Semantic Rules for Machine Diagnostics: Execution and Management

Evgeny Kharlamov
University of Oxford
evgeny.kharlamov@cs.ox.ac.uk

Ognjen Savković
Free University of Bozen-Bolzano
Ognjen.Savkovic@unibz.it

Guohui Xiao
Free University of Bozen-Bolzano
xiao@inf.unibz.it

Rafael Penaloza
Free University of Bozen-Bolzano
Rafael.Penaloza@unibz.it

Gulnar Mehdi
Siemens Corporate Technology
Technical University of Munich
gulnar.mehdi@siemens.com

Mikhail Roshchin
Siemens Corporate Technology
mikhail.roshchin@siemens.com

Ian Horrocks
University of Oxford
ian.horrocks@cs.ox.ac.uk

ABSTRACT

Rule-based diagnostics of equipment is an important task in industry. In this paper we present how semantic technologies can enhance diagnostics. In particular, we present our semantic rule language sigRL that is inspired by the real diagnostic languages used in Siemens. SigRL allows to write compact yet powerful diagnostic programs by relying on a high level data independent vocabulary, diagnostic ontologies, and queries over these ontologies. We study computational complexity of SigRL: execution of diagnostic programs, provenance computation, as well as automatic verification of redundancy and inconsistency in diagnostic programs.

CCS CONCEPTS

• **Information systems → Enterprise information systems**;

KEYWORDS

Diagnostic Systems, Rules, Sensor Signals, Ontologies, Complexity

1 INTRODUCTION

Motivation. Diagnostic systems play an important role in industry since they help to maximise equipment's up-time and minimise its maintenance and operating costs [19]. In the energy sector companies like Siemens often rely on *rule-based* diagnostics to analyse power generating equipment by, e.g., testing newly deployed electricity generating gas turbines [15], or checking vibration instrumentation [17], performance degradation [18], and faults in operating turbines. For this purpose diagnostic engineers create and use complex diagnostic rule-sets to detect equipment abnormalities.

An important class of rules that are commonly used in Siemens are *signal processing rules* (SPRs) that allow to (1) filter, aggregate, combine, and compare *signals*[1] coming from sensors installed in equipment and (2) fire notification messages when a certain pattern in signals is detected. *Authoring* and *maintaining* SPR based rule-sets is a challenging problem. We now give an example of SPRs, discuss the challenges in mode details, and present our solutions.

Example 1.1. Consider the purging diagnostic task: *Verify that the purging*[2] *is over in the main flame component of the turbine T1.* Intuitively this task requires to check in the turbine T1 that: *(i)* the main flame was on for at least 10s and then stopped, *and (ii)* the purging of rotors in the starting-component of T1 started, *(iii)* 20s after this, the purging stopped. The fact that the purging of a rotor started or ended can be detected by analysing its speed, that it, by comparing the average speed of its speed sensors with purging thresholds that are specific for individual rotors. Step *(ii)* can be written using Siemens SPRs as follows:

```
$PurgStartRotor1 : truth(avg('S21R1T1', 'S22R1T1', 'S23R1T1'), > 1.300).
```

Here, the first rotor in the starting-component of T1 has three speed sensors 'S21R1T1','S22R1T1', and 'S23R1T1' and the rotation speed of 1.300 indicates that the purging started. □

Challenges with Authoring SPRs. The main challenge for authoring is that SPRs in most of existing diagnostic systems including the ones used in Siemens are highly *data dependent* in the sense that specific characteristic of individual sensors and pieces of equipment are explicitly encoded in SPRs. As the result for a typical turbine diagnostic task engineers have to write from dozens to hundreds of SPRs that involve hundreds of sensor ids, component codes, sensor and threshold values as well as equipment configuration and design data. For example, a typical Siemens gas turbine has about 2,000 sensors and the purging diagnostic task for it requires around 300 SPRs, most of which are similar in structure but different equipment specific data values. Thus, there is a need in industry, and in particular in Siemens for a higher level diagnostic rule language that allows to express *what* the diagnostic task should do rather than *how* it should do it for specific equipment. Such language should be high level, data independent, while powerful enough to express in a concise way most of typical diagnostic tasks in Siemens.

CIKM'17, November 6–10, 2017, Singapore.
© 2017 ACM. ISBN 978-1-4503-4918-5/17/11…$15.00
DOI: http://dx.doi.org/10.1145/3132847.3133159

[1]Signals are are time stamped sequences of measurement values.
[2]Purging is the process of flushing out liquid fuel nozzles or other parts which may contain undesirable residues.

Challenges with Management of SPRs. Development of a diagnostic rule-set in Siemens is typically a collaborative and open-ended process by a group of diagnostic engineers. Thus, the engineers may introduce rules that either *repeat* what other rules already express or *contradict* them, i.e., by stating that purging is 'over' while the other rules say that is it 'in progress'. The former problem of *redundancy* in diagnostic rule-sets affects the performance of diagnostics and the latter of *inconsistency* among rules makes diagnostic results counter-intuitive and unreliable. Moreover, the larger the rule-set gets, the harder it becomes to trace the *provenance* of the messages it fires which again affects the reliability of diagnostic results. Thus, there is a need in Siemens for semi-automatic rule management support that includes detection of redundancy and inconsistency in rule sets, as well as computation of provenance for diagnostic results.

Our Solution. We rely on *semantic technologies* to address the the above mentioned challenges. In particular we rely on *ontologies* [2] to define a novel SPR language and on *reasoning* [5] over ontologies to foster execution and maintenance of diagnostic tasks. In short, an ontology is a formal conceptualisation of the domain of interest that consists of a *vocabulary*, i.e., names of classes, attributes and binary relations, and *axioms* over the terms from the vocabulary that, e.g., assign attributes of classes, define relationships between classes, compose classes, class hierarchies, etc. Since ontologies are specified using a formal logical language such W3C standardised ontology web language OWL 2, one can query ontologies and check their properties using reasoning that typically corresponds to logical entailment and implemented in many efficient state-of-the-art reasoning systems. We refer the reader to [2] for more details on ontologies and reasoning and to [9, 10, 12–14] to our previous studies of the semantic diagnostic problem.

In order to address the authoring challenge we propose:

- an SPR language *sigRL* that treats signals as first class citizens and allows to process signals (filter, aggregate, combine, and compare signals) in a high level, declarative, and data independent fashion;
- semantic diagnostic programs that combine *sigRL* rules with diagnostic background knowledge captured using ontologies and allow to express complex diagnostic tasks in an abstract fashion by exploiting both ontological vocabulary and queries over ontologies to identify relevant information (such as sensor ids and threshold values) about the equipment that should undergo the diagnostics.

In order to address the management challenge, we

- proposed how to execute diagnostic programs, verify redundancy and inconsistency in diagnostic programs, and to compute provenance that explains the reasons for diagnostic results.

Note that we designed *sigRL* in such a way that, on the one hand, it captures the main signal processing features required by Siemens turbine diagnostic engineers and, on the other hand allows for efficient execution and management of diagnostic programs. Moreover, we believe that our approach is generic enough to find its pathways in other diagnostic applications, e.g., for other type of complex equipment of networking [11].

Related Work. Recent efforts have been made to extend semantical languages with temporal concepts, e.g., [1, 3]. More prominently, DatalogMTL which allows operations on time intervals and favorable computational properties, was presented in [4]. While such language are very powerful in terms of modeling time-dependent rules, the authors do not consider yet the integration of such rules in analytical functions that are crucial for diagnostic analysis. Still results from these work can help us in understanding the modeling of the temporal dimension. Finally, there is a body of work on rule-driven diagnostics of distributed systems, e.g. [6, 11]. Relation of our work to this approaches requires further investigation.

2 *SIGRL* DIAGNOSTIC LANGUAGE

In order to capture diagnostic tasks we propose the *sigRL* language that allows to specify diagnostic programs and message rules, and combine them with ontological axioms.

Diagnostic Programs. In our language *sigRL*, a *(diagnostic) programs* $\Pi = (\mathcal{D}, \Sigma)$ consisting of two layers: *data layer* $\mathcal{D}$ and *rule layer* Σ. Intuitively, the data layer keeps all the data of interest: signals and details of concrete equipment; the rule layer manipulates the data layer using rules and ontological axioms that are generic for turbines.

The data layer $\mathcal{D} = (\mathcal{S}, \mathcal{A})$ consists of two components: *basic signals* $\mathcal{S}$ which represent signal data and *class assertions* $\mathcal{A}$ which define, for instance, types of sensors and asserts types of turbine components, e.g., as follows:

$$\mathsf{ClassAssertion(RotorSensor\ `S21R1T1')}.$$

In our setting, a *signal* s is a first-class citizen and defined as a pair (o_s, f_s) where o_s is *sensor id* and *signal function* f_s defined on $\mathbb{R}$ to $\mathbb{R} \cup \{\bot\}$, where $\bot$ denotes the absence of a value. We assume that we are given a finite set $\mathcal{S} = \{s_1, \ldots, s_n\}$ of *basic* signals that are readings obtained from a single sensor (e.g., in a turbine) for different time points.

The rule layer $\Sigma = (C, O)$ contains signal processing expressions C and ontological axioms O. Signal expressions filter and manipulate basic signals and create new more complex signals. In our language we group signals in *basic (ontological) concepts* and signal expression are defined on the level of concepts. Formally, a *signal processing expression* is recursively defined as follows:

$$\begin{array}{lllll} C & \leftarrow & \alpha \circ C_1 & \mid \mathit{agg}\ C_1 & \mid \\ & & C_1 : \mathsf{filterValue}(\odot, \alpha) & \mid C_1 : \mathsf{filterTime}(\odot, \alpha) & \mid \\ & & C_1 : \mathit{filterAlign}\ C_2 & \mid C_1 : \mathsf{trending}(\mathit{direction}). & \end{array}$$

where C, C_1, C_2 are concepts, $\circ \in \{+, -, \times, /\}$, $\alpha \in \mathbb{R}$, $\mathit{agg} \in \{\mathsf{min}, \mathsf{max}, \mathsf{avg}, \mathsf{sum}\}$, $\odot \in \{<, >, \leq, \geq\}$, $\mathit{filterAlign} \in \{\mathsf{within}, \mathsf{after}[t], \mathsf{before}[t]\}$ where t is a period and *direction* is either $\{\mathsf{up},\ \mathsf{down}\}$.

The semantics of expressions is as follows. If $C = \alpha \circ C_1$ then C contains one signal s' for each signal s in C_1 with function defined with $f_{s'} = \alpha \circ f_s$, or if $C_1 : \mathsf{filterValue}(\odot, \alpha)$ then C contains one signal s' for each signal s in C_1 with $f_{s'}(t) = \alpha \odot f_s(t)$ if $f_s(t) \odot \alpha$ at time point t; otherwise $f_{s'}(t) = \bot$. Due to the lack of space we illustrate other constructs of expressions with an example.

Example 2.1. The following expression defines the start of a purging process from the running example:

$$\mathsf{PurgingStart} = \mathit{avg}\ \mathsf{mainSpeedSensor} : \mathsf{filterValue}(>, \mathit{purgingSpeed}) \quad (1)$$

Intuitively, this rule computes a new signal from a given one s in a bottom-up fashion. First, it computes *avg* mainSpeedSensor by creating new auxiliary signal s'' with signal function $f_{s''}(t) =$

$avg_{s \in C_1} f_s(t)$, i.e., the signal takes an average at each time point t. Then, using filter filterValue($>$, *purgingSpeed*) it creates s' from s'' by taking only values greater than *purgingSpeed*. Formally, $f_{s'}(t) = f_{s''}(t)$ if $f_{s''}(t) > purgingSpeed$; otherwise $f_{s'}(t) = \bot$. □

Ontological axioms O describe general characteristics of power generating equipment which includes partonomy of its components, characteristics and locations of its sensors, etc. As an example consider the following ontological axioms that says that RotorSensor is a kind of SpeedSensor:

$$\text{SubClassOf(RotorSensor, MainSpeedSensor)}. \tag{2}$$

In order to guarantee efficiency of diagnostics with *sigRL*, we consider a tractable ontology language OWL 2 QL [5] that is standardised by W3C and allows to express subclass (resp. sub-property) relationship between classes and projections of properties (resp. between properties).

Message Rules. On top of diagnostic programs Π *sigRL* allows to define *message rules* that report the current status of a system. Formally, they are defined as Boolean combinations of signal processing expressions:

$$msg(m) \leftarrow D, \quad where \quad D := \; C \;\mid\; not\, D_1 \;\mid\; D_1 \; and \, D_2.$$

Example 2.2. In order to complete our running example in *sigRL* we now define PurgingStop signal processing expression in (3) and then a message rule in (1):

$$\text{PurgingStop} = avg\,\text{mainSpeedSensor} : \text{filterValue}(<, nonPurgSpeed) \tag{3}$$

$$\text{msg}(\textit{“Purging over”}) = \text{FlameSensor} : \text{filterTime}(>, 10s) \; and \; \text{PurgingStart} : after[20s]\ \text{PurgingStop}. \tag{4}$$

Semantics of *sigRL*. We extend first-order interpretations that are used to define semantics of OWL 2 QL axioms. In *sigRL*, an *interpretation* $\mathcal{I}$ is a pair $(\mathcal{I}_{FOL}, \mathcal{I}_{\mathcal{S}})$ where $\mathcal{I}_{FOL}$ interprets objects and their relationships (like in OWL 2 QL) and $\mathcal{I}_{\mathcal{S}}$ interprets signals. For brevity, we immediately define when an interpretation $\mathcal{I}$ is a *model* of a program $\Pi = ((\mathcal{S}, \mathcal{A}), (C, O))$. First, we define how $\mathcal{I}$ interprets data layer. For the given set of signals $\mathcal{S}$: $\mathcal{S}^{\mathcal{I}} = \{s_1^{\mathcal{I}}, \dots, s_n^{\mathcal{I}}\}$ such that $\mathcal{I}_{FOL}$ 'returns' the signal id, $s^{\mathcal{I}_{FOL}} = o_s$ and $\mathcal{I}_{\mathcal{S}}$ 'returns' the signal itself, $s^{\mathcal{I}_{\mathcal{S}}} = s$. For each object o in $\mathcal{A}$ we define $o^{\mathcal{I}_{\mathcal{S}}}$ as s if o is the id of s from $\mathcal{S}$; otherwise $(o, f_\bot)$ where $f_\bot(t) = \{\bot\}$, for all $t \in \mathbb{R}$. Then we define when $\mathcal{I}$ "models" the rule layer. In particular, $O^{\mathcal{I}}$ extends the notion of first-order logics interpretation as follows: $O^{\mathcal{I}_{FOL}}$ is a first-order logics interpretation O consistent with $\mathcal{A}^{\mathcal{I}_{FOL}}$, i.e., $\mathcal{I}_{FOL} \models (O, \mathcal{A})$; and $O^{\mathcal{I}_{\mathcal{S}}}$ is defined for objects, concepts, roles and attributes following $\mathcal{S}^{\mathcal{I}}$ and $\mathcal{A}^{\mathcal{I}}$. For example, for a concept A we define $A^{\mathcal{I}_{\mathcal{S}}} = \{s^{\mathcal{I}_{\mathcal{S}}} \mid o_s^{\mathcal{I}_{FOL}} \in A^{\mathcal{I}_{FOL}}\}$. Finally, $\mathcal{I}$ models C if it *satisfies* all signal expressions; and this can be defined recursively in a button-up fashion following the definition dependencies between them. For example, if $C = \text{agg } C_1$ then $C^{\mathcal{I}_{\mathcal{S}}} = \{(o_{\text{agg},C_1}, f_{\text{agg},C_1})\}$ such that $f_{\text{agg},C_1}(t) = \text{agg}\{f_s(t) \mid s \in C_1^{\mathcal{I}_{\mathcal{S}}}\}$. And similarly, one can define satisfiability for the other expressions.

3 EXECUTION AND MAINTENANCE OF DIAGNOSTIC PROGRAMS

In this section, we discuss the computational tasks of interests. Assume we are given a program $\Pi = (\mathcal{D}, \Sigma)$. For brevity, for a message head $msg(m)$ we simply use m (the text of the message).

Firing a message. Let m be a message of $msg(m) \leftarrow D$. A principal reasoning task is to determine whether Π *fires* a message m. We say that Π fires m, written $\Pi \models m$, if for each model $\mathcal{I}$ of Π it holds that $D^{\mathcal{I}} = true$. Here, $D^{\mathcal{I}}$ evaluates as a Boolean expression starting from the concepts: if $D = C$ then $D^{\mathcal{I}} = true$ iff $C^{\mathcal{I}} \neq \emptyset$; and $\mathcal{I}$ extends over logical operators *and* and *not* naturally.

Our programs enjoy the *canonical* model property, i.e., each program has a unique (Herbrand) interpretation [2] which is minimal and can be constructed starting from a data layer and extending over the rule layer. We now illustrate this on the running example.

Example 3.1. Consider our running program Π composed of rules and axioms from the examples in the paper. Let $\mathcal{I}_{can}$ be its canonical interpretation. *(i)* First, we form the dependency graph among concepts and evaluate one by one starting from leaves. In our case, we start by evaluating basic concept rotorSensor that collects all rotor sensor ids. *(ii)* Then, we evaluate other ontological concepts. In particular, we evaluate (2), and populate mainSpeedSensor with sensors from rotorSensor. *(iii)* Once we finish with ontological evaluation, we start evaluation signal expression. In this case, we evaluate the expression from Equation (1), again in a bottom-up fashion. Imagine that $\text{rotorStart}^{\mathcal{I}_{can}}$ contains sensor ids: 'S21R1T1', 'S22R1T1' and 'S23R1T1'. At the same time, those sensors have signal functions assigned from $\mathcal{S}^{\mathcal{I}_{can}}$. Let us call them f_1, f_2 and f_3. Expression avg rotorStart computes a new signal, say s_4, by taking average of f_1, f_2 and f_3 at each time point. After this, filterValue($>$, *purgingSpeed*) eliminates all values of s_4 that are not $>$ *purgingSpeed*. Similarly, we compute signal transformations for the expression from Equation (3). *(iv)* Finally, we use those two expressions to evaluate the Boolean expression of Message (4). The message is fired if: there exists at least one signal in FlameSensor being active for >10 sec, and (3) starts within 20 sec after (1). □

Thus, one can check "$D^{\mathcal{I}}$" only on the canonical model. This implies that one can interpret *sigRL* programs and messages in a bottom-up fashion.

Redundancy of messages. One of the critical problems of the rules maintenance is redundancy. To analyse this problem, the simplest test is to understand whether one message is always fired when another message is fired. In order to make sure that redundancy check is data independent, that is, it holds in general and not only for a given data set (which may change), we check redundancy only on the rule layer of a program. Formally, given messages m_1 and m_2 we say that m_1 is *implied* by m_2 over the rule layer Σ, written $\Sigma \models m_1 \Rightarrow m_2$, if for every data layer $\mathcal{D}$ we have that if $(\mathcal{D}, \Sigma) \models m_1$ then $(\mathcal{D}, \Sigma) \models m_2$. This implication is closely related to the problem of query containment with aggregates over constraints studied in database theory. Already query containment of SQL queries without aggregates is a very difficult task (in fact it is undecidable). Containment with aggregates has been partially studied in a limited settings [7], without negation and nesting. For this reasons, we simplify the definition of redundancy by assuming that aggregates do not change the signal. This obviously eliminates the problem of reasoning over aggregates and numeric functions (e.g., of checking whether filterValue($>$, 20) $\Rightarrow$ filterValue($>$, 10) holds). Moreover, in order to avoid exponential generation of new interval in signal rules (see, [4]), we assume that signal functions are step functions over uniform size intervals and that signal expression are following this interval granularity when defining new

Complexity	*Data*	*Combined*
Firing	AC^0	PTime
Redundancy	n.a.	coNP-c
Consistency	n.a.	coNP-c
Provenance	PTime	PTime

Table 1: Computational complexity; '-c' means 'complete'.

signals. Under these assumptions, one can verify whether there is a redundancy between two message rules by unfolding bodies of these rules following the signal expressions, turning the resulting expressions into propositional boolean formulae, and by checking that they are jointly satisfiable.

Consistency of messages. Another important task in rule maintenance is checking if two messages are behaving consistently with their meaning. In particular, we check whether for a given rule layer we do not have the case that two messages of an opposite meaning are fired at the same. For instance, we want to ensure that we composed rules such that our system is not firing that "rotor is overheating" and "rotor is not overheating." Analogously to redundancy, we do such check independently of data, that quantify consistency for any data layer. Formally, given messages m_1 and m_2, that should not fire simultaneously, we say that m_1 is *consistent* with m_2 over the rule layer Σ, written $\Sigma \models \mathsf{consist}(m_1, m_2)$ if for every data layer $\mathcal{D}$ we have that if $(\mathcal{D}, \Sigma) \models m_1$ then $(\mathcal{D}, \Sigma) \not\models m_2$ and *vice versa*. We observe that the consistency problem can be reduced to redundancy and vice versa. Namely, let $m_2 \leftarrow D$ and $m_2^{\neg} \leftarrow \mathit{not}\, D$, then $\Sigma \models \mathsf{consist}(m_1, m_2)$ iff $\Sigma \models m_1 \Rightarrow m_2^{\neg}$. Hence, one can check consistency by adapting the procedure for checking redundancy.

Provenance of messages. Finally, we consider another important practical task: when a message is fired a diagnostic engineer would like to know the reason for this. For example, which signals caused the firing. In this case, we are interested in finding minimal (w.r.t. set inclusion) sub-programs of Π that fire the message. Notice that there may exist several such minimal sub-programs. One of these can be computed by iteratively removing all superfluous axioms, until only relevant ones remain [8].

Computational Complexity. Now we analyze the computational complexity of each of the tasks above. We refer to these tasks as: *Firing*, *Redundant*, *Consistency* and *Provenance*. For *Firing* and *Provenance*, we distinguish between *data* and *combined* complexity. Data complexity is the complexity of a problem when all parameters are fixed except for the data layer.

The complexity results we obtained are summarized in Table 1. In the following we provide intuitions for each of the tasks. The bottom up approach for firing messages gives that the problem of *firing* a message can be decided in PTime in combined complexity for the following reasons. *(i)* For each concept classification in OWL 2 QL can be computed in PTime [5], and each basic concepts has at most polynomial many subconcepts in OWL 2 QL. *(ii)* Each filter and arithmetic operation in signal expressions can be computed in PTime. Only the PTime complexity of alignFilter is not obvious because it is operating on two concepts at the same time, however, since it outputs only signal from the first concepts concatenating such filters is still in PTime. *(iii)* Finally, evaluating Boolean expressions is in also in PTime and thus it is firing a message as well. The problem of *firing* is in AC^0 in data complexity since we can create one (large) first-order logic query [5] by unfolding Boolean expressions, signal expressions and ontology and then checking firing as query evaluation. Regarding *redundancy* and *consistency*, the membership in coNP follows from verification approach we proposed, and the hardness from coNP-hardness of unsatisfiability problem for Boolean formulas. For *provenance*, the PTime upper bound for deciding firing of rules implies that one minimal sub-program that fires them is computable also in polynomial time. However, computing all of them, or just those of minimal size is known to be a harder problem [16].

4 CONCLUSIONS

In this work we presented an ongoing work that aims at both a high level diagnostic language and a system that can support both authoring and maintenance of diagnostic programs. We have already implemented some of our ideas [12, 13]. Our future work includes: evaluation of our proposal both with users and for scalability; extension of our results on redundancy to cover the case of unrestricted programs; other important maintenance task for diagnostic programs, e.g., automatic debugging.

Acknowledgements. This research is supported by the EPSRC projects MaSI^3, DBOnto, ED^3; and by the FUB project QUEST.

REFERENCES

[1] A. Artale, R. Kontchakov, F. Wolter, and M. Zakharyaschev. Temporal description logic for ontology-based data access. In *IJCAI 2013*, pages 711–717, 2013.
[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
[3] S. Borgwardt, M. Lippmann, and V. Thost. Temporal query answering in the description logic dl-lite. In *FroCoS*, pages 165–180, 2013.
[4] S. Brandt, E. Kalaycı, R. Kontchakov, V. Ryzhikov, G. Xiao, and M. Zakharyaschev. Ontology-based data access with a horn fragment of metric temporal logic. In *AAAI*, 2017.
[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *JAR*, 39(3), 2007.
[6] A. Chen, Y. Wu, A. Haeberlen, B. T. Loo, and W. Zhou. Data provenance at internet scale: Architecture, experiences, and the road ahead. In *CIDR*, 2017.
[7] S. Cohen, W. Nutt, and Y. Sagiv. Containment of aggregate queries. In *ICDT*, pages 111–125, 2003.
[8] A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *ISWC*, pages 267–280, 2007.
[9] E. Kharlamov, S. Brandt, E. Jiménez-Ruiz, Y. Kotidis, S. Lamparter, T. Mailis, C. Neuenstadt, Ö. Özçep, C. Pinkel, C. Svingos, D. Zheleznyakov, I. Horrocks, Y. E. Ioannidis, and R. Möller. Ontology-Based Integration of Streaming and Static Relational Data with Optique. In *SIGMOD*, 2016.
[10] E. Kharlamov, N. Solomakhina, Ö. Özçep, D. Zheleznyakov, T. Hubauer, S. Lamparter, M. Roshchin, A. Soylu, and S. Watson. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In *ISWC*, pages 601–619, 2014.
[11] B. T. Loo, T. Condie, M. N. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking. *CACM*, 52(11):87–95, 2009.
[12] G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. Kalayci, S. Brandt, I. Horrocks, M. Roshchin, and T. Runkler. SemDia: Semantic rule-based equipment diagnostics tool. In *CIKM*, 2017.
[13] G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. Kalayci, S. Brandt, I. Horrocks, M. Roshchin, and T. Runkler. Semantic rule-based equipment diagnostics. In *ISWC*, 2017.
[14] G. Mehdi, Evgeny Kharlamov, Ognjen Savkovic, Guohui Xiao, Elem Guzel Kalayci, Sebastean Brandt, Iand Horrocks, Michail Roshchin, and Thomas Runkler. Semantic rules for siemens turbines. In *ISWC Posters & Demos*, 2017.
[15] J. S. Mitchell. *An introduction to machinery analysis and monitoring*. Pennwell Books, 1993.
[16] R. Peñaloza and B. Sertkaya. Complexity of axiom pinpointing in the dl-lite family of description logics. In *ECAI*, pages 29–34, 2010.
[17] R. Bond Randall. *Vibration-based condition monitoring: industrial, aerospace and automotive applications*. Wiley, 2011.
[18] B. K. N. Rao. *Handbook of condition monitoring*. Elsevier, 1996.
[19] G. Vachtsevanos, F. L. Lewis, M. Roemer, A. Hess, and B. Wu. *Intelligent Fault Diagnosis and Prognosis for Engineering Systems*. Wiley, 2006.