Two-Dimensional Rule Language for Querying Sensor Log Data: a Framework and Use Cases

S. Brandt[1], D. Calvanese[2], E. Güzel Kalaycı[2], R. Kontchakov[3], B. Mörzinger[4], V. Ryzhikov[3], G. Xiao[2], and M. Zakharyaschev[3]

[1] Siemens CT, München, Germany,
[2] Free University of Bozen-Bolzano, Italy
[3] Birkbeck, University of London, U.K.
[4] Technische Universität Wien, Austria

Abstract. Motivated by two industrial use cases that involve detecting events of interest in (asynchronous) time series from sensors in manufacturing rigs and gas turbines, we design an expressive rule language *DslD* equipped with interval aggregate functions (such as weighted average over a time interval), Allen's interval relations, and various metric constructs. We demonstrate how to model events in the uses cases in terms of *DslD* programs. We show that answering non-recursive *DslD* queries can be reduced to evaluating SQL queries. Our experiments with the use cases, carried out on the Apache Spark system, show that such SQL queries scale well on large real-world datasets.

Keywords: Rule language · temporal logic · sensor log data.

1 Introduction

Our concern in this paper is spotting events a domain expert is interested in among time-stamped sensor log data stored in a database.

The amount of sensor data produced in all areas of modern life is proliferating with a tremendous speed, probably even faster than the volume of social media data. In the industrial sector, in particular manufacturing, sensor data is crucial for monitoring, control, various types of analytics (descriptive, predictive and prescriptive), decision-making as well as research aiming to improve processes and products. Sensor data is a key to the 'fourth industrial revolution' in global manufacturing [30, 22], which is witnessed by the German Industrie 4.0 programme and the smart manufacturing initiatives in the United States.

The scenario we are interested in is querying historical (rather than streaming) sensor data stored in a database with the aim of detecting temporal events that can help to explain, predict and improve the behaviour of the system in question. A typical example already considered in [17, 29, 18, 6] is querying the data from gas turbine sensors measuring the active power, rotor speed, blade temperature, etc. and stored at Siemens remote diagnostic centres, where service engineers are looking for events such as

active power trip, which happens at a moment when the active power was above 1.5MW for a period of at least 10 seconds, maximum 3 seconds after which there was a period of at least one minute where the active power was below 0.15MW; to avoid short-term fluctuations, the value of active power should be smoothed out by taking the simple moving average of the raw power measurements over the last 5 seconds' window.

However, accessing the data generated by sensors and finding the events of interest is hard because the engineers and researchers with domain expertise necessary to interpret the data usually lack the knowledge required to independently interact with databases and formalise temporal queries such as the one in the example above. Following the current workflow at Siemens, the engineer usually asks an IT specialist to produce a custom-made script (in a proprietary signal processing language) such as

$$
\begin{aligned}
&\texttt{message}(\text{“}\texttt{active power trip}\text{”}) = \\
&\quad \$t1 : \texttt{eval}(>, \mathit{simpleMovingAverage}(\#\texttt{activePower}, 5s), 1.5) : \\
&\qquad \texttt{for}(>= 10s) \\
&\qquad \&\& \\
&\qquad \texttt{eval}(<, \mathit{simpleMovingAverage}(\#\texttt{activePower}, 5s), 0.15) : \\
&\qquad \texttt{start}(\texttt{after}[0s, 3s]\$t1 : \texttt{end}) : \\
&\qquad \texttt{for}(>= 1m);
\end{aligned}
$$

and run it over pre-processed, that is, properly aggregated and smoothed out sensor data.

A fundamentally different approach is offered by ontology-based data access and management [26, 23, 32] (OBDA, for short), a semantic technology that has been developed over the past decade with the aim of facilitating access to various types of data. The role of ontologies in OBDA is threefold: to integrate distributed and heterogeneous data sources, enrich incomplete data with background knowledge, and provide a user-friendly and familiar language for querying. Unfortunately, however, the existing standard ontology languages are not able to represent temporal events, let alone those that depend on aggregated numerical data. Extending the OBDA paradigm to temporal data has recently become an active research area in semantic technologies [25, 16, 10, 3, 5, 19, 2, 9, 15]. For example, the active power trip event (without smoothing the active power measurements) was captured in the language *datalogMTL*, a combination of metric temporal logic *MTL* with datalog [6]; see (2) in Section 3 below. *MTL* for signal monitoring was also used in [24].

One of the aims of this paper is to design an ontology language that could capture interval aggregate functions such as simple moving average together with various metric constructs. Aggregates in the context of OBDA have been considered in [8, 19, 21], where they were only allowed in queries, and so again an IT middleman may be needed to help the user; [4] introduced description logics with aggregation features over concrete domains and showed that reasoning with

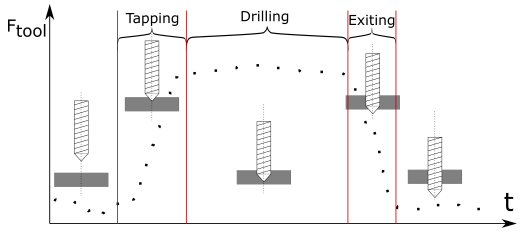


Fig. 1. A simplified course of force measurements for a drilling process.

them is often undecidable. A first-order aggregate logic was investigated in [13]. For datalog with monotonic aggregates, see [28] and references therein.

The second real-world use case motivating our language concerns the experimental investigation of drilling processes [14]. In manufacturing research, large numbers of experimental drilling processes, such as the one in Fig. 1, are executed using a wide range of different tools and process parameters. Sensors are used to surveil these processes and generate data, which is then analysed in order to identify, e.g., ways to reduce tool wear, maximise productivity and ensure product quality. To achieve these goals, researchers define intervals of interest (say, processes leading to quality anomalies or those that were interrupted by tool breaks) within such readings and use them to select data for further analysis.

An essential difference from the turbine use case is that, in contrast to the active power trip event described as a sequence of timestamp-value pairs satisfying explicit numerical bounds on their values, now we are looking for certain sequences of *shapes* such as an interval (tapping) when the force is increasing followed by an interval (drilling) when the force is stable and then by an interval (exiting) when the force is decreasing. The duration of each of these intervals and the force values may vary widely due to different combinations of workpiece materials, tools, process parameters and tool wear.

To tackle this problem, we further extend our language with the Allen interval relations [1] such as *after* (A), *begins* (B), etc., using which we can capture the normal drilling event by the following rule:

$$\mathit{NormalDrilling}(\boldsymbol{x}) \leftarrow \mathit{Tapping}(\boldsymbol{x}_1), \mathit{Drilling}(\boldsymbol{x}_2), \mathit{Exiting}(\boldsymbol{x}_3), \\ \boldsymbol{x}_1 \,\mathsf{A}\, \boldsymbol{x}_2,\ \boldsymbol{x}_2 \,\mathsf{A}\, \boldsymbol{x}_3,\ \boldsymbol{x}\,\mathsf{is}\,(\boldsymbol{x}_1 \uplus \boldsymbol{x}_3 \uplus \boldsymbol{x}_3),$$

where $\boldsymbol{x}$ and the $\boldsymbol{x}_i$ are intervals over the real numbers, $\uplus$ returns the union of connected intervals in case it is also a (connected) interval, and the predicates $\mathit{Tapping}(\boldsymbol{x}_1)$, $\mathit{Drilling}(\boldsymbol{x}_2)$ and $\mathit{Exiting}(\boldsymbol{x}_3)$ involve complex interval aggregation to smooth out the fluctuating measurements of force and ensure that it is increasing, stable and decreasing, respectively. An OBDA ontology language, combining datalog with a fragment of the Halpern-Shoham logic *HS* [12], which uses modal operators interpreted by Allen's relations, was introduced in [20, 7].

In this paper, motivated by the sufficiently representative use cases outlined above, we propose a framework of rule-based languages for ontology-mediated

queries over sensor log data stored in a database. The framework, provisionally called *DslD* (datalog for sensor log data), contains unary predicates for representing events over temporal intervals and binary predicates for capturing numerical sensor measurements over temporal intervals and also the results of aggregation. Thus, *DslD* is essentially two-dimensional, with a temporal dimension comprising intervals over the real numbers $\mathbb{R}$ and a (measurement) value dimension $\mathbb{R}$. *DslD* also contains built-in predicates for expressing quantitative constraints on intervals and values, and the Allen interval relations for representing qualitative constraints. Finally, *DslD* allows one to define built-in interval aggregation operators which, given a (functional) relation, say representing measurements, would return their moving or weighted average, compute its coalescing (maximal intervals with the same value), etc. Our goal here is to check experimentally, using the two real-world scenarios and data, whether ontology-mediated queries formulated in *DslD* can be evaluated efficiently by standard database engines.

The structure of the paper is as follows. In Section 2, we define the syntax and (procedural) semantics of *DslD*. In Sections 3 and 4, we write *DslD* programs for the turbine and drilling use cases. In Section 5, we show how to reduce answering queries mediated by non-recursive *DslD* programs to evaluation of SQL queries. In Section 6, we report on testing such SQL translations in our use cases. Finally, in Section 7, we conclude and describe future work.

2 Syntax and Semantics of *DslD*

We begin by defining the syntax of the ontology language *DslD*, *datalog for sensor log data*, which is designed to represent data and knowledge about events in sensor-based systems. As the main temporal entity we take the notion of *interval* over the set $(\mathbb{R}, <)$ of real numbers. More precisely, an *interval*, ι, is any nonempty subset of $\mathbb{R}$ of the form $\langle t_1, t_2 \rangle$, where $t_1, t_2 \in \mathbb{R} \cup \{-\infty, \infty\}$, '$\langle$' is '(' or '[' and '$\rangle$' is ')' or ']'. Note that we admit *punctual* intervals of the form $[t, t]$. We denote by $\mathsf{int}_{\mathbb{R}}$ the set of all intervals over $\mathbb{R}$ and by $|\iota|$ the *length* of an interval ι. Sensor measurements are deemed to be real numbers. We write $R(\iota, v)$ to say that $v \in \mathbb{R}$ is the *value* measured by sensor R over the interval $\iota \in \mathsf{int}_{\mathbb{R}}$, and we write $A(\iota)$ to say that event A occurs in the interval ι. Thus, *DslD* has unary and binary predicate symbols only. Let $A_1, A_2, \dots$ be a list of unary predicate symbols and $R_1, R_2, \dots$ a list of binary predicate symbols in *DslD*; we refer to them as *signature predicates*.

Interval and Value Terms. We distinguish between two sorts of variables and terms. The variables $\boldsymbol{x}, \boldsymbol{y}, \dots$ of sort *interval* range over $\mathsf{int}_{\mathbb{R}}$, while the variables $x, y, \dots$ of sort *value* range over $\mathbb{R}$. *Constants* of sort interval are concrete intervals with rational end-points (from $\mathbb{Q} \cup \{-\infty, \infty\}$). The language contains various (partial) *functions* of sort interval, which include the intersection $\boxplus$ and union $\uplus$ of intervals defined by taking

$$\iota \boxplus \iota' = \begin{cases} \iota \cap \iota', & \text{if } \iota \cap \iota' \neq \emptyset, \\ \text{undefined}, & \text{otherwise;} \end{cases} \qquad \iota \uplus \iota' = \begin{cases} \iota \cup \iota', & \text{if } \iota \cap \iota' \neq \emptyset, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

It also includes the length $|\iota|$ of ι, which is undefined for unbounded ι; the *shift* functions such as lshift_r and rshift_r, for $r \in \mathbb{Q}$, that shift the left and, respectively, right end of a given interval by r (for example, lshift_{-1} maps every interval $\iota = \langle b, e\rangle$ to $\mathit{lshift}_{-1}(\iota) = \langle b-1, e\rangle$); the functions lext_r and rext_r that extend the left (respectively, right) end of ι to an interval $\langle b, b+r\rangle$ (respectively, $\langle e-r, e\rangle$); and possibly other useful operations on intervals. *Terms* of sort interval are built from variables and constants of sort interval using these functions. Likewise, *constants* of sort value are rational numbers; *functions* of sort value include standard $x+y$, $x-y$, $x \times y$, $|x|$, etc.; *terms* of sort value are built from variables and constants of sort value using functions of sort value.

Predicates. Apart from the signature predicates, the language of *DslD* contains a wide range of *built-in predicates*. To begin with, we have the standard $=$, $\neq$, $<$, $\leq$, etc. over $\mathbb{R}$. Next, *DslD* features the Allen relations [1] over $\mathsf{int}_{\mathbb{R}}$: *after* ($\mathsf{A}$), *begins* ($\mathsf{B}$), *ends* ($\mathsf{E}$), *during* ($\mathsf{D}$), *later* ($\mathsf{L}$), *overlaps* ($\mathsf{O}$) and their inverses $\bar{\mathsf{A}}$, $\bar{\mathsf{B}}$, etc. Finally, *DslD* has unary built-in *discretisation predicates* $D^{\langle\rangle}_{s,b}(\boldsymbol{x})$, for $s, b \in \mathbb{Q}$, which comprise all intervals of the form $\langle s+ib, s+(i+1)b\rangle$, for $i \in \mathbb{Z}$, and which are particularly useful with aggregation; see examples in Section 4.

Aggregation Functionals. An *aggregation functional* is a function over binary relations R on $\mathsf{int}_{\mathbb{R}} \times \mathbb{R}$. For example, denote $R_\iota = \{(\iota', v') \in R \mid \iota' \cap \iota \neq \emptyset\}$, for a relation R and an interval $\iota \in \mathsf{int}_{\mathbb{R}}$, and let

$$\mathit{max}(R) = \big\{\big(\iota, \max_{(\iota',v')\in R_\iota} v'\big) \mid \iota \in \mathsf{int}_{\mathbb{R}} \text{ and } \iota \subseteq \mathit{dom}\, R\big\},$$

where $\mathit{dom}\, R = \bigcup_{(\iota,v)\in R} \iota$ is the *domain* of R. This aggregation functional returns a relation with the maximum value of R on any subinterval of its domain. Another typical example is the *simple moving average*:

$$\mathit{sma}(R) = \big\{\big(\iota, \frac{1}{|R_\iota|} \sum_{(\iota',v')\in R_\iota} v'\big) \mid \iota \in \mathsf{int}_{\mathbb{R}}, \iota \subseteq \mathit{dom}\, R \text{ and } R_\iota \text{ is finite}\big\}.$$

To define other aggregation functionals, we require the following definition. We say that R is a *functional relation* if $v_1 = v_2$, for any $(\iota_1, v_1), (\iota_2, v_2) \in R$ with $\iota_1 \cap \iota_2 \neq \emptyset$. In this case, R gives rise to the function $f_R\colon\ \mathit{dom}\, R \to \mathbb{R}$ defined by taking

$$f_R(t) = v, \text{ for all } t \in \iota \text{ with } (\iota, v) \in R.$$

The *weighted average* is defined on functional relations R as

$$\mathit{wavg}(R) = \big\{\big(\iota, \frac{1}{|\iota|}\int_\iota f_R(x)\,dx\big) \mid \iota \in \mathsf{int}_{\mathbb{R}} \text{ is bounded and } \iota \subseteq \mathit{dom}\, R\big\},$$

and $\mathit{wavg}(R) = \emptyset$ for non-functional relations R. Note that, since $\mathit{wavg}(R)$ is defined on all subintervals of $\mathit{dom}\, R$, it is *not* a functional relation in general.

Another important aggregation functional is *coalescing*, which is defined on functional relations R by taking

$$\mathit{coalesce}(R) = \big\{(\iota, v) \mid \iota \in \mathsf{int}_{\mathbb{R}} \text{ is maximal with } \iota \subseteq \mathit{dom}\, R \text{ and } f_R(x) = v, \text{ for all } x \in \iota\big\},$$

and $\mathit{coalesce}(R) = \emptyset$ for non-functional R. Note that $\mathit{coalesce}(R)$ is functional and $f_R = f_{\mathit{coalesce}(R)}$. For unary predicates A over $\mathsf{int}_{\mathbb{R}}$, we define coalescing by taking $\mathit{coalesce}(A) = \{\iota \mid (\iota, 0) \in \mathit{coalesce}(A \times \{0\})\}$.

***DslD* Programs and Data Instances.** By a *built-in atom* we mean any well-formed atomic formula with a built-in predicate in the two-sorted language outlined above (excluding the aggregation functionals). An *assignment* atom is an expression $\boldsymbol{x} \,\mathsf{is}\, \boldsymbol{t}$ or $x \,\mathsf{is}\, t$, where $\boldsymbol{x}$ is a variable and $\boldsymbol{t}$ a term of sort interval, and x a variable and t a term of sort value. *Pure atoms* take the form $A(\boldsymbol{x})$ and $R(\boldsymbol{x}, y)$, and *aggregate atoms* are defined recursively as

$$\mathit{agg}\{(\boldsymbol{x}, y) \mid \alpha_1, \dots, \alpha_k\}, \quad \mathit{coalesce}\{\boldsymbol{x} \mid \alpha_1, \dots, \alpha_k\},$$

where *agg* is an aggregation functional and the α_i are pure, built-in, assignment or aggregate atoms. We call $(\boldsymbol{x}, y)$ and, respectively, $\boldsymbol{x}$ the *head* of the aggregate atom and assume that the variables $\boldsymbol{x}$ and y (respectively, $\boldsymbol{x}$) occur in $\alpha_1, \dots, \alpha_k$, where by 'occur in an aggregate atom' we mean occur in its head.

A *rule* in *DslD* is an expression of the form

$$\alpha \leftarrow \beta_1, \dots, \beta_n, \tag{1}$$

where α is a *pure atom* and the β_i are atoms. As usual in datalog, we call α the *head* of the rule and $\beta_1, \dots, \beta_n$ its *body*. We impose the following constraints on the rules: (*i*) every variable that occurs in a built-in predicate or on the right-hand side of an assignment must occur in a signature atom or the head of an aggregate atom in the body of the rule; (*ii*) each variable z in the head of (1) must be *guarded* in its body in the sense that one of the following holds:

- z occurs in some β_i, which is a signature atom;
- z occurs in some β_i, which is an atom with a discretisation predicate;
- some β_i is of the form $z \,\mathsf{is}\, t$;
- $(\boldsymbol{x}, z)$ occurs in the head of a functional aggregate atom β_i (note that all the aggregates in our use cases are functional) and $\boldsymbol{x}$ is guarded in $\beta_1, \dots, \beta_n$;
- there is $\beta_i = \mathit{coalesce}\{z \mid \gamma_1, \dots, \gamma_m\}$ and z is guarded in $\gamma_1, \dots, \gamma_m$.

(These constraints are needed to avoid infinitely many instantiations of variables in rules.)

A *DslD* program, Π, is a finite set of rules. *Data atoms* take the form $A(\varrho)$ and $R(\varrho, c)$, where ϱ is a constant of type interval and c a constant of type value. A *data instance*, $\mathcal{D}$, is a finite set of data atoms.

Semantics. We next define the procedural semantics of *DslD* programs. By a *substitution*, $\mathfrak{s}$, we understand a map from the set of interval variables $\boldsymbol{x}$ to $\mathsf{int}_{\mathbb{R}}$ and from the set of value variables x to $\mathbb{R}$. The definition of $\mathfrak{s}$ is inductively extended to terms in a standard way. Note, however, that the extended $\mathfrak{s}$ is in general a partial function: for example, $\mathfrak{s}([0,1] \uplus [2,3])$ is undefined.

Let Π be a *DslD* program and $\mathcal{D}$ a data instance. We first set

$$A_i^0 = \{\iota \in \mathsf{int}_{\mathbb{R}} \mid A_i(\iota) \in \mathcal{D}\}, \quad R_i^0 = \{(\iota, v) \in \mathsf{int}_{\mathbb{R}} \times \mathbb{R} \mid R_i(\iota, v) \in \mathcal{D}\}, \quad i \geq 1,$$

and $\mathcal{I}^0 = (A_1^0, \dots, R_1^0, \dots)$. Suppose next inductively that $\mathcal{I}^\xi = (A_1^\xi, \dots, R_1^\xi, \dots)$, for an ordinal ξ, and $\mathfrak{s}$ is a substitution. We define a truth-relation $\mathcal{I}^\xi, \mathfrak{s} \models \beta$ as follows, where $\boldsymbol{t}, \boldsymbol{t}'$ are terms of type interval and t, t' terms of type value:

- $\mathcal{I}^\xi, \mathfrak{s} \models A(\boldsymbol{t})$ iff $\mathfrak{s}(\boldsymbol{t})$ is defined and $\mathfrak{s}(\boldsymbol{t}) \in A^\xi$;
- $\mathcal{I}^\xi, \mathfrak{s} \models R(\boldsymbol{t}, t)$ iff $\mathfrak{s}(\boldsymbol{t})$ and $\mathfrak{s}(t)$ are both defined and $(\mathfrak{s}(\boldsymbol{t}), \mathfrak{s}(t)) \in R^\xi$;
- $\mathcal{I}^\xi, \mathfrak{s} \models (\boldsymbol{t} \neq \boldsymbol{t}')$ if $\mathfrak{s}(\boldsymbol{t})$ and $\mathfrak{s}(\boldsymbol{t}')$ are both defined with $\mathfrak{s}(t) \neq \mathfrak{s}(t')$, and similarly for other standard built-in predicates and assignments;
- $\mathcal{I}^\xi, \mathfrak{s} \models D_{s,b}^{\langle\rangle}(\boldsymbol{t})$ iff $\mathfrak{s}(\boldsymbol{t}) = \langle s + ib, s + (i+1)b \rangle$, for some $i \in \mathbb{Z}$;
- $\mathcal{I}^\xi, \mathfrak{s} \models \mathit{agg}\{(\boldsymbol{x}, y) \mid \alpha_1, \dots, \alpha_k\}$ iff $(\mathfrak{s}(\boldsymbol{x}), \mathfrak{s}(y)) \in \mathit{agg}(R)$, where

$$R = \big\{(\mathfrak{s}'(\boldsymbol{x}), \mathfrak{s}'(y)) \mid \text{ there is } \mathfrak{s}' \text{ such that } \mathcal{I}^\xi, \mathfrak{s}' \models \alpha_j \text{ for all } j\big\};$$

- $\mathcal{I}^\xi, \mathfrak{s} \models \mathit{coalesce}\{\boldsymbol{x} \mid \alpha_1, \dots, \alpha_k\}$ iff $\mathfrak{s}(\boldsymbol{x}) \in \mathit{coalesce}(A)$, where

$$A = \big\{\mathfrak{s}'(\boldsymbol{x}) \mid \text{ there is } \mathfrak{s}' \text{ such that } \mathcal{I}^\xi, \mathfrak{s}' \models \alpha_j \text{ for all } j\big\}.$$

Given $\mathcal{I}^\xi$, we define $\mathcal{I}^{\xi+1} = (A_1^{\xi+1}, \dots, R_1^{\xi+1}, \dots)$ as follows. For each A_i, we set $A_i^{\xi+1}$ to be A_i^ξ together with all $\mathfrak{s}(\boldsymbol{x})$ such that Π contains $A_i(\boldsymbol{x}) \leftarrow \beta_1, \dots, \beta_n$ and $\mathcal{I}^\xi, \mathfrak{s} \models \beta_j$ for all j, $1 \leq j \leq n$; and analogously for $R_i^{\xi+1}$. For a limit ordinal ζ, we set $\mathcal{I}^\zeta = (A_1^\zeta, \dots, R_1^\zeta, \dots)$, where $A_i^\zeta = \bigcup_{\xi<\zeta} A_i^\xi$ and $R_i^\zeta = \bigcup_{\xi<\zeta} R_i^\xi$. The construction is terminated when $\mathcal{I}^\xi = \mathcal{I}^{\xi+1}$, which should happen for some ξ not exceeding the smallest ordinal, $2^{\aleph_0+}$, whose cardinality is greater than $2^{\aleph_0}$.

Example 1. Consider the data instance $\mathcal{D} = \{A([0,1])\}$ and a *DslD* program Π with the following rules:

$$\begin{aligned} A(\boldsymbol{x}) &\leftarrow A(\boldsymbol{y}),\ \boldsymbol{x} \,\mathsf{is}\, \mathit{rshift}_1(\boldsymbol{y}),\\ B(\boldsymbol{x}) &\leftarrow \mathit{coalesce}\{\boldsymbol{x} \mid A(\boldsymbol{x})\},\\ B(\boldsymbol{x}) &\leftarrow B(\boldsymbol{y}),\ \boldsymbol{x} \,\mathsf{is}\, \mathit{lshift}_{-1}(\boldsymbol{y}). \end{aligned}$$

In this case, $\mathcal{I}^\omega$ comprises $A([0,n])$, for $0 < n$, $B([m,n])$, for $m < 0 < n$, and $B([0,\infty))$; $\mathcal{I}^{\omega\cdot 2}$ also contains $B([m,\infty))$, for $m < 0$, and $\mathcal{I}^{\omega\cdot 2} = \mathcal{I}^{\omega\cdot 2+1}$.

Ontology-Mediated Queries. By a *conjunctive query* (CQ) we mean a first-order formula

$$\boldsymbol{q}(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n) \quad = \quad \exists \boldsymbol{y}_1, \dots, \boldsymbol{y}_k, y_1, \dots y_k\, \Phi,$$

where Φ is a conjunction of atoms with signature predicates whose variables are among the $\boldsymbol{x}_i$, $\boldsymbol{y}_i$, y_i. Using the tuple $\bar{\boldsymbol{x}} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$, we denote this CQ by $\boldsymbol{q}(\bar{\boldsymbol{x}})$ and call $(\Pi, \boldsymbol{q}(\bar{\boldsymbol{x}}))$ an *ontology-mediated query* (OMQ). A *certain answer* to the OMQ $(\Pi, \boldsymbol{q}(\bar{\boldsymbol{x}}))$ over the data instance $\mathcal{D}$ is any tuple $\bar{\boldsymbol{a}} = (\iota_1, \dots, \iota_n)$ such that the ends of the intervals ι_i occur in $\mathcal{D}$ and $\mathcal{I}^{2^{\aleph_0+}}, \mathfrak{s} \models \boldsymbol{q}(\bar{\boldsymbol{x}})$, where $\mathfrak{s}$ maps the variables in $\bar{\boldsymbol{x}}$ to the respective constants in $\bar{\boldsymbol{a}}$.

We leave the investigation of semantical and computational properties of answering *DslD* OMQs to future work, focusing below on representing and querying events in our use cases, where *DslD* programs do not involve recursion.

3 The Turbine Use Case

We illustrate the expressive power of *DslD* and its semantics by representing and querying the event 'active power trip' formulated in Section 1. We assume that the turbine sensors measure various parameters such as active power, blade temperature, etc. asynchronously and that the measurement data is stored in a database as relations $\mathit{ActivePower}([t,t'),v)$ stating that, over the interval $[t,t')$, the measured value of the active power was v.

The event 'active power trip' was considered in [6], where the raw measurement data was first aggregated to avoid short-term fluctuations and then pre-processed by defining (via mappings) temporal concepts 'active power below 0.15' and 'active power above 1.5' using pretty complex mappings. After that, 'active power trip' was captured by means of the following *MTL* rule where, for example, $\boxminus_{[b,e]}\varphi$ (or $\diamondminus_{[b,e]}\varphi$) is true at a moment t if and only if φ is true at every (respectively, some) moment in the interval $[t-b,t-e]$:

$$\begin{aligned}\mathit{ActivePowerTrip} \leftarrow \boxminus_{[0,1m]}\mathit{ActivePowerBelow0.15} \wedge \\ \diamondminus_{[60s,63s]}\boxminus_{[0,10s]}\ \mathit{ActivePowerAbove1.5}.\end{aligned} \quad (2)$$

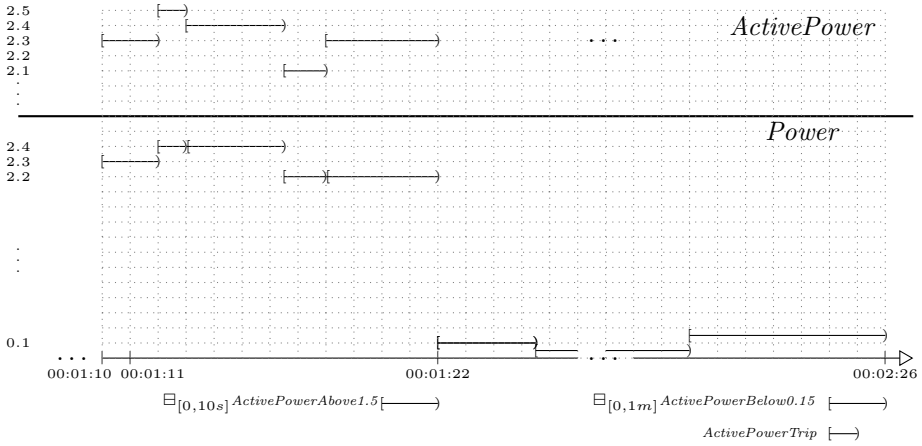
Unlike the one-dimensional metric temporal *MTL*, the language *DslD* is two-dimensional and allows us to aggregate the raw data by taking, as required, the simple moving average of the raw power measurements over the last 5 seconds, capture 'active power below 0.15' and 'active power above 1.5' and define 'active power trip' in a *DslD* program Π_{apt} with two rules:

$$\begin{aligned}\mathit{ActivePowerTrip}(\boldsymbol{w}) \leftarrow\ & \boldsymbol{w} \mathsf{\ is\ } \boldsymbol{z} \fatsemi \boldsymbol{y},\ \ \boldsymbol{z} \mathsf{\ is\ } \mathit{lshift}_{1m}(\boldsymbol{x}),\ \ |\boldsymbol{x}| \geq 1m,\\ & \mathit{coalesce}\{\boldsymbol{x} \mid y < 0.15,\ \ \mathit{Power}(\boldsymbol{x},y)\},\\ & \mathit{coalesce}\{\boldsymbol{y} \mid \boldsymbol{y} \mathsf{\ is\ } \mathit{lshift}_{60s}(\mathit{rshift}_{63s}(\boldsymbol{u})),\\ & \qquad \boldsymbol{u} \mathsf{\ is\ } \mathit{lshift}_{10s}(\boldsymbol{x}),\ \ |\boldsymbol{x}| \geq 10s,\\ & \qquad \mathit{coalesce}\{\boldsymbol{x} \mid y > 1.5,\ \ \mathit{Power}(\boldsymbol{x},y)\}\},\\ \mathit{Power}(\boldsymbol{x},v) \leftarrow\ & \mathit{ActivePower}(\boldsymbol{x},v'),\ \ \boldsymbol{y} \mathsf{\ is\ } \mathit{rext}_{-5s}(\boldsymbol{x}),\\ & \mathit{sma}\{(\boldsymbol{y},v) \mid \mathit{ActivePower}(\boldsymbol{y},v)\},\end{aligned} \quad (3)$$

Thus, the second rule says that the value of *Power* in an interval $\boldsymbol{x}$, for which the data contains a value of *ActivePower*, is defined as the simple moving average of *ActivePower* over the 5s window spanning to the past from the right end of $\boldsymbol{x}$. For example, consider a data instance with the measurements

$$\mathit{ActivePower}([00{:}01{:}10, 00{:}01{:}12), 2.3), \mathit{ActivePower}([00{:}01{:}12, 00{:}01{:}13), 2.5), \ldots$$

shown in the upper part of the picture below



In this case, the interpretation $\mathcal{I}^{2^{\aleph_0+}}$ will contain

$$Power([00{:}01{:}10, 00{:}01{:}12), 2.3), Power([00{:}01{:}12, 00{:}01{:}13), 2.4), \ldots$$

The variable $\boldsymbol{z}$ in the first rule of Π_{apt}, capturing $\boxminus_{[0,10s]}$ *ActivePowerAbove1.5*, will be instantiated by $[00{:}01{:}20, 00{:}01{:}22)$, while $\boldsymbol{y}$, capturing $\diamondminus_{[60s,63s]}$ $\boxminus_{[0,1m]}$ *ActivePowerBelow0.15*, will be instantiated by $[00{:}02{:}24, 00{:}02{:}26)$. As a result, *ActivePowerTrip*($\boldsymbol{w}$) will be true in the interval $\boldsymbol{w} = [00{:}02{:}24, 00{:}02{:}25)$, exactly where *ActivePowerTrip* defined by (2) is true.

4 The Drilling Use Case

Next, we consider several types of drilling process and model them in *DslD*.

Drilling Process. We begin with the process shown in Fig. 1, which can be defined as a sequence of three sub-processes following each other. First, the drill makes contact with the workpiece, and the borehole is created (tapping). In this phase, the diameter increases, which results in increasing process forces. Then, when the diameter, and therefore the process forces stay almost constant (drilling), until the drill breaks through on the other side of the plate, which results in a decrease of process forces (exiting).

Figure 2 shows sensor data from force measurements, sampled at 2kHz drilling process. This signal, however, does not resemble the anticipated shape, which is based on the expectations of domain experts. These expectations are concerned with a particular low-frequency phenomenon. Higher frequency signals within the sampled data are therefore considered noise and need to be filtered accordingly. A simple version of such a filter is moving average [27]. The parameters of this filter can be determined based on domain knowledge, which is also

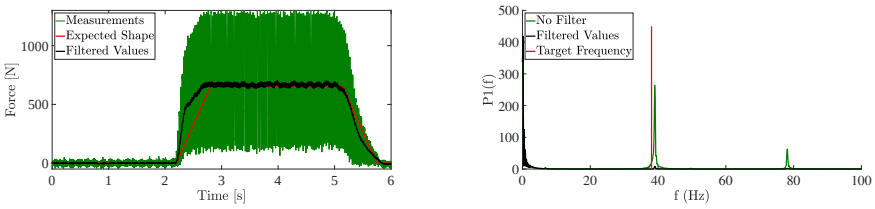


Fig. 2. Filtered data compared with expected shapes for time constants determined based on analytical estimation (left) and respective spectrum before and after aggregation (right).

illustrated in Fig. 2. The drilling process is captured by the following rules:

$$\begin{aligned}
\mathit{AvgForce}(\boldsymbol{x}, y) &\leftarrow \mathit{wavg}\{(\boldsymbol{x}, y) \mid \mathit{Force}(\boldsymbol{x}, y)\}, \\
\mathit{AvgForce}'(\boldsymbol{x}, y) &\leftarrow \mathit{wavg}\{(\boldsymbol{x}, y) \mid \mathit{Force}(\boldsymbol{x}, y)\}, D^{[,)}_{0,13s}(\boldsymbol{x}), && (4)\\
\mathit{ForceDelta}(\boldsymbol{x}, z) &\leftarrow \mathit{AvgForce}'(\boldsymbol{x}, y), \mathit{AvgForce}'(\boldsymbol{x}', y'), \boldsymbol{x}\ \mathsf{A}\ \boldsymbol{x}', z\ \mathsf{is}\ (y - y'), && (5)\\
\mathit{IncForce}(\boldsymbol{x}) &\leftarrow \mathit{coalesce}\{\boldsymbol{x} \mid \mathit{ForceDelta}(\boldsymbol{x}, d), d > 0.1\}, \\
\mathit{ConstForce}(\boldsymbol{x}) &\leftarrow \mathit{coalesce}\{\boldsymbol{x} \mid \mathit{ForceDelta}(\boldsymbol{x}, d), |d| \leq 0.1\}, \\
\mathit{DecForce}(\boldsymbol{x}) &\leftarrow \mathit{coalesce}\{\boldsymbol{x} \mid \mathit{ForceDelta}(\boldsymbol{x}, d), d < -0.1\}, \\
\mathit{SimpleDrilling}(\boldsymbol{x}) &\leftarrow \mathit{IncForce}(\boldsymbol{x}_1), \mathit{max}\{(\boldsymbol{x}_1, y_1) \mid \mathit{AvgForce}(\boldsymbol{x}_1, y_1)\},\ \boldsymbol{x}_1\ \mathsf{A}\ \boldsymbol{x}_2, \\
&\quad \mathit{ConstForce}(\boldsymbol{x}_2), \mathit{max}\{(\boldsymbol{x}_2, y_2) \mid \mathit{AvgForce}(\boldsymbol{x}_2, y_2)\},\ \boldsymbol{x}_2\ \mathsf{A}\ \boldsymbol{x}_3, \\
&\quad \mathit{DecForce}(\boldsymbol{x}_3), \mathit{max}\{(\boldsymbol{x}_3, y_3) \mid \mathit{AvgForce}(\boldsymbol{x}_3, y_3)\}, \\
&\quad |y_1 - y_2| < y_2 \times 0.05,\ |y_2 - y_3| < y_3 \times 0.05, \\
&\quad \boldsymbol{x}\ \mathsf{is}\ (\boldsymbol{x}_1 \uplus \boldsymbol{x}_3 \uplus \boldsymbol{x}_3). && (6)
\end{aligned}$$

Smooth Drilling. Drilling processes can be further classified based on other characteristics within the force measurements. Throughout such a drilling process, however, anomalies can occur, the simplest of which is the presence of significant peaks that might come, for example, from material inhomogeneities. In this sense, a *smooth drilling process* can be distinguished by evaluation of the maximum and minimum values within the simple drilling event:

$$\begin{aligned}
\mathit{Drilling}(\boldsymbol{x}) &\leftarrow \mathit{ConstForce}(\boldsymbol{x}), \\
\mathit{SmoothDrilling}(\boldsymbol{x}) &\leftarrow \mathit{SimpleDrilling}(\boldsymbol{x}),\ \mathit{Drilling}(\boldsymbol{x}_1),\ \boldsymbol{x}_1 \subseteq \boldsymbol{x},\ \mathit{AvgForce}(\boldsymbol{x}_1, y), \\
&\quad \mathit{max}\{(\boldsymbol{x}_1, y_1) \mid \mathit{AvgForce}(\boldsymbol{x}_1, y_1)\},\ y_1 - y \leq 0.1 \times y, \\
&\quad \mathit{min}\{(\boldsymbol{x}_1, y_2) \mid \mathit{AvgForce}(\boldsymbol{x}_1, y_2)\},\ y - y_2 \leq 0.1 \times y.
\end{aligned}$$

Unstable Drilling. The event *unstable drilling process* can be classified based on the standard deviation of the force readings. Even though they might look similar based on their average process forces in each phase, unstable processes will have significantly higher standard deviations than stable ones. Again, the

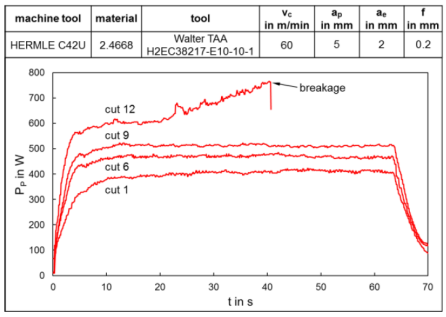


Fig. 3. Electrical power readings for a sequence of similar cuts until tool break occurred [11].

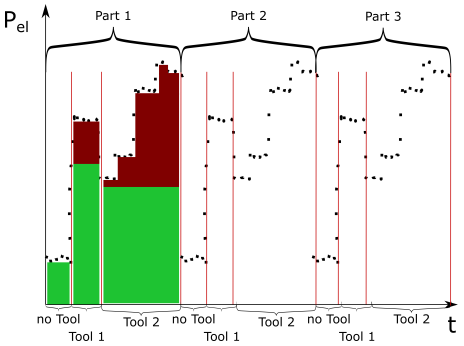


Fig. 4. Total electrical power input (P_{el}) for a production machine while manufacturing three similar parts.

threshold that would be used here has to be defined based on domain knowledge:

$$\begin{aligned} \mathit{SmoothDrilling}(\boldsymbol{x}) \leftarrow & \mathit{SimpleDrilling}(\boldsymbol{x}),\ \mathit{Drilling}(\boldsymbol{x}_1),\ \boldsymbol{x}_1 \subseteq \boldsymbol{x},\ \mathit{AvgForce}(\boldsymbol{x}_1, y_1), \\ & \mathit{sdev}\{(\boldsymbol{x}_1, y_2) \mid \mathit{Force}(\boldsymbol{x}_1, y_2)\},\ |y_2 - y_1| \leq 0.1 \times y_1, \end{aligned}$$

where *sdev* is an aggregation functional computing the standard deviation.

Tool Break. Tool breaks happen regularly in industrial production processes. Example power readings for such an event are depicted in Fig. 3. In this example, power is directly related to apparent process forces. As a consequence, force readings would differ from the displayed power readings only by a constant conversion factor. To identify tool breaks, we simply find any patterns that match the concept *simple drilling*, but are shorter then some expected duration:

$$\mathit{ToolBreak}(\boldsymbol{x}) \leftarrow \mathit{SimpleDrilling}(\boldsymbol{x}),\ \mathit{ExpectedDrillingTime}(\boldsymbol{x}, y),\ |\boldsymbol{x}| \leq y,$$

where $\mathit{ExpectedDrillingTime}(\boldsymbol{x}, y)$ specifies the expected time of drilling for the given drill model, material thickness, etc. and is defined by means of mappings.

Energy Per Tool. Closely related to tool wear is the remaining tool lifetime. One way to determine this measure from sensor data is the amount of accumulated electrical energy that was used to drive a given tool. To illustrate, consider Fig. 4 showing power measurements for a production machine. Within the observed window, three similar parts (having the same geometric features) are manufactured. In order to do so, two different tools are used. To calculate the energy per tool, the power measurements need to be integrated over the intervals in which the respective tool was active. Before doing so, however, it is necessary to deduct the base load of the machine. A machine base load is defined as its (electrical) power demand without material removal due to auxiliary systems or

power loss in bearings. We use the following *DslD* rules:

$$\begin{aligned}
\mathit{Pbase}(\boldsymbol{x}, v) \leftarrow & \mathit{SimpleDrilling}(\boldsymbol{y}), \boldsymbol{x} \,\mathsf{is}\, \mathit{lext}_{-10s}(\boldsymbol{y}), \mathit{wavg}\{(\boldsymbol{x}, v) \mid \mathit{Power}(\boldsymbol{x}, v)\},\\
\mathit{Ptool}(\boldsymbol{x}, u) \leftarrow & \mathit{wavg}\{(\boldsymbol{x}, v) \mid \mathit{Power}(\boldsymbol{x}, v)\}, \mathit{Pbase}(\boldsymbol{x}', v'), \boldsymbol{x} \subseteq \boldsymbol{x}', u \,\mathsf{is}\, v - v',\\
\mathit{Etool}(\boldsymbol{x}, v) \leftarrow & \mathit{int}\{(\boldsymbol{x}, v) \mid \mathit{Ptool}(\boldsymbol{x}, v)\}, \mathit{SimpleDrilling}(\boldsymbol{x}),
\end{aligned}$$

where, for a functional relation R, aggregation functional $\mathit{int}(R)$ is defined as

$$\mathit{int}(R) = \big\{\big(\iota, \int_\iota f_R(x)\,dx\big) \mid \iota \in \mathsf{int}_{\mathbb{R}} \text{ is bounded and } \iota \subseteq \mathit{dom}\, R\big\}.$$

5 Evaluating Non-Recursive *DslD* by SQL

In this section, we show that query answering in non-recursive *DslD* can be reduced to evaluation of SQL queries. The algorithm is an extension of the classical algorithm for evaluating non-recursive datalog programs (see, e.g., [31]). In a nutshell, the algorithm introduces views for all head predicates, and computes these views in a bottom-up fashion following the dependency relation. For each unary predicate A, we create a view, called `A`, with columns `begin` and `end`; for each binary predicate R, a view `R` with columns: `begin`, `end` and `value`. In the translation, Allen's relations and the metric constructs can be implemented in a straightforward way. The major challenge is to deal with aggregation atoms.

We illustrate our translation with examples. To start, we show how to deal with the simple drilling use case. First, we compute the predicate $\mathit{AvgForce}'$ in rule (4), which depends on view `Force(begin, end, value)`. We need to deal with the discretisation predicate $D^{[,)}_{0,13s}$, which 'splits' the rows of the `Force` view into windows of 13s. For convenience, we assume that we have an auxiliary table `nums(id)` storing enough numbers 0, 1, ... The following view `D_force` discretises `Force`. In the resulting table, each row contains a split interval [`begin`, `end`), which is contained in the window [`w_begin`, `w_end`):

```
CREATE  VIEW D_force AS (
   SELECT GREATEST(begin, nums.id * 13) AS begin,
          LEAST((nums.id + 1) * 13, end) AS end,
          Force.value AS value,
          nums.id * 13 AS w_begin, (nums.id + 1)* 13 AS w_end
   FROM Force, nums
   WHERE begin / 13 <= nums.id  AND nums.id <= end / 13);
```

The usage of the auxiliary table `nums` is not always necessary, since it can be generated on the fly with a function like `generate_series`. Alternatively, one can also simulate this join with user-defined functions.

Then we are ready to define the view `AvgForceP` by grouping according to their windows and computing the weighted average by means of the built-in aggregate function `SUM`:

```
CREATE VIEW AvgForceP AS (
   SELECT MIN(begin) AS begin, MAX(end) AS end,
```

```
        SUM(value * (end - begin)) / SUM(end - begin) AS value
    FROM D_force
    GROUP BY w_begin, w_end);
```

Note that, in general, many aggregation functionals (e.g., *coalesce* in rule (6)) cannot be directly defined with built-in aggregate functions in SQL, and one needs to introduce user-defined aggregate functions (UDAFs) according to the SQL engine, e.g., Apache Spark[5] and MS SQL Server[6]. With UDAFs, the coalesce operator can be implemented using, e.g., the standard algorithm of [33].

Next, for the predicate `ForceDelta`, we introduce the view following rule (5):

```
CREATE VIEW ForceDelta AS (
    SELECT a1.begin, a1.end, a2.value - a1.value AS value
    FROM AvgForceP a1, AvgForceP a2
    WHERE a1.end = a2.begin);
```

Other predicates in this simple drilling use case can be computed in a similar fashion. Eventually, the view `SimpleDrilling` provides us the answer.

Finally, we consider the turbine use case. The major technical difference is the handling of windows. For rule (3), which does not have a discretisation schema in the body, we can define time-based windows for each row relying on the range-based windows in the standard SQL query language. In this example, assuming the data is provided as `tb_power(ts, value)` table, the time-based window can be defined as follows:

```
CREATE VIEW SmaPower AS (
    SELECT LAG(ts,1) OVER (ORDER BY ts) AS begin,
           ts AS end, AVG(value) OVER  W AS avg
    FROM tb_power
    WINDOW W AS
     (ORDER BY ts RANGE BETWEEN 5 PRECEDING AND 0 FOLLOWING));
```

6 Evaluation

We evaluated the SQL translations of the rules defined in our two use cases over large amounts of data. For the turbine use case, we have 4 days of power data in the form of timestamp, value pairs for one running turbine. We replicated this sample to imitate the data for one turbine over 10 different periods ranging from 32 to 320 months (from 0.23 GB to 2.3 GB). For the drilling use case, we collected 2.6 GB of real data from a manufacturing company. This data contains force and power measurements associated with timestamps from one drilling tool. We ran our experiments on an AWS server with an Intel Xeon Platinum-8175 processor having 8 logical cores at 2.5 GHz and 64 GB of RAM. The SQL queries were executed on Apache Spark 2.4.0.

[5] https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/expressions/UserDefinedAggregateFunction.html

[6] https://docs.microsoft.com/en-us/sql/relational-databases/clr-integration-database-objects-user-defined-functions/clr-user-defined-aggregates

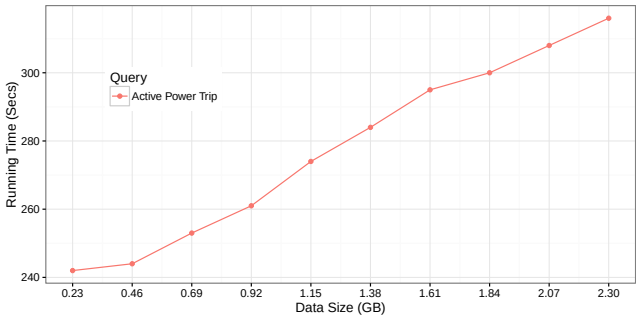


Fig. 5. Running times of the active power trip query.

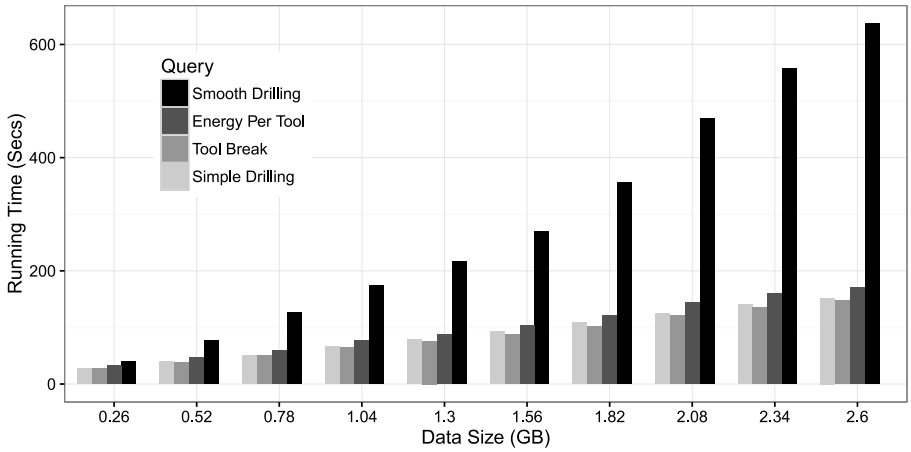


Fig. 6. Running times of the drilling queries.

Figure 5 illustrates the execution time for the SQL translation of the 'active power trip' rule in the turbine use case (Section 3), while Figure 6 shows the times for the drilling use case (Section 4). Both results from the two use cases show that the execution times scale linearly over monotonically increasing data. As expected, in the cases where we can benefit from discretisation schema (e.g., drilling use case queries) the computation load reduce significantly. On the other hand, the 'active power trip' query requires row by row windowing, and this leads to additional work load compared to the cases involving discretisation schema.

7 Conclusion

As shown by the use-cases considered in this paper, engineers analysing the behaviour of industrial systems by detecting events among sensor log data are facing a challenging task of representing those events in terms of the existing

query languages. The OBDA paradigm would drastically simplify the engineers' work if the events in question could be captured in ontologies rather than queries. Based on the use-cases, we proposed a suitable OBDA ontology language *DslD*, featuring aggregate functionals, Allen's interval relations and various metric constructs. We showed that the non-recursive fragment of *DslD* is enough to capture the events in our cases and can be translated into sufficiently efficient SQL queries, which we tested on real-world data. We achieved good performance results with large amounts of data.

Encouraged by the satisfaction of the involved engineers, we are planning to do a proper user study to see how well the engineers can use this language. Further, an investigation of the theoretical properties of the proposed language and optimisation techniques will be conducted.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)
2. Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., Zakharyaschev, M.: First-order rewritability of temporal ontology-mediated queries. In: Proc. of IJCAI. pp. 2706–2712. IJCAI/AAAI (2015)
3. Baader, F., Borgwardt, S., Lippmann, M.: Temporalizing ontology-based data access. In: Proc. of CADE-24. LNCS, vol. 7898, pp. 330–344. Springer (2013)
4. Baader, F., Sattler, U.: Description logics with aggregates and concrete domains. Inf. Syst. **28**(8), 979–1004 (2003)
5. Borgwardt, S., Lippmann, M., Thost, V.: Temporal query answering in description logic DL-Lite. In: Proc. of FroCoS. LNCS, vol. 8152, pp. 165–180. Springer (2013)
6. Brandt, S., Kalayci, E.G., Ryzhikov, V., Xiao, G., Zakharyaschev, M.: Querying log data with metric temporal logic. J. Artif. Intell. Res. **62**, 829–877 (2018)
7. Bresolin, D., Kurucz, A., Muñoz-Velasco, E., Ryzhikov, V., Sciavicco, G., Zakharyaschev, M.: Horn fragments of the Halpern-Shoham interval temporal logic. ACM Trans. Comput. Log. **18**(3), 22:1–22:39 (2017)
8. Calvanese, D., Kharlamov, E., Nutt, W., Thorne, C.: Aggregate queries over ontologies. In: Proc. of ONISW. pp. 97–104 (2008)
9. Gutiérrez-Basulto, V., Jung, J., Kontchakov, R.: Temporalized EL ontologies for accessing temporal data: Complexity of atomic queries. In: Proc. of IJCAI. (2016)
10. Gutiérrez-Basulto, V., Klarman, S.: Towards a unifying approach to representing and querying temporal data in description logics. In: Proc. of RR. LNCS, vol. 7497, pp. 90–105. Springer (2012)
11. Hacksteiner, M., Duer, F., Finkeldei, D., Obermair, M., Bleicher, F.: Monitoring of process power and energy during machining. In: Int. Conf. on High Speed Machining (2016)
12. Halpern, J., Shoham, Y.: A propositional modal logic of time intervals. J. ACM **38**(4), 935–962 (1991)
13. Hella, L., Libkin, L., Nurmonen, J., Wong, L.: Logics with aggregate operators. In: LICS, pp. 35–44 (1999)
14. Hussein, R., Sadek, &.A., Elbestawi, M.A., Attia, M.H.: Low-frequency vibration-assisted drilling of hybrid CFRP/Ti6Al4V stacked material. The International Journal of Advanced Manufacturing Technology (98), 2801–2817 (2018)

15. Kalaycı, E.G., Brandt, S., Calvanese, D., Ryzhikov, V., Xiao, G., Zakharyaschev, M.: Ontology-based access to temporal data with ontop: a framework proposal. Int. Journal of Applied Mathematics and Computer Science **29**(1), 17–30 (2019)
16. Kalayci, E.G., Xiao, G., Ryzhikov, V., Kalayci, T.E., Calvanese, D.: Ontop-temporal: A tool for ontology-based query answering over temporal data. In: CIKM. pp. 1927–1930. ACM (2018)
17. Kharlamov, E., Mailis, T., Mehdi, G., Neuenstadt, C., Özçep, Ö.L., Roshchin, M., Solomakhina, N., Soylu, A., Svingos, C., Brandt, S., Giese, M., Ioannidis, Y.E., Lamparter, S., Möller, R., Kotidis, Y., Waaler, A.: Semantic access to streaming and static data at Siemens. J. Web Semant. **44**, 54–74 (2017)
18. Kharlamov, E., Mehdi, G., Savkovic, O., Xiao, G., Kalayci, E.G., Roshchin, M.: Semantically-enhanced rule-based diagnostics for industrial internet of things: The SDRL language and case study for Siemens trains and turbines. JWS. (2018)
19. Klarman, S., Meyer, T.: Querying temporal databases via OWL 2 QL. In: Proc. of RR. LNCS, vol. 8741, pp. 92–107. Springer (2014)
20. Kontchakov, R., Pandolfo, L., Pulina, L., Ryzhikov, V., Zakharyaschev, M.: Temporal and spatial OBDA with many-dimensional Halpern-Shoham logic. In: IJCAI. pp. 1160–1166. IJCAI/AAAI Press (2016)
21. Kostylev, E.V., Reutter, J.L.: Complexity of answering counting aggregate queries over DL-Lite. J. Web Semantics **33**, 94–111 (2015)
22. Kusiak, A.: Smart Manufacturing. International Journal of Production Research **56**(1-2), 508–517 (jan 2017)
23. Lenzerini, M.: Managing data through the lens of an ontology. AI Magazine **39**(2), 65–74 (2018)
24. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: FORMATS and FTRTFT. LNCS, vol. 3253, pp. 152–166. Springer (2004)
25. Özçep, Ö.L., Möller, R.: Ontology based data access on temporal and streaming data. In: RW. LNCS, vol. 8714, pp. 279–312. Springer (2014).
26. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics **X**, 133–173 (2008)
27. Runkler, T.A.: Datenvorverarbeitung. In: Data Mining, pp. 21–34. Vieweg+Teubner, Wiesbaden (2010)
28. Shkapsky, A., Yang, M., Zaniolo, C.: Optimizing recursive queries with monotonic aggregates in deals. In: ICDE. pp. 867–878. IEEE Computer Society (2015)
29. Soylu, A., Kharlamov, E., Zheleznyakov, D., Jiménez-Ruiz, E., Giese, M., Skjæveland, M., Hovland, D., Schlatte, R., Brandt, S., Lie, H., Horrocks, I.: OptiqueVQS: A visual query system over ontologies for industry. Sem. Web **9**(5), 627–660 (2018).
30. Thoben, K.D., Wiesner, S.A., Wuest, T.: "Industrie 4.0" and Smart Manufacturing-A Review of Research Issues and Application Examples. International Journal of Automation Technology **11**, 4–16 (jan 2017).
31. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, Volume I. Computer Science Press (1988)
32. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. In: IJCAI. (2018)
33. Zhou, X., Wang, F., Zaniolo, C.: Efficient temporal coalescing query support in relational database systems. In: Proc. of DEXA. Springer (2006)