Raising Time Awareness in Model-Driven Engineering

Amine Benelallam, Thomas Hartmann, Ludovic Mouline, Francois Fouquet, Johann Bourcier, Olivier Barais, Yves Le Traon

Raising Time Awareness in Model-Driven Engineering
Vision Paper

Amine Benelallam*, Thomas Hartmann†, Ludovic Mouline†*, Francois Fouquet†,
Johann Bourcier*, Olivier Barais*, and Yves Le Traon†
**University of Rennes 1, IRISA, INRIA Centre Rennes, France. email: {first.last}@irisa.fr*
† *Interdisciplinary Center for Security Reliability and Trust (SnT), University of Luxembourg. email:{first.last}@uni.lu*

Abstract—**The conviction that big data analytics is a key for the success of modern businesses is growing deeper, and the mobilisation of companies into adopting it becomes increasingly important. Big data integration projects enable companies to capture their relevant data, to efficiently store it, turn it into domain knowledge, and finally monetize it. In this context, historical data, also called temporal data, is becoming increasingly available and delivers means to analyse the history of applications, discover temporal patterns, and predict future trends. Despite the fact that most data that today's applications are dealing with is inherently temporal current approaches, methodologies, and environments for developing these applications don't provide sufficient support for handling time. We envision that Model-Driven Engineering (MDE) would be an appropriate ecosystem for a seamless and orthogonal integration of time into domain modelling and processing. In this paper, we investigate the state-of-the-art in MDE techniques and tools in order to identify the missing bricks for raising time-awareness in MDE and outline research directions in this emerging domain.**

Index Terms—**Model-Driven Engineering, Analytics, Big Data, Temporal Data, Internet of Things**

1. Introduction

"History is a Greek word which means, literally, just investigation"
–ARNOLD TOYNBEE

1.1. Time-Aware Data-Driven Applications

Thanks to the strong emergence of modern data analytics platforms, data surrounding organisations (enterprises, scientific researchers) is being mobilised to provide reliable insights and a clear and consistent picture within and around their ecosystems. As a matter of fact, according to Forbes[1], a study conducted by Accenture and General Electric shows that 84% of enterprises see the combination of modern analytics and IoT as essential for competitive growth. This undeniable gain is rushing organisations into adopting this new trend (a.k.a. big data integration).

1. https://tinyurl.com/hg7s2x9

The achievement of any big data integration project is tied to the ability of *capturing* relevant data from different sources (sensors, customer's behaviour, etc.), efficiently *storing* it in a reliable and consistent way, finally, turning it into domain knowledge by uncovering data patterns and insights. The quality and relevance of this data play a major role in making accurate data analytics happen. In particular, acquiring the evolution of data in time contributes to high-quality data and delivers means to analyse the history of applications, discover temporal patterns, and predict future trends.

Indeed, temporal data is one of the most common forms of data in data-driven applications. In runtime applications, data evolution is unlikely stationary but rather evolving over time. Nonetheless, currently, existing approaches, methodologies, and environments for developing data-driven applications do not have a native support for time. For instance, existing conceptual modelling languages are not yet capable of capturing the essential semantics of time-evolving information at design time, nor describing evolution constraints over it. Moreover, existing graph processing libraries and query languages are not well-adapted for writing temporal queries and algorithms (e.g. temporal pattern matching).

We envision that MDE, thanks to its level of abstraction, would be an appropriate ecosystem for a seamless and orthogonal integration of time, during the whole development lifecycle of data-driven applications. Moreover, we expect that by raising the awareness of temporal aspects, in application development with MDE, we can significantly enhance the quality of data by giving it a well-defined structure and constrain its undesirable evolution over time.

1.2. Time Awareness in MDE: What Does it Take?

Promoting temporal awareness in application development with MDE is not a new subject. Rivera et al. [1] have identified Time as one of the three challenges that should be addressed by the MDE community. Many approaches have been proposed to extend existing practices in MDE with temporal aspects [2], [3], [4]. To name a few, Bousse et al. [2] use trace management facilities to enable omniscient debugging of xDSLMs. E-Motions [5] extends in-place graph transformation rules with quantitative model of time to allow the analysis and simulation of DSLs. Kanso et al. [3] extended OCL with support for temporal constraints specification for controlling systems behaviour over time. Nonetheless, as of today, most of these approaches focus either on the behavioural aspect or the structural evolution of the system over time. We notice that facets related to temporal data evolution are understudied, namely, its modelling, persistence, and processing. Moreover, most approaches represent time-evolving data as a simple sequence of snapshots of a model [6], [7], e.g. one snapshot per change. Such discretization not only leads to lots of duplicated data (unchanged elements are duplicated in the snapshots of a model) but, more importantly, the state of a model between two snapshots is not defined. This results in losing the semantic of continuously evolving data. How the continuous semantic of time can be efficiently preserved is a challenging research direction [8], [9].

As such, new approaches should be proposed so that MDE may deliver productivity, quality, and maintainability promises to data-driven application development. In this perspective, we pinpoint the following research directions, on which we will elaborate as we proceed: (i) a modelling language and approach to design time-aware data-driven applications, (ii) an adequate persistence framework for persisting and indexing historical data, (iii) an expressive query language for processing historical data.

1.3. Outline of the Paper

In the remainder of the paper, we first introduce a motivational example in Section 2, followed by some preliminary concepts in Section 3. Afterwards, we investigate what progress has been made and what remains to be done in Sections 4–6. We give an overview of the state of the art, and we review some of the main challenges and limitations when applies. In Section 7 we describe how we conceive a time-aware modelling language, then we discuss the integration of temporal aspects as a first-class entity in modelling environments. Finally, Section 8 closes the paper and reveals our future work.

2. The Smart Grid Use Case

To exemplify the need for raising temporal awareness, we use throughout this paper a smart grid case study [9]. Smart grids emerge as the new generation of existing electricity grids to keep pace with the raising demand for energy. They are expected to provide utility companies with full and remote control by leveraging modern information and communications technologies.

To turn this vision into a reality, smart grids accommodate a variety of devices, which organise the grid in self-adaptive and dynamic micro-grids. The important devices for the context of this paper are: **Smart meters**, which are used to continuously measure the consumption of customers and to remotely report it to utility companies throughout data concentrators, for monitoring and billing purposes. **Repeaters** are regular smart meters acting as a bridge for other smart meters. This is useful if, for example, a smart meter can due to disturbances and noise not directly reach a data concentrator. **Data concentrators** control, collect, and store data from the smart meters connected to it. **Central systems** is the main station where all data is aggregated, stored, and analysed.

The topology of a smart-grid network is organised in dynamic subtrees, where each concentrator acts as the root element of the tree. Depending on the signal strength, which, for example, can be influenced by the distance from a smart meter to a data concentrator, weather conditions, noise and other disturbances smart meters dynamically connect to the data concentrator with the best connection characteristics. In addition, at any time new smart meters and concentrators can be added to the network or removed from it. **These dynamic changes can be considered as the evolution of the network topology over time**.

Data about energy consumption is sent on a regular basis from smart meters to their data concentrators. The intervals of collecting consumption data may vary, for example, from 5, 10 to 60 minutes. A common interval is 15 minutes [10]. When extracting knowledge from this data, the temporal dimension of data must be taken into consideration. An example is scheduling charging cycles of electric cars. In order to decide if there could be an overload risk if too many cars charge on the same cable at the same time, the usual load on this cable for this time and date (weekend/working day, winter/summer) needs to be considered. **These tasks require efficient ways to structure, represent, query, and store temporal data efficiently.**

The temporal dimension of data often results in inefficient data querying and iteration operations to find and aggregate the requested data. Therefore, it is of utmost importance to be able to efficiently navigate and query temporal data. Taking again our smart grid example, for each customer, several consumption values per hour are collected. In order to predict the electric load in a certain area (i.e. at one cable), all customers currently connected to this cable need to be queried and then the history of their consumption values need to be analysed. **This illustrates the need for expressive temporal queries**.

In fact, this counts for many application domains. Most data is inherently temporal: from our smart grid example, through self-driving cars, financial applications, medical systems, to insurance applications.

3. Background

Hereafter, we describe the essential background: temporal relational databases, time granularity, and temporal graphs.

3.1. Temporal Relational Databases

Temporal databases have been under active investigation for the last three decades. Some of these studies have focused on how to best address the persistence of temporal data with regards to the nature and the intent of the application under design. Most of these approaches perceive data evolution over time as a sequence of snapshots, each representing a single state of the real-world.

In relational temporal databases, temporal aspects commonly include two attributes, *valid time*, which is the time period during which a fact is true in the real world, and *transaction time* that represents the period during which a record stored in a database is known. Bitemporal databases include both attributes. In this section, we focus on valid time temporal databases. Temporal databases using valid time add two more fields to temporal objects, *valid_from* and *valid_to*. These fields specify the period during which a field or relation is valid. Every time the object evolves in time, a new record is inserted with a validity value starting from the insertion time. The *valid_to* field of the previous record is updated to state that the record is no longer valid.

In an early work, Clifford et al. [11] define a formal semantic for historical databases and intentional logic. Rose and Segev [12] suggest to incorporate temporal structures in the data model itself and extend the entity-relationship data model with temporal concepts. They also discuss the need for a temporal query language for their model and propose some examples. Some of these ideas have been introduced as an extension to the SQL standard, which plenty of commercial tools implement. Similarly to traditional relational databases, temporal ones face serious scalability issues and new approaches based on NoSQL databases have been proposed [13].

3.2. Time Granularity

Data evolution management requires using timestamps to capture the evolution of data over time. Timestamps may vary in granularity as well as representation. A granularity can be regarded as a mapping from integers to a subset of a time domain. Two well-known formalisms have been proposed to express time granularities, collections and slices. Collections come with two classes of operators called **dice** and **slice**, and a primitive type called **calendar**. A calendar is an ordered collection composed of infinite intervals. The dice operator enables to divide an interval to a more fine-grained collection. Whilst, the slice operator selects specific intervals from collections. Similarly to collections, the slices formalism is based on the calendar primitive. However, calendars are considered as a circular list with no first nor last element. Calendars can be dynamically generated from existing calendars. Finally, a slice is a symbolic expression denoting a set of not necessarily consecutive intervals identified by their starting point and their duration.

The expressions C and S denotes the list of all the Mondays of Year 2016 expressed in the collections formalism and the slices formalism respectively:

$C = Mondays : during : Years. = .2016$

$S = \{2016\}.Years{+}all.Weeks{+}\{1\}.Days \triangleright 1.Days$

Time granularity is a major challenge when developing applications concerning temporal data [8].

3.3. Temporal Graphs

One of the early works on temporal graphs were introduced by Vassilis Kostakos [14] as a mechanism for understanding the dynamic properties of systems. The temporal dimension leads to completely new insights and knowledge that are not present in static graphs. For example, while the page rank algorithm enables ranking web pages in search engines, the temporal page rank may bring more insights about how page ranks change over time, and potentially why. According to his work, a temporal graph is a graph that changes over time. A change may be characterised by either adding a new vertex or removing an existing one. Although his definition focuses only on the evolution of the graph topology, in many data-driven applications, developers may also be interested in the evolution of attribute values.

Following the work of Kostakos, several temporal graph processing and storage systems have been proposed in recent years. For example, Chronos [15], and its extension ImmortalGraph [16] are storage and execution engines for iterative graph computation on temporal graphs. Other examples are Historical Graph Store (HGS) [17], Kineograph [18], and GraphTau [6]. With the exception of the work of Hartmann et al. [8], [9], most of these approaches propose data models which basically define temporal graphs as sequences of graph snapshots at specific timepoints, plus deltas in-between these snapshots. These approaches vary in the granularity level at which they track changes over time, as well as at the underlying data model.

4. Time-aware Modelling

The MDE approach has already proved its capacity to cope with software and hardware heterogeinity, executability, and scalability in self-adaptive systems (e.g. IoT). In particular, the models@runtime approach has gained acceptance, and has become the de-facto MDE-based approach to model and execute dynamic adaptive systems.

Fouquet et al. introduced an alternative meta-modelling framework, KMF, adequate for modelling and generating models@runtime-based tools. Rapidly, an ecosystem of tools, DSLs, and code generation techniques was built around it in order to simplify the provisioning, deployment, and reconfiguration of systems software. Similarly to KMF, CloudMF [19] also targets the integration of dynamic adaptive systems in the Cloud. At a different level of abstraction come MindCPS (doMaIN moDel for CPS) [20] and ThingML [21], two DSLs and code generation frameworks targeting CPS and IoT respectively. While MindCPS consists of a DSML to specify software for CPS using a MAPE-K loop style. ThingML is inspired by the UML component and state-chart diagrams to separate the architecture design from the action language. These languages provide a high-level of abstraction to address the heterogeneity and complexity of systems, however, none of them seems to handle the temporal dimension.

On the other hand, the ER (Entity Relationship) community has a long-standing history of groundwork for conceptual modelling to support temporal models. Different approaches have been proposed [22], [23], [24]. Generally, either they adapt the semantics of the existing ER model constructs to support temporal data, or introduce new constructs to the ER model.

Motivated by the lack of support for temporal features in existing (meta-)modelling languages and approaches, we identify the following features that require special attention:

4.1. Evolving Topology and Attributes

Most approaches on systems modelling focus on the static view of the world and neglect the representation of its dynamics. It is only at development time that software developers introduce temporal concerns into the application's business logic. Such knowledge can be captured in advance, by raising time awareness at design time. In this perspective, many proposals have been introduced in late 90's for modelling temporal data, however, most of them are designed for ER modelling, but more importantly, they are based on discrete time scale.

Spaccapietra et al. [22] propose a conceptual temporal model based on the well-known conceptual modelling principles. The authors introduce their solution, MADS, for spatio-temporal applications modelling. The authors defined a concise semantics of different timestamping strategies and levels (attribute, class, and relationship), and introduced four dynamic relationships for modelling dynamic aspects. Finally, the authors showed that their conceptual model can be mapped to traditional temporal databases. Other approaches have followed the same research direction, and an interesting survey expanded on different existing ER temporal models [25]. Complementary approaches [23], [24] were also proposed to guide application designers modelling temporal aspects. As pointed out by our motivation example, in modern data-driven applications, changes in the system, depending on a concept, do not change with the same frequency. More importantly, changes in some concepts need to be tracked on near real-time (pseudo-continuous). Unfortunately, none of the existing approaches seems to support these features.

4.2. Evolution Constraints

While the timestamping mechanism enriches the static view of data by recording its evolution in the form of contiguous intervals, evolution constraints are imposed to control how data should evolve over time. Usually, modelling languages do not include constructs to express dynamic constraints. OCL [26] (Object Constraint Language) is the defacto constraint specification language in MDE. Constraints expressed in OCL must hold at any point in time. They are evaluated against a single system state, except when having @**pre** or @**post**, they are evaluated with respect to the previous or next state as well. This type of constraints is suitable for attributes or associations values that either have a constant value or regular types. As we discussed earlier the need for handling continuous data types, standard OCL stands not suitable.

Several studies [3], [27], [28] have extended OCL with time. In their paper [27], Hamie et al. introduced two operators to the OCL language, **eventually** for describing liveness constraints, and **initially** for describing initial constraints. Conrad et al. [28] extend with a set of operators, inspired from temporal logic, to enable better expression of both future and past tenses. Kanso et al. [3] propose a closely similar extension, which they augment with the concept of state change events. None of the existing approaches handles time granularities nor continuous time. The constraint specification language should have the ability to not only consider time as a discrete set of state changes but also, one global state continuously evolving over time.

4.3. Further Development

We identify the need for a time-aware modelling as the first essential brick. Most of the proposed conceptual temporal models agreed on several requirements when dealing with modelling time evolving data. In particular, they consider orthogonality as the most important requirement. It consists in the ability to specify temporal constructors separately and independently from other static constructors (classes, attributes, and relationships). The language should give the application designer the freedom to decide whether to add or not the temporal dimension to a concept in the system. An annotation mechanism can be adopted for this purpose (as it is a common practice among MDE developers) and a well-defined and concise semantics should be provided. A user-friendly notation for time granularities should also be embedded in the language to help simplify temporal travelling. Finally, the modelling language itself should be simple, visual, and user-friendly.

The second brick being an evolution constraint language, it would be intended to constrict the evolution of data over time. To do so, new temporal types and operators should be supported. The proposed language should be able to traverse the model and time travel regardless of the time granularity used at design time. Finally, the constraint language should consider time as continuous rather than a discrete time scale. Optionally, the language may support the definition of events, actions, and notifications with different severity levels.

In Section 7, we present a brief example of how we imagine our modelling language and constraint evolution language. In future work, we plan to provide a concise semantics and syntax for both languages.

5. Temporal Data Representation and Storage

The interest on scalable model persistence has grown significantly in recent years. Several approaches have been proposed, each one relying on different persistence model and backend [29], [30], [31], [32], [33]. These approaches store only the latest state of the model. None is designed for storing temporal data. A notable exception is the work of [8], [34], which specifically discusses the lack of native mechanisms to efficiently support the notion of history and time in the context of MDE in general and models@run.time [35], [36] in particular. They propose kind of a delta storage based on key/value stores to efficiently persist the history of time-evolving models. This has been implemented and integrated into the open-source framework GreyCat[2]. Other research

2. GreyCat: http://greycat.ai/

efforts in MDE have concentrated on maintaining the history of the evolution of model-based artefacts. EMFStore [4] and CDO are some examples among others. They are designed to leverage modelling in collaborative environments and do not store the evolution of objects and attributes over time.

Data in MDE is represented in its essential form by Directed Typed Attributed Graphs. In this perspective, an intuitive representation to retain temporal information in MDE would be temporal graphs. Unluckily, existing temporal graph databases do not support storing typed graphs. A possible solution to cope with this, is to provide a mapping from temporal typed graphs to existing high-performance databases. Campos et al. [37] propose a mapping based on a graph database. This mapping differentiates between four kinds of nodes: object nodes, edge nodes, attribute nodes, and value nodes. Each node is identified by a UID, a name, and an interval in which the node is or was valid. Portal [38] represents a temporal graph using four SQL relations. Two valid-time relations are used to represent vertex and edges separately. The other two relations are used to represent vertex and edge attributes. In this section, we survey existing model-data representations in MDE and exhibit their limitations. Later, we discuss possible temporal model-data mappings in MDE.

5.1. Graph-data Representations in MDE

Several approaches have been proposed to store model-data in MDE on top of different kind of databases. In what follows, we present existing frameworks and data models.

5.1.1. Relational databases. Mapping data in MDE to relational databases is an elderly subject. CDO [32] is the de facto standard solution to handle large models in EMF. It relies on relational databases for mapping and storing models. It was initially envisioned as a framework to manage large models in a collaborative environment with a low memory footprint. CDO adopts a traditional UML2Relational mapping, where Classes and multivalued references are mapped to relations, class attributes are organised in columns in the corresponding relations, and finally, objects are represented by tuples. Unfortunately, the model-data mapping in CDO does not support the temporal dimension.

5.1.2. NoSQL databases. One of the good examples illustrating different model-data mappings to NoSQL databases is NeoEMF [39]. It is a multi-backend model persistence framework that couples state-of-the-art NoSQL stores. NeoEMF/Map [31] relies on a map-based data model to store model elements using a hashtable data structure. NeoEMF/Column [30] is designed to enable the development of distributed MDE-based applications by relying on a distributed column store. Finally, NeoEMF/Graph [29] uses GraphDB to store model-data in its natural form by means of an attributed labelled graph database. The model-data mapping in NeoEMF/Graph is straight forward, except for model elements type. It is represented through a relationship labelled InstanceOf towards a vertex representing the type. NeoEMF/Column uses a single table with three column families to store the information of the models (Type, Properties, and Containment). NeoEMF/Map uses a similar mapping to NeoEMF/Column where column families are represented as separate hashtables.

5.2. Further Development

Except GreyCat, none of the presented solutions does support storing typed temporal graphes. However, GreyCat is not EMF-compliant and relies on its own meta-modelling framework, thereby complicating the use of existing EMF-based tools. The reason for this disruptive change is the use of time as a first class entity, cross-cutting all model access. The pro and cons for this change is longer discussed into Section 7. Nonetheless, we believe that novel model-data mappings to existing temporal databases should be envisaged, by considering time as a special attribute. The Portal database can be an inspiration for CDO as both rely on SQL as an underlying data model. However, as pointed by existing work [31], implementing complex algorithms atop relational databases fails drastically due to the expensive cost of join operators. Likewise, NeoEMF/Graph and NeoEMF/Column model-data mapping can get inspired by existing work [13], [37] and propose a novel model-data mapping for supporting temporal dimension.

The proposed mapping should be tailored in a way that guarantees good performance while carrying out common temporal graph operations (Section 6). Moreover, novel data caching and indexing techniques adequate for storing temporal graphs should be investigated. Indeed, the temporal dimension exhibits a new layout other than the structural structure locality. Instead of having connected elements stored in the same partition, which is good for performing graph traversals on the same snapshot, one may be interested instead in storing consecutive timepoints, which is good for performing time travels. Also, graph data compaction techniques can be thought of to reduce the amount of persisted data. Finally, new means to automatically generate APIs that are adapted for manipulating temporal models should be provided. In future work, we plan to extend existing persistence frameworks such as NeoEMF with capabilities to store temporal data and generate adequate API to query it. Conversely, we also plan to continue the development of GreyCat to bring closer traditional meta-modelling techniques with meta-model with time as first class entity.

6. Temporal Data Processing in MDE

The most important step in a data integration projects is the analysis and the processing of collected data to extract valuable knowledge. A very known activity-example in temporal data analysis is temporal data mining. This activity seeks to exhibit temporal patterns in time evolving data. Typical tasks involved in temporal data mining include temporal clustering, temporal prediction, temporal pattern analysis. These tasks, as well as other temporal ones, involve iterative interaction with temporal data stores before acquiring the desired knowledge. Although languages for querying temporal data exist, they are not adapted for expressing temporal graph traversals and temporal graph matching. In particular, most existing languages are SQL-like [37], [38]. Expressing graph queries with such a family of languages is not transparent, and requires application developers to be aware of the underlying graph-data mapping. Moreover, SQL-like languages have a heavy aggregation syntax that results in unmanageable queries.

Furthermore, recent work [40] argue that, even though commonly-used graph algorithms and operations such as depth-first search and breadth-first search are well-defined for static graphs, they are non-trivial when considering the temporal dimension. We believe that by providing a temporal graph querying language, we would simplify the development of new temporal graph algorithms and operations in an easy and intuitive manner. In this section, we investigate existing temporal graph query languages and identify their limitations.

6.1. Temporal query languages

The most predominant language style for querying temporal databases is SQL-like. Inspired by the recent extension to SQL:2011 (TSQL2), existing temporal query languages supply different kinds of temporal databases with means to effectively retrieve and process data. TSQL2 introduces three temporal types to query data, date-time, period, and interval. The first one corresponds to a time t, without duration. The second one corresponds to a set of consecutive snapshots at a precise period, identified by two boundaries, while the last one corresponds to a duration, which is not anchored in time axis. TSQL2 provides a syntactic extension to SQL statements that let users specify the period of interest. Intervals are used inside the *where* clauses. However, the period columns (*valid_from* & *valid_to*) should be explicitly mentioned. TSQL2 is also shipped with period predicates for expressing conditions involving one or more time interval such as *contains*, *overlaps*, *equals*, etc..

Campos et al. [37] introduce TEG-QL, a graph query language inspired by SQL and Cypher [41]. The *from* clause contains the pattern to be matched, in the form of one or more paths. The *select* indicates paths or attributes to be returned. The *from* clause is used to retrieve one or more paths, over which a selection is performed, while the *where* expresses the filtering predicate. TEG-QL defines two temporal modifiers, *snapshot* and *in*. While *snapshot* enables the slicing of the results in a specific time granularity, the *in* modifier enables the specification of a time interval where attribute values, nodes, and edges are valid. TSQL2 is not well-suitable for expressing temporal pattern matching. And both languages fail to express graph traversal. Portal [38] proposes a powerful API to query and process temporal graphs. It relies on a graph algebra, TGraph, that extends relational algebra by specifying how temporal graph operators are applied to temporal relations. In particular, TGraph introduces the **slice** operator, which is responsible for cutting a temporal slice from a TGraph w.r.t. a time period. TGraph comes with a set of aggregation operators such as *avg*, *sum*, over time-evolving values. Portal is well-suited for defining temporal graph processing operation, however, it is not suitable for processing typed temporal graphs.

6.2. Further Development

Many graph processing frameworks exploited the strong emergence of systems and programming models for distributed and parallel processing to leverage the processing of big data. Some of these frameworks are provided with high-level declarative languages designed for specific applications such as data warehousing, querying,etc.. Unfortunately, none of these languages is suitable for temporal graphs queries and traversals over typed attributed graphs.

We argue that a declarative high-level language for querying and traversing temporal graphs is of big importance. Except for Portal, existing languages are not suitable for expressing temporal graph algorithms. The language should enable the expression of temporal pattern matching and temporal graph traversal, as they are keystones for expressing temporal graph algorithms. Moreover, SQL-like temporal languages express time constraints only in the *where* clause. Expressing two distinct time travel expressions (or more) over two different subgraphs within the same graph traversal, requires two (or more) nested select expressions. This results in a cumbersome query expression. The proposed language should flawlessly enable moving back from the temporal dimension to the structural one, and vice-versa. The language should support temporal aggregation operations, grouping data by time or by structure, and may express queries that return time intervals. Finally, the language should embed a user-friendly notation to express different granularities within the same query. Performance-wise, the language should deliver efficiency both in terms of time and memory consumption. We can rely on indexing and caching techniques provided by the persistence framework to faster query evaluation, and data compacton techniques to enable a low memory footprint.

7. Discussion

7.1. Towards a Time-aware Modelling Language

Listing 1 shows an example of how a temporal smart grid metamodel (as described in Section 2) could be defined using an imagined textual modelling language, which allows defining temporal properties. The language used in this example is inspired by existing work [9]. The particularity of this language is the integration of *temporal* annotations, namely **temporalSensitivity**, **temporalPeriodicity**, **temporalConstraint**, and **precision**. With these annotations, attributes and relationships can—in a declarative way—be extended with temporal semantic. For example, in the *SmartMeter* metaclass, the attribute *activeEnergyConsumed* is decorated with an annotation *temporalSensitivity*, which declares a granularity of 15 minutes. Based on this definition, all consumptions values would only be stored every 15 minutes, regardless of the pace of measurements of the real sensor. All values in-between would be averaged. Similarly, the concentrator relationship is annotated with a granularity of 1 second in order

Listing 1 A temporal smart grid metamodel

```
class Entity {}

class SmartMeter extends Entity {
  @temporalSensitivity(15.MINUTE)
  att activeEnergyConsumed: Double

  @temporalSensitivity(100.MILLISECOND)
  @temporalPeriodicity(24.HOUR)
  @precision(0.1)
  att voltage: Double

  @temporalSensitivity(1.SECOND)
  rel concentrator: Concentrator
  @temporalSensitivity(1.MONTH)
  rel customer: Customer }
  [...]
class Cable {
  [...]
  @temporalConstraint(if self.isOverloaded then
self.isOverloaded.maxDuration(15.MINUTE) = true
endif )
  @temporalSensitivity(1.SECOND)
  att isOverloaded: Boolean }
```

to be able to track its changes in near real-time. The flexibility of such annotation mechanism enables a metamodel design that mixes data which evolves at different paces.

The *temporalPeriodicity* annotation has no effect on storage, however, it allows reasoning engines to optimise their checks according to the expected periodicity of changes. Following this idea, we could declare that voltage attributes can be compared daily. As for the annotation *precision*, it mixes temporal and domain information. Used on the voltage attribute, it describes the fact that temporal variations are meaningless if not greater than a threshold value. In other words, it specifies that the voltage should be stored only if a variation of more than 0.1 volts is measured. Under this threshold, values are ignored. Such specification allows the model to automatically filter insignificant data based on experts knowledge. Finally, the annotation *temporalConstraint* is used to specify evolution constraints over attributes or relationships. For example, in the metaclass *Cable*, the temporal constraint over the attribute *isOverloaded*, says that a Cable cannot be overloaded for more than 15 minutes. The *temporalConstraint* uses a conditional expression (if–then–endif) to check that the attribute value *isOverloaded* has been equal to true for less than 15 minutes. The operation *maxDuration(15.MINUTE)* returns true if the current value has been valid for less than 15 minutes.

7.2. Time as a First-Class Entity

Today, OCL [26] is still the de-facto standard in the modelling community for describing functional and non-functional properties in form of metamodel extensions. OCL follows one of the main principles of model-driven engineering, the separation of concerns between structure and behaviour definition. In the past, various extensions of OCL have been proposed, e.g. [28] and more recently [42] [3], to add support for temporal constraints. All of this work have in common that they propose to use temporal OCL in order to check the evolution of model instances over time. In this paper, we envisaged the use of time as a fully functional property, which can enable to define the temporal behaviour of a system over time. For the sake of simplicity and to introduce temporal properties as smooth as possible, we suggest in this paper to define temporal behaviour as a cross-cutting annotation in an OCL-like style. We envisage the structure not as a static canvas for data, but as an entry to dynamically evolving data. In other words, an attribute with temporal semantic does not have a value, but one value for every given point in time. This can be compared with time series [43], which lately raised lots of attention.

On the other hand, we also discussed the idea to consider time as a first-class property, cross-cutting any model element, i.e. every element in a model, as suggested by Hartmann et al. [34]. In this way, every model element would always have a temporal semantic and would be able to independently evolve over time. This profound shift from static modelling to temporal aware modelling enforces temporal considerations for every element. Traditional modelling concepts could be seamlessly mixed with temporal definitions, such as **@temporalSensivity(-1)** which could mean that every temporal variation will be stored in a single timepoint. This shift has already been engaged in the database community, which already defines the notion of temporal graphs, where every node has a time attribute [6], [15], [37]. Our hypothesis is that temporal knowledge is part of a domain itself and we believe that MDE and its tooling ecosystem has the potential to pave the way to more structured, typed, and safe temporal data management systems.

8. Conclusion

In this paper we have argued about the need for raising time awareness in MDE. In particular, we discussed that raising time awareness in MDE involves, at least, the integration of the temporal dimension in an orthogonal and seamless manner, the scalable persistence of historical data, finally, the ability to intuitively query and process this data. We investigated the state of the art in these areas. We exhibited the missing points, then we pointed to some research direction towards achieving time awareness in MDE. Finally, we gave a quick overview of how we imagine a time-aware modelling language as well as an evolution constraint language.

In future work, we plan to provide a full support for these two languages and we intend to provide an adequate persistence framework and an expressive query language to enable the development of temporal graph algorithms and operations.

References

[1] J. E. Rivera, J. R. Romero, and A. Vallecillo, “Behavior, time and viewpoint consistency: Three challenges for mde,” in *International*

Conference on Model Driven Engineering Languages and Systems. Springer, 2008, pp. 60–65.

[2] E. Bousse, J. Corley, B. Combemale, J. Gray, and B. Baudry, "Supporting efficient and advanced omniscient debugging for xdsmls," in *Proc. of the 8th Int. Conf. on SLE*. ACM, 2015, pp. 137–148.

[3] B. Kanso and S. Taha, "Temporal constraint support for ocl," in *Proc. of the 5th Int. Conf. on SLE*. Springer, 2012, pp. 83–103.

[4] M. Koegel and J. Helming, "Emfstore: a model repository for emf models," in *Proc. of the 32nd ACM/IEEE Int. Conf. on Software Engineering-Volume 2*. ACM, 2010, pp. 307–308.

[5] J. E. Rivera, A. Vallecillo, and F. Durán, "e-motions: A graphical approach for modeling timedependent behavior of domain specific languages," 2009.

[6] A. P. Iyer, L. E. Li, T. Das, and I. Stoica, "Time-evolving graph processing at scale," in *Proc. of the 4th Int. Workshop GRADES*. ACM, 2016, pp. 5:1–5:6.

[7] W. Han, Y. Miao, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, W. Chen, and E. Chen, "Chronos: A graph engine for temporal graph analysis," in *Proc. of the 9th EuroSys Conf.* ACM, 2014, pp. 1:1–1:14.

[8] T. Hartmann, F. Fouquet, G. Nain, B. Morin, J. Klein, and Y. L. Traon, "Reasoning at runtime using time-distorted contexts: A models@run.time based approach," in *The 26th Int. Conf. SEKE*, Jul. 2014, pp. 586–591.

[9] T. Hartmann, "Enabling model-driven live analytics for cyber-physical systems: The case of smart grids," Ph.D. dissertation, University of Luxembourg, 2016.

[10] T. Hartmann, F. Fouquet, J. Klein, Y. L. Traon, A. Pelov, L. Toutain, and T. Ropitault, "Generating realistic smart grid communication topologies based on real-data," in *2014 IEEE Int. Conf. on SmartGridComm*, Nov 2014, pp. 428–433.

[11] J. Clifford and D. S. Warren, "Formal semantics for time in databases," *ACM Trans. Database Syst.*, vol. 8, no. 2, pp. 214–254, Jun 1983.

[12] E. Rose and A. Segev, "Toodm: A temporal object-oriented data model with temporal constraints," Lawrence Berkeley Lab., CA (United States), Tech. Rep., 1991.

[13] M. Kaufmann, A. A. Manjili, P. Vagenas, P. M. Fischer, D. Kossmann, F. Färber, and N. May, "Timeline index: A unified data structure for processing queries on temporal data in sap hana," in *Proc. of the 2013 Int. Conf. on Management of Data*. ACM, 2013, pp. 1173–1184.

[14] V. Kostakos, "Temporal graphs," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.

[15] W. Han, Y. Miao, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, and E. Chen, "Chronos: a graph engine for temporal graph analysis," in *Proc. of the 9th Conf. on Comp. Sys.* ACM, 2014, pp. 1–14.

[16] Y. Miao, W. Han, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, E. Chen, and W. Chen, "Immortalgraph: A system for storage and analysis of temporal graphs," *Trans. Storage*, vol. 11, no. 3, pp. 14:1–14:34, Jul. 2015.

[17] U. Khurana and A. Deshpande, "Storing and analyzing historical graph data at scale," in *Proc. of the 19th Int. Conf. on EDBT, Bordeaux, France*, Mar 2016, pp. 65–76.

[18] R. Cheng, J. Hong, A. Kyrola, Y. Miao, X. Weng, M. Wu, F. Yang, L. Zhou, F. Zhao, and E. Chen, "Kineograph: Taking the pulse of a fast-changing and connected world," in *Proc. of the 7th ACM EuroSyS*. ACM, 2012, pp. 85–98.

[19] N. Ferry, F. Chauvel, A. Rossini, B. Morin, and A. Solberg, "Managing multi-cloud systems with cloudmf," in *Proc. of the 2th Nordic Symposium NordiCloud*. ACM, 2013, pp. 38–45.

[20] C. Vidal, C. Fernández-Sánchez, J. Díaz, and J. Pérez, "A model-driven engineering process for autonomic sensor-actuator networks," *Int. Journal of Distributed Sensor Networks*, vol. 11, no. 3, 2015.

[21] N. Harrand, F. Fleurey, B. Morin, and K. E. Husa, "Thingml: a language and code generation framework for heterogeneous targets," in *Proc. of the 19th Int. Conf. on MODELS*. ACM, 2016, pp. 125–135.

[22] S. Spaccapietra, C. Parent, and E. Zimanyi, "Modeling time from a conceptual perspective," in *Proc. of the 7th ACM Int. Conf CIKM*. ACM, 1998, pp. 432–440.

[23] A. Artale, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev, "A cookbook for temporal conceptual data modelling with description logics," *ACM Trans. on Comp. Logic*, vol. 15, no. 3, pp. 1–50, 2014.

[24] A. Artale, C. Parent, and S. Spaccapietra, "Evolving objects in temporal information systems," *Annals of Mathematics and Artificial Intelligence*, vol. 50, no. 1, pp. 5–38, 2007.

[25] H. Gregersen and C. S. Jensen, "Temporal entity-relationship models-a survey," *IEEE TKDE*, vol. 11, no. 3, pp. 464–497, 1999.

[26] Object Management Group, "Object Constraint Language, OCL," April, 2017, URL: http://www.omg.org/spec/OCL/.

[27] A. Hamie, R. Mitchell, and J. Howse, "Time-based constraints in the object constraint language," Technical Report CMS-00-01, University of Brighton, Tech. Rep., 2000.

[28] S. Conrad and K. Turowski, "Temporal ocl: Meeting specification demands for business components," *Unified modeling language: Systems analysis, design and development issues*, pp. 151–166, 2001.

[29] A. Benelallam, A. Gómez, G. Sunyé, M. Tisi, and D. Launay, "Neo4EMF, A Scalable Persistence Layer for EMF Models," in *In Proc. of the 10th European Conf. on ECMFA*. Springer, 2014, pp. 230–241.

[30] A. Gómez, A. Benelallam, and M. Tisi, "Decentralized Model Persistence for Distributed Computing," in *Proc. of 3rd BigMDE Workshop*, vol. 1406. CEUR Workshop Proc., July 2015.

[31] A. Gómez, M. Tisi, G. Sunyé, and J. Cabot, "Map-based transparent persistence for very large models," in *In Proc of the Int. Conf. on FASE*. Springer, 2015, pp. 19–34.

[32] "CDO Model Repository," 2014. [Online]. Available: http://www.eclipse.org/cdo/

[33] K. Barmpis and D. Kolovos, "Hawk: Towards a Scalable Model Indexing Architecture," in *Proc. of the Workshop on Scalability in Model Driven Engineering*. ACM, 2013, p. 6.

[34] T. Hartmann, F. Fouquet, G. Nain, B. Morin, J. Klein, O. Barais, and Y. L. Traon, "A native versioning concept to support historized models at runtime," in *The 17th Int. Conf. MODELS, Valencia, Spain, September 28 - October 3, 2014. 2014*, 2014, pp. 252–268.

[35] G. Blair, R. B. France, and N. Bencomo, "Models@ run.time," *Computer*, vol. 42, pp. 22–27, 2009.

[36] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@ run.time to support dynamic adaptation," *Computer*, vol. 42, no. 10, pp. 44–51, Oct. 2009.

[37] A. Campos, J. Mozzino, and A. Vaisman, "Towards temporal graph databases," *arXiv preprint arXiv:1604.08568*, 2016.

[38] V. Z. Moffitt and J. Stoyanovich, "Towards a distributed infrastructure for evolving graph analytics," in *Proc. of the 25th Int. Conf. on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*, 2016, pp. 843–848.

[39] G. Daniel, G. Sunyé, A. Benelallam, M. Tisi, Y. Vernageau, A. Gómez, and J. Cabot, "NeoEMF: a Multi-database Model Persistence Framework for Very Large Models," in *Proc. of the 19th Int. Conf. MoDELS, (Demo track)*, Oct 2016, pp. 1–7.

[40] S. Huang, J. Cheng, and H. Wu, "Temporal graph traversals: Definitions, algorithms, and applications," *arXiv preprint*, 2014.

[41] Neo4j Corp., "Cypher," April, 2017, URL: https://www.neo4j.com/.

[42] W. Dou, D. Bianculli, and L. Briand, "OCLR: A More Expressive, Pattern-Based Temporal Extension of OCL," in *In Proc. of the 10th European Conf. ECMFA*. Springer Int. Publishing, 2014, pp. 51–66.

[43] C. Chatfield, *The analysis of time series: an introduction*. CRC press, 2016.