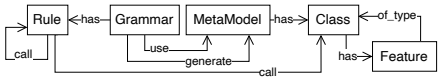
B. Consistency management of Xtext grammar definition

With this case-study we perform a change impact analysis on the SysML-V2 Xtext grammar [38] supporting the maintenance of its consistency. Xtext is an open-source framework for creating textual syntax for DSLs. Here we use various Xtext specific terms, explained in [12], omitting explanations for brevity. Although seemingly unrelated, we believe this case-study is relevant for DTs since (1) the usage of DSLs is increasing within the scope of DTs [1], [27], [33]; (2) while general-purpose modeling languages (e.g., UML and Simulink) evolve infrequently, in-house DSLs in industrial settings are more frequently adapted [11], [22], increasing the likelihood of introducing inconsistencies.

An overview of our implementation is shown in Figure 5a. We start with extracting and storing various entities (i.e., classes, features and rules) and relations among them, shown in Figure 5b, from the meta-model and grammar definitions available in the repository. The consistency rules again generate Cypher queries [19] that extract information from the graph database. For example, Figure 5a shows the consistency rule



(a) Implemented framework entities



(b) Part of the unified data schema

Fig. 5. Overview of the implementation of the consistency management framework for Xtext grammars and meta-models in SysML-V2 repo [38].

`sysml_metamodel_change_rule`, which takes as input a class name from the *sysml.ecore* meta-model and returns the set of grammar rules that use the aforementioned class. Listing 2 shows such a generated query for the `Documentation` class. This simple yet useful consistency rule can offer insight into potential impacts on grammar rules when modifying *sysml.ecore*, allowing modelers to make informed decisions and potentially avoid consistency issues.

```
match (gm:Grammar)-[:use]->(mm:MetaModel{
   uri:'https://www.omg.org/spec/SysML/20240201'})
match (gm)-[:has]->(r:Rule)-[:call]->(
   cls:Class {name:'Documentation'})<-[:has]-(mm)
return r, gm
```

Listing 2. Example of a generated cypher query for `Documentation` class

VI. Graph-based Consistency Management: State of the Art

In this section, we study prominent graph-based consistency management and related approaches from existing literature.

Inconsistency detection among software models, design artifacts, and related entities is a well-established research topic. The work of Balzer [3] shows that similar concepts were known as early as the late 1980s. The existing inconsistency detection approaches can be categorised into: proof-theory based, rule-based, and synchronisation-based [15]. In this section, we explore prominent graph-based approaches which are primarily categorised as rule-based. Herzig et al. [23] argued that inconsistencies can be represented with graph patterns and formally identified through pattern matching. Furthermore, Feldmann et al. [15] argued that the flexibility of the rule-based approaches are better suited for situations with incomplete knowledge of the underlying system.

Feldmann et al. published a series of papers focusing on inter-model consistency management [13], [14], [16], [17]. In [16] the authors proposed a conceptual approach for inconsistency detection and performed a case-study demonstrating its technical viability. They used semantic web technologies for knowledge representation and querying encoded inconsistencies. This work was further extended to include different levels of graph-based visualisation of models and detected inconsistencies where lower-level visuals offer more information to pinpoint the latter [14]. A formal definition of inconsistency management problem is presented in [17] where they defined the concept of links, which establish relations among cross-domain models, and assessable consistency relations. This work was extended with two domain specific languages (DSL) for modeling the links and pattern based consistency rules [13]. We discuss the difference between our approach and the work of Feldmann et al. in Section II.

Mens et al. [30], [31] proposed a graph transformation based model inconsistency detection and resolution method where the inconsistencies are represented as graph transformation rules. Afterwards, they use critical pair analysis, a method typically used in graph rewriting for identifying minimal example of a conflicting situation, to identify inconsistencies.

These approaches demonstrated promising results for respective use-cases. In comparison, our consistency management framework is unique in two aspects: firstly, it is specifically tailored for the consistency management of DT models, with requirements derived from the characteristics of them; secondly, while the discussed approaches are mostly theoretical, our method is guided by software engineering practices (e.g., reuse, modularity, and practical implementability).

VII. Results, Discussions, and Future Possibilities

In Section III-A, we outlined characteristics of DT models. Based on these, we defined three requirements (cf. Section III-B) for any system aiming at consistency management of such models. Requirement *REQ01* emphasizes the need for a general enough representation capable of representing data from heterogeneous models. To address this, our consistency management framework includes the concept of `CommonModelStructure` (cf. Section IV), which is a uniform representation of data extracted from heterogeneous models used within a particular MBS. Our case-studies (cf. Section V) are perfect examples of implementation of this concept, which are depicted in Figures 4b and 5b.

REQ02 is about flexible and efficient data storage. In our case studies, we use graph-based storage (i.e., Neo4j), which is known for its flexibility and exceptional ability to manage highly interconnected and complex data [18], [35]. Furthermore, Neo4j has demonstrated high efficiency across various benchmarks [18], [26]. Based on these works and our case-studies, we believe that graph-based data representation and storage has great potential for consistency management of DT models and, therefore, addressing *REQ02*. This potential has been recognised in various existing research works, as we presented in Section VI. However, these works, including ours, are mostly academic in nature and we are unaware of any followup work involving more complex systems such as DTs. We believe further investigation is essential to reveal the full potential of graph-based model consistency management. Moreover, we are yet to discover the frequency of evolution our framework can handle and its ability for providing real-time feedback (e.g., while developing using a modeling tool).

The concept of `ConsistencyRule` plays a central role in our consistency management framework. Such rules encapsulate the nature of the relationships among models within

an MBS and make them explicit. Our case-studies show two examples of such rules. However, we foresee a significant increase in the number of these rules for more complex and larger systems. As a result, consistency rule management becomes essential and potentially resource intensive, specially for DTs as emphasised by *REQ03*. This requirement is not sufficiently addressed in our consistency management framework and remains a priority for our ongoing research.

With the two case-studies presented in this paper, we demonstrated the applicability and functionality of our consistency management framework. However, these case-studies are relatively small and academic in nature. This significantly limits the evaluation of the framework in terms of the kinds of inconsistencies it is able to handle, modeling environments that can be supported, data limitations, efficiency and so on. As a result, the practicality and applicability of this approach remain unexplored for more complex systems. Therefore, we foresee great potential in experimenting with larger MBSs and DTs. Such experiments can also be used to further evaluate our claim regarding the suitability of the graph-based implementation of our framework.

We discussed the key functionalities of our consistency management framework in Section IV and their implementation for our case-studies in Section V, where we shared the similarities between the case-studies in terms of technologies and activities. However, the actual implementations of the case-studies do not share any code or implementation artifacts. We believe further research on this topic can be useful in identifying framework components that can have standard and reusable implementation configurable via parameters to suit different modeling environments and projects.

Methods for defining consistency rules remain an open research topic. From the literature we see various proposed methods to do so. For example, Feldmann et al. [13] proposed using triple graph grammars for defining consistency rules. In our case-studies, we generated Cypher queries for essentially the same purpose. However, we were unable to identify any widespread adoption of previously proposed methods within academia or outside. Further research on this topic, involving large use-cases, is essential to find a solution with wider applicability. This is especially significant for DTs considering their extensive acceptance in recent years and the catastrophic consequences inconsistencies can have on them.

VIII. Conclusion

In this paper, we presented our research on DT model consistency management. We initiated the discussion by presenting our conceptualisation of consistency management, which allowed us to establish a clear and unambiguous definition of consistency management and identify the core concepts related to it. We continued our discussion with a deeper look into the available literature aiming to understand the current perception about different characteristics of DT models, their differences with models belonging to traditional MBSs, and the role of these differences in the context of consistency management of DT models. We found that DT models are known to be, among other characteristics, more *heterogeneous*, highly *data intensive*, and encounter more frequent *evolution*. This list of characteristics indicated the importance of implementing a proper consistency management for the DTs to maintain their continued and correct functioning. With this understanding and based on our definition of consistency management, we proposed our consistency management framework and implemented it in two relatively small yet highly relevant case-studies. With these case-studies we demonstrated the applicability of our framework and its potential in addressing the challenges related to consistency management in the context of DT models. We also showed the capabilities of graph-based storage techniques in storing heterogeneous data, which is particularly important for heterogeneous DT models.

The work presented in this paper is part of our ongoing research on consistency management of DT models. We intend to further explore this topic with additional case-studies involving systems having greater complexity and scale. Our objective is to rigorously test our framework across diverse scenarios, refining and enhancing it to achieve a higher degree of maturity. Moreover, we want to further evaluate our claim regarding the advantages of graph-based formalisms for the representation and management of heterogeneous models, which hold great potential for DT applications.

Acknowledgment

This research was funded by NWO (the Dutch national research council) under the NWO AES Perspectief program, project code P18-03 P3. We acknowledge the contributions of Judith Houdijk and Mohammad Ibrahim to the case-studies.

References

[1] H. Ahmad, A. Chaudhary, and T. Margaria, "Dsl-based interoperability and integration in the smart manufacturing digital thread," *Electronic Communications of the EASST*, vol. 81, p. 2021, 11 2022.

[2] P. Baker, S. Loh, and F. Weil, "Model-driven engineering in a large industrial context — motorola case study," in *Model Driven Engineering Languages and Systems*, L. Briand and C. Williams, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 476–491.

[3] R. Balzer, "Tolerating inconsistency," in *Proceedings of the 5th International Software Process Workshop*. Los Alamitos, CA, USA: IEEE Computer Society, oct 1989, pp. 41,42.

[4] J. Bao, D. Guo, J. Li, and J. Zhang, "The modelling and operations for the digital twin in the context of manufacturing," *Enterprise Information Systems*, vol. 13, pp. 534–556, 4 2019.

[5] I. Barosan, A. A. Basmenj, S. G. R. Chouhan, and D. Manrique, "Development of a virtual simulation environment and a digital twin of an autonomous driving truck for a distribution center," in *Software Architecture*. Cham: Springer International Publishing, 2020, pp. 542–557.

[6] B. R. Barricelli, E. Casiraghi, and D. Fogli, "A survey on digital twin: Definitions, characteristics, applications, and design implications," *IEEE Access*, vol. 7, 2019.

[7] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards model-driven digital twin engineering: Current opportunities and future challenges," in *Systems Modelling and Management*, Ö. Babur, J. Denil, and B. Vogel-Heuser, Eds. Cham: Springer International Publishing, 2020, pp. 43–54.

[8] C. Cronrath, L. Ekstrom, and B. Lennartson, "Formal properties of the digital twin-implications for learning, optimization, and control," *IEEE International Conference on Automation Science and Engineering*, vol. 2020-August, pp. 679–684, 8 2020.

[9] J. Davies, J. Gibbons, J. Welch, and E. Crichton, "Model-driven engineering of information systems: 10 years and 1000 versions," *Science of Computer Programming*, vol. 89, pp. 88–104, 2014, special issue on Success Stories in Model Driven Engineering.

[10] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, "A survey of model driven engineering in robotics," *Journal of Computer Languages*, vol. 62, p. 101021, 2021.

[11] D. Durisic, M. Staron, M. Tichy, and J. Hansson, "Evolution of long-term industrial meta-models – an automotive case study of autosar," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2014, pp. 141–148.

[12] S. Efftinge and M. Spoenemann, "Xtext - The Grammar Language — eclipse.dev," https://eclipse.dev/Xtext/documentation/301_grammarlanguage.html, 2024, [Accessed 15-02-2024].

[13] S. Feldmann, K. Kernschmidt, M. Wimmer, and B. Vogel-Heuser, "Managing inter-model inconsistencies in model-based systems engineering: Application in automated production systems engineering," *Journal of Systems and Software*, vol. 153, pp. 105–134, 7 2019.

[14] S. Feldmann, F. Hauer, D. Pantförder, F. Pankratz, G. Klinker, and B. Vogel-Heuser, "Management of inconsistencies in domain-spanning models – an interactive visualization approach," in *Human Interface and the Management of Information: Information, Knowledge and Interaction Design*, S. Yamamoto, Ed. Cham: Springer International Publishing, 2017, pp. 71–87.

[15] S. Feldmann, S. J. I. Herzig, K. Kernschmidt, T. Wolfenstetter, D. Kammerl, A. Qamar, U. Lindemann, H. Krcmar, C. J. J. Paredis, and B. Vogel-Heuser, "A comparison of inconsistency management approaches using a mechatronic manufacturing system design case study," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, vol. 2015-October. IEEE, 8 2015, pp. 158–165.

[16] S. Feldmann, S. J. Herzig, K. Kernschmidt, T. Wolfenstetter, D. Kammerl, A. Qamar, U. Lindemann, H. Krcmar, C. J. Paredis, and B. Vogel-Heuser, "Towards effective management of inconsistencies in model-based engineering of automated production systems," *IFAC-PapersOnLine*, vol. 48, pp. 916–923, 1 2015.

[17] S. Feldmann, M. Wimmer, K. Kernschmidt, and B. Vogel-Heuser, "A comprehensive approach for managing inter-model inconsistencies in automated production systems engineering," *IEEE International Conference on Automation Science and Engineering*, vol. 2016-November, pp. 1120–1127, 11 2016.

[18] D. Fernandes. and J. Bernardino., "Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb," in *Proceedings of the 7th International Conference on Data Science, Technology and Applications - DATA*, INSTICC. SciTePress, 2018, pp. 373–380.

[19] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1433–1445.

[20] C. L. Gargalo, S. C. de las Heras, M. N. Jones, I. Udugama, S. S. Mansouri, U. Krühne, and K. V. Gernaey, *Towards the Development of Digital Twins for the Bio-manufacturing Industry*. NLM (Medline), 2020, vol. 176, pp. 1–34.

[21] E. Glaessgen and D. Stargel, "The digital twin paradigm for future nasa and u.s. air force vehicles," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference;20th AIAA/ASME/AHS Adaptive Structures Conference;14th AIAA*, p. 1818, 4 2012.

[22] R. Hebig, D. E. Khelladi, and R. Bendraou, "Approaches to co-evolution of metamodels and models: A survey," *IEEE Transactions on Software Engineering*, vol. 43, pp. 396–414, 2017.

[23] S. J. Herzig, A. Qamar, and C. J. Paredis, "An approach to identifying inconsistencies in model-based systems engineering," *Procedia Computer Science*, vol. 28, pp. 354–362, 1 2014.

[24] W. Jia, W. Wang, and Z. Zhang, "From simple digital twin to complex digital twin part i: A novel modeling method for multi-scale and multi-scenario digital twin," *Advanced Engineering Informatics*, vol. 53, p. 101706, 8 2022.

[25] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks, "Characterising the digital twin: A systematic literature review," *CIRP Journal of Manufacturing Science and Technology*, vol. 29, pp. 36–52, 5 2020.

[26] S. Jouili and V. Vansteenberghe, "An empirical comparison of graph databases," in *2013 International Conference on Social Computing*, 2013, pp. 708–715.

[27] J. C. Kirchhof, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Model-driven digital twin construction: Synthesizing the integration of cyber-physical systems with their information systems," *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2020*, vol. 12, pp. 90–101, 10 2020.

[28] L. Lattanzi, R. Raffaeli, M. Peruzzini, and M. Pellicciari, "Digital twin for smart manufacturing: a review of concepts towards a practical industrial implementation," *International Journal of Computer Integrated Manufacturing*, vol. 34, pp. 567–597, 6 2021.

[29] X. Liu, D. Jiang, B. Tao, F. Xiang, G. Jiang, Y. Sun, J. Kong, and G. Li, "A systematic review of digital twin about physical entities, virtual models, twin data, and applications," *Advanced Engineering Informatics*, vol. 55, p. 101876, 1 2023.

[30] T. Mens and R. V. D. Straeten, "Incremental resolution of model inconsistencies," in *Recent Trends in Algebraic Development Techniques*, vol. 4409 LNCS. Springer, Berlin, Heidelberg, 2007, pp. 111–126.

[31] T. Mens, R. V. D. Straeten, and M. D'Hondt, "Detecting and resolving model inconsistencies using transformation dependency analysis," in *Model Driven Engineering Languages and Systems*, vol. 4199 LNCS. Springer Verlag, 2006, pp. 200–214.

[32] F. Mhenni, F. Vitolo, A. Rega, R. Plateaux, P. Hehenberger, S. Patalano, and J. Y. Choley, "Heterogeneous models integration for safety critical mechatronic systems and related digital twin definition: Application to a collaborative workplace for aircraft assembly," *Applied Sciences 2022, Vol. 12, Page 2787*, vol. 12, p. 2787, 3 2022.

[33] H. M. Muctadir, D. A. Manrique Negrin, R. Gunasekaran, L. Cleophas, M. van den Brand, and B. R. Haverkort, "Current trends in digital twin development, maintenance, and operation: an interview study," *Software and Systems Modeling*, 2024.

[34] S. R. Newrzella, D. W. Franklin, and S. Haider, "5-dimension cross-industry digital twin applications model and analysis of digital twin classification terms and models," *IEEE Access*, vol. 9, pp. 131 306–131 321, 2021.

[35] I. Robinson, J. Webber, and E. Eifrem, *Graph databases: new opportunities for connected data*. O'Reilly Media, Inc., 2015.

[36] A. Rodrigues da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Computer Languages, Systems & Structures*, vol. 43, pp. 139–155, 2015.

[37] D. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[38] E. Seidewitz, H. Miyashita, M. Wilson, H. P. de Koning, Z. Ujhelyi, I. Gomes, T. Schreiber, C. Bock, B. Grill, K. Zoltán, S. Marquez, W. Piers, A. Adavani, and A. Graf, "Sysml-v2-pilot-implementation," https://github.com/Systems-Modeling/SysML-v2-Pilot-Implementation, September 2023, [Accessed 12-04-2024].

[39] S. Shirowzhan, S. M. E. Sepasgozar, R. Ahmad, L. Zhang, T. Ademujimi, and V. Prabhu, "Digital twin for training bayesian networks for fault diagnostics of manufacturing systems," *Sensors 2022, Vol. 22, Page 1430*, vol. 22, p. 1430, 2 2022.

[40] F. Tao, M. Zhang, and A. Nee, "Chapter 3 - five-dimension digital twin modeling and its key technologies," in *Digital Twin Driven Smart Manufacturing*, F. Tao, M. Zhang, and A. Nee, Eds. Academic Press, 2019, pp. 63–81.

[41] M. van den Brand, L. Cleophas, R. Gunasekaran, B. Haverkort, D. Negrin, and H. Muctadir, "Models meet data: Challenges to create virtual entities for digital twins," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2021, pp. 225–228.

[42] G. Walravens, H. M. Muctadir, and L. Cleophas, "Virtual soccer champions: A case study on artifact reuse in soccer robot digital twin construction," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 463–467.

[43] L. Wright and S. Davidson, "How to tell the difference between a model and a digital twin," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 7, pp. 1–13, 12 2020.

[44] L. Zhang, L. Zhou, and B. K. Horn, "Building a right digital twin with model engineering," *Journal of Manufacturing Systems*, vol. 59, pp. 151–164, 4 2021.