Semantic Rule-Based Equipment Diagnostics

Gulnar Mehdi[1,2(✉)], E. Kharlamov[3], Ognjen Savković[4], G. Xiao[4], E. Güzel Kalaycı[4], S. Brandt[1], I. Horrocks[3], Mikhail Roshchin[1], and Thomas Runkler[1,2]

[1] Siemens CT, Munich, Germany
gulnar.mehdi@siemens.com
[2] Technical University of Munich, Munich, Germany
[3] University of Oxford, Oxford, UK
[4] Free University of Bozen-Bolzano, Bolzano, Italy

Abstract. Industrial rule-based diagnostic systems are often data-dependant in the sense that they rely on specific characteristics of individual pieces of equipment. This dependence poses significant challenges in rule authoring, reuse, and maintenance by engineers. In this work we address these problems by relying on Ontology-Based Data Access: we use ontologies to mediate the equipment and the rules. We propose a semantic rule language, *sigRL*, where sensor signals are first class citizens. Our language offers a balance of expressive power, usability, and efficiency: it captures most of Siemens data-driven diagnostic rules, significantly simplifies authoring of diagnostic tasks, and allows to efficiently rewrite semantic rules from ontologies to data and execute over data. We implemented our approach in a semantic diagnostic system, deployed it in Siemens, and conducted experiments to demonstrate both usability and efficiency.

1 Introduction

Intelligent *diagnostic systems* play an important role in industry since they help to maximise equipment's up-time and minimise its maintenance and operating costs [29]. In energy sector companies like Siemens often rely on *rule-based* diagnostics systems to analyse power generating equipment by, e.g., testing newly deployed electricity generating gas turbines [24], or checking vibration instrumentation [26], performance degradation [27], and faults in operating turbines. For this purpose diagnostic engineers create and use complex diagnostic rule-sets to detect abnormalities during equipment run time and sophisticated analytical models to combine these abnormalities with models of physical aspects of equipment such as thermodynamics and energy efficacy.

An important class of rules that are commonly used in Siemens for rule-based turbines diagnostics are *signal processing rules* (SPRs). SPRs allow to filter, aggregate, combine, and compare *signals*, that are time stamped measurement values, coming from sensors installed in equipment and trigger error or notification messages when a certain criterion has been met. Thus, sensors report

DOI: 10.1007/978-3-319-68204-4_29

temperature, pressure, vibration and other relevant parameters of equipment and SPRs process this data and alert whenever a certain pattern is detected. Rule-based diagnostics with SPRs can be summarises as in Fig. 1 (left), where the data consists of signals coming from sensors of turbines, and the diagnostic layer consists of SPRs and analytical models.

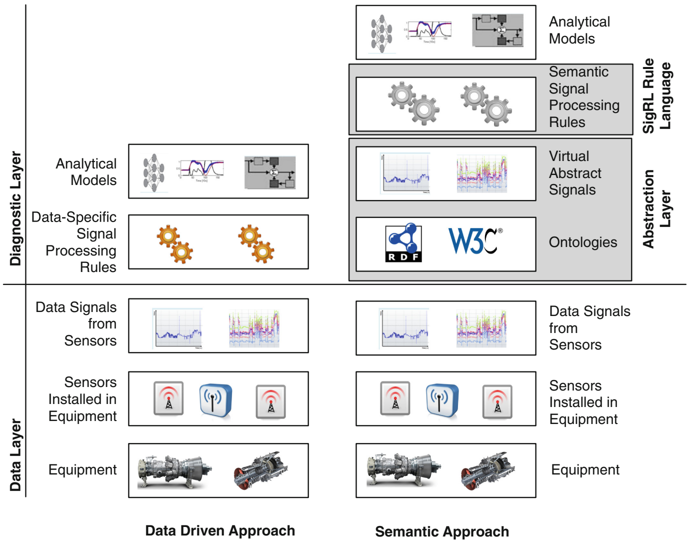


Fig. 1. General scheme of diagnostics with signal-processing rules. Left: data driven approach. Right: semantic approach.

SPRs that are currently offered in most of existing diagnostic systems and 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 an engineer has to write from dozens to hundreds of SPRs that involve hundreds of sensor tags, 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 a diagnostic task to detect whether the purging[1] in over can be captured with 300 SPRs. We now illustrate diagnostic tasks and corresponding SPRs on a running example.

Example 1 (Purging Running Example). Consider the purging diagnostic task:

Verify that the purging is over in the main flame component of the turbine T1.

[1] Purging is the process of flushing out liquid fuel nozzles or other parts which may contain undesirable residues.

Intuitively this task requires to check in the turbine T1 that: *(i)* the main flame was on for at least 10 s and then stopped, *(ii)* 15 s after this, the purging of rotors in the starting-component of T1 started, *(iii)* 20 s 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. Let us assume that the first rotor in the starting-component of T1 has three speed sensors 'S21R1T1', 'S22R1T1', and 'S23R1T1' and that the 1.300 and 800 rotations respectively indicate that the purging starts and ends. Then, the following data dependent SPRs written in a syntax similar to the one of Siemens SPRs can be used to detect that purging has started and ended:

$$\$\mathsf{PurgStartRotor1} : \mathsf{truth}(\mathsf{avg}(\text{`}\mathsf{S21R1T1}\text{'}, \text{`}\mathsf{S22R1T1}\text{'}, \text{`}\mathsf{S23R1T1}\text{'}), > 1.300). \tag{1}$$

$$\$\mathsf{PurgStopRotor1} : \mathsf{truth}(\mathsf{avg}(\text{`}\mathsf{S21R1T1}\text{'}, \text{`}\mathsf{S22R1T1}\text{'}, \text{`}\mathsf{S23R1T1}\text{'}), < 800). \tag{2}$$

Here $PurgStartRotor1 and $PurgStopRotor1 are variables that store those average time stamped values of measurements (from the three speed sensors in the first rotor of T1) that passed the threshold. Complete encoding of the purging task for an average Siemens turbine requires to write around 300 SPRs some of which are as in the running example. Many of these SPRs differ only on specific sensor identifiers and the number of speed signals to aggregate. Adapting these SPRs to another turbine will also require to change a lot of identifiers. For example, in another turbine T2 the rotor may have the purging threshold values 1.000 and 700 and contain four sensors 'S01R2T2', 'S02R2T2', 'S03R2T2', and 'S04R2T2 mrq and thus the corresponding start and stop purging rules will be as above but with these new sensors ids and threshold values. □

Data dependence of SPRs poses three significant challenges for diagnostic engineers in *(i)* authoring, *(ii)* reuse, and *(iii)* maintenance of SPRs. Indeed, authoring such rules is time consuming and error prone, e.g., while aggregating the speed signals from a given rotor one has ensure that all the relevant speed signals are included in the aggregation and that other signals, e.g., temperature signals, are not included. As the result, in the overall time that a Siemens engineer spends on diagnostics up to 80% is devoted to rule authoring where the major part of this time is devoted to data access and integration [19]. Reuse of such rules is limited since they are too specific to concrete equipment and in many cases it is easier to write a new rule set than to understand and adapt an existing one. As the result, over the years Siemens has acquired a huge library of SPRs with more than 200,000 rules and it constantly grows. Maintenance of such SPRs is also challenging and require significant manual work since there is limited semantics behind them.

Adding Semantics to SPRs. Semantic technologies can help in addressing these three challenges. An *ontology* can be used to abstractly represent sensors and background knowledge about turbines including locations of sensors, structure and characteristics of turbines. Then, in the spirit of *Ontology Based Data Access* (OBDA) [25], the ontology can be 'connected' to the data about the actual turbines, their sensors and signals with the help of declarative *mapping specifications*. OBDA has recently attracted a lot of attention by the research community: a solid theory has been developed, e.g. [7,28], and a number of mature systems have been implemented, e.g. [5,6]. Moreover, OBDA has been successfully applied in several industrial applications, e.g. [8,14,15,17].

Adopting OBDA for rule-based diagnostics in Siemens, however, requires a rule based language for SPRs that enjoys the following features:

(i) Signals orientation: The language should treat signals as first class citizens and allow for their manipulation: to filter, aggregate, combine, and compare signals;
(ii) Expressiveness: The language should capture most of the features of the Siemens rule language used for diagnostics;
(iii) Usability: The language should be simple and concise enough so that the engineers can significantly save time in specifying diagnostic tasks;
(iv) Efficiency: The language should allow for efficient execution of diagnostic tasks.

To the best of our knowledge no rule language exists that fulfills all these requirements (see related work in Sect. 5 for more details).

Contributions. In this work we propose to extend the traditional data driven approach to diagnostics with an OBDA layer and a new rule language to what we call *Semantic Rule-based Diagnostics*. Our approach is schematically depicted in Fig. 1 (right). To this end we propose a language *sigRL* for SPRs that enjoys the four requirements above. Our language allows to write SPRs and complex diagnostic tasks in an abstract fashion and to exploit both ontological vocabulary and queries over ontologies to identify relevant sensors and data values. We designed the language 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, it has good computational properties. In particular, *sigRL* allows for rewriting [7] of diagnostic rule-sets written over OWL 2 QL ontologies [2] into multiple data-dependent rule-sets with the help of ontologies and OBDA mappings. This rewriting allows to exploit standard infrastructure, including the one used in Siemens, for processing data-dependent SPRs.

We implemented *sigRL* and a prototypical Semantic Rule-based Diagnostic system. We deployed our implementation in Siemens over 50 Siemens gas turbines and evaluated the deployment with encouraging results. We evaluated usability of our solution with Siemens engineers by checking how fast they are in formulating diagnostic tasks in *sigRL*. We also evaluated the efficiency of our solution in processing diagnostic tasks over turbine signals in a controlled environment. Currently, our deployment is not included in the production processes, it is a prototype that we plan to evaluate and improve further before it can be used in production.

[2] OWL 2 QL is the W3C standardised ontology language that is intended for OBDA.

2 Signal Processing Language *sigRL*

In this section we introduce our signal processing language *sigRL*. It has three components: *(i)* Basic signals that come from sensors; *(ii) Knowledge Bases* (KBs) that capture background knowledge of equipment and signals as well as concrete characteristics of the equipment that undergoing diagnostics, and *(iii)* Signal processing expressions that manipulate basic signals using mathematical functions and queries over KBs.

Signals. In our setting, a *signal* is a first-class citizen. A signal s is a pair (o_s, f_s) where o_s is *sensor id* and *signal function* f_s defined on $\mathbb{R}$ to $\mathbb{R} \cup \{\perp\}$, where $\perp$ denotes the absence of a value. A *basic signal* is a signal which reading is obtained from a single sensor (e.g., in a turbine) for different time points. In practice, it may happen that a signal have periods without identified values. Also, such periods are obtained when combining and manipulating basic signals. We say that a signal s is *defined* on a real interval I if it has a value for each point of the interval, $\perp \notin f_s(I)$. For technical reasons we introduce *undefined* signal function $f_\perp$ that maps all reals into $\perp$. In practice signals are typically step functions over time intervals since they correspond to sensor values delivered with some frequency. In our model, we assume that we are given a finite set of basic signals $\mathcal{S} = \{s_1, \ldots, s_n\}$.

Knowledge Bases and Queries. A Knowledge Base $\mathcal{K}$ is a pair of an *ontology* $\mathcal{O}$ and a *data set* $\mathcal{A}$. An *ontology* describes background knowledge of an application domain in a formal language. We refer the reader to [7] for detailed definitions of ontologies. In our setting we consider ontologies that 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 expression that says that $\mathsf{RotorSensor}$ is a kind of $\mathsf{SpeedSensor}$:

$$\mathsf{SubClassOf}(\mathsf{RotorSensor}\ \ \mathsf{SpeedSensor}). \tag{3}$$

Data sets of KBs consist of *data assertions* enumerating concrete sensors, turbines, and their components. The following assertions says that sensors '$\mathsf{S21R1T1}$','$\mathsf{S22R1T1}$' and '$\mathsf{S23R1T1}$' are all rotor sensors:

$$\begin{aligned}&\mathsf{ClassAssertion}(\mathsf{RotorSensor}\ \text{`}\mathsf{S21R1T1}\text{'}), \mathsf{ClassAssertion}(\mathsf{RotorSensor}\ \text{`}\mathsf{S22R1T1}\text{'}),\\ &\mathsf{ClassAssertion}(\mathsf{RotorSensor}\ \text{`}\mathsf{S23R1T1}\text{'}).\end{aligned} \tag{4}$$

In order to enjoy favorable semantic and computational characteristics of OBDA, we consider well-studied ontology language OWL 2 QL that allows to express subclass (resp. sub-property) axioms between classes and projections of properties (resp. corollary between properties). We refer the reader to [7] for details on OWL 2 QL.

To query KBs we rely on conjunctive queries (CQs) and certain answer semantics that have been extensively studied for OWL 2 QL KBs and proved to be tractable [7]. For example, the following CQ returns all rotor sensors from start component:

$$\mathsf{rotorStart}(x) \leftarrow \mathsf{rotorSensor}(x) \wedge \mathsf{locatedIn}(x,y) \wedge \mathsf{startComponent}(y). \qquad (5)$$

Signals Processing Expressions. Now we introduce signal expressions that filter and manipulate basic signals and create new complex signals. Intuitively, in our language we group signals into ontological concepts and signal expression are defined on the level of concepts. Then, a *signal processing expression* is recursively defined as follows:

$$\begin{array}{llllll} C & = & Q & \mid C_1 : \mathit{value}(\odot, \alpha) & \mid \{s_1, \ldots, s_m\} & \mid \mathsf{agg}\ C_1 \quad \mid \\ & & \alpha \circ C & \mid C_1 : \mathit{duration}(\odot, t) & \mid C_1 :\ \mathit{align}\ C_2 & \end{array}$$

where C is a concept, Q is a CQ, $\circ$ is in $\{+,-,\times,/\}$, agg is in $\{\mathsf{min}, \mathsf{max}, \mathsf{avg}, \mathsf{sum}\}$, $\alpha \in \mathbb{R}$, $\odot \in \{<,>,\leq,\geq\}$ and $\mathit{align} \in \{\mathit{within}, \mathit{after}[t], \mathit{before}[t]\}$ where t is a period.

Table 1. Meaning of signal processing expressions. For the interval I, size(I) is its size. For intervals I_1, I_2 the *alignment* is: "I_1 *within* I_2" if $I_1 \subseteq I_2$; "I_1 *after*[t] I_2" if all points of I_2 are after I_1 and the start of I_2 is within the end of I_1 plus period t; "I_1 *before*[t] I_2" if "I_2 *start*[t] I_1".

$C =$	Concept C contains
Q	All signal ids return by Q evaluated over the KB
$\alpha \circ C_1$	One signal s' for each signal s in C_1 with $f_{s'} = \alpha \circ f_s$
$C_1 : \mathit{value}(\odot, \alpha)$	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 $f_{s'}(t) = \perp$
$C_1 : \mathit{duration}(\odot, t')$	One signal s' for each signal s in C_1 with $f_{s'}(t) = f_s(t)$ if exists an interval I such that: f_s is defined I, $t \in I$ and $\mathit{size}(I) \odot t'$; otherwise $f_{s'}(t) = \perp$
$\{s_1, \ldots, s_m\}$	All enumerated signal $\{s_1, \ldots, s_m\}$
$C = \mathsf{agg}\ C_1$	One signal s' with $f_{s'}(t) = \mathsf{agg}_{s \in C_1} f_s(t)$, that is, s' is obtained from all signals in C_1 by applying the aggregate agg at each time point t
$C_1 :\ \mathit{align}\ C_2$	A signal s_1 from C_1 if: there exists a signal s_2 from C_2 that is *aligned* with s_1, i.e., for each time interval I_1 where f_{s_1} is defined there is an interval I_2 where f_{s_2} is defined s.t. I_1 *aligns* with I_2

The formal meaning of signal processing expressions is defined in Table 1. In order to make the mathematics right, we assume that $c \circ \perp = \perp \circ c = \perp$ and

$c \odot \bot = \bot \odot c = \mathit{false}$ for $c \in \mathbb{R}$, and analogously we assume for aggregate functions. If the value of a signal function at a time point is not defined with these rules, then we define it as $\bot$.

Example 2. The start and end of a purging process data-driven rules as in Eqs. (1) and (2) from the running example can be expressed in *sigRL* as follows:

$$\mathsf{PurgingStart} = \mathit{avg}\ \mathsf{rotorStart} : \mathsf{value}(>, \mathit{purgingSpeed}), \tag{6}$$

$$\mathsf{PurgingStop} = \mathit{avg}\ \mathsf{rotorStart} : \mathsf{value}(<, \mathit{nonPurgingSpeed}). \tag{7}$$

Here, rotorStart is the CQ defined in Eq. (5). For brevity we do not introduce a new concept for each expression but we just join them with symbol ":". Constants *purgingSpeed* and *nonPurgingSpeed* are parameters of an analysed turbine, and they are instantiated from the turbine configuration when the expressions are evaluated. □

Diagnostic Programs and Messages. We now show how to use signal expressions to compose diagnostic programs and to alert messages. In the following we will consider *well formed* sets of signal expressions, that is, sets where each concept is defined at most once and where definitions of new concepts are assumed to be acyclic: if C_1 is used to define C_2 (directly or indirectly) then C_1 cannot be defined (directly or indirectly) using C_1.

A *diagnostic program* (or simply *program*) Π is a tuple $(\mathcal{S}, \mathcal{K}, \mathcal{M})$ where $\mathcal{S}$ is a set of basic signals, $\mathcal{K}$ is a KB, $\mathcal{M}$ is a set of well formed signal processing expressions such that each concept that is defined in $\mathcal{M}$ does not appear in $\mathcal{K}$.

Example 3. The running example program $\Pi = (\mathcal{S}, \mathcal{K}, \mathcal{M})$ has the following components: sensors $\mathcal{S} = \{$'S21R1T1', 'S22R1T1', 'S23R1T1'$\}$, KB $\mathcal{K}$ that consists of axioms from Eqs. (3)–(4), and $\mathcal{M}$ that consists of expressions from Eqs. (6)–(7). □

A *message rule* is a rule of the form, where C is a concept and m is a (text) message:

$$\mathit{message}(m) = C.$$

Example 4. Using expressions (6) and (7) we define the following message:

$$\begin{aligned}\mathsf{message}(\textit{“Purging over”}) = \mathsf{FlameSensor} : \mathsf{duration}(>, 10s) :&\\ \mathit{after}[15s]\ \mathsf{PurgingStart} : \mathit{after}[20s]\ \mathsf{PurgingStop}&\end{aligned} \tag{8}$$

The message intuitively indicates that the purging is over. □

Now we are ready to define the semantics of the rules, expression and programs.

Semantics of ***sigRL.*** We now define how to determine whether a program Π fires a rule r. To this end, we extend first-order interpretations that are used to define semantics of OWL 2 KBs. In OWL 2 a first class citizen is an object o and interpretation is defining whether $C(o)$ is true or not for particular concept C. In our scenario, domain of objects is a domain of sensor ids (basic or ones defined by expressions). Thus each object o is also having assigned function f_o that represents the signal value of that object. Obviously, an identifier o can also be an id of a turbine component that does not have signal function. At the moment, (since it is not crucial for this study and it simplifies the formalism) we also assign undefined signal $f_\perp$ to such (non-signal) objects.

Formally, our *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) and $\mathcal{I}_{\mathcal{S}}$ interprets signals. First, we define how $\mathcal{I}$ interprets basic signals. Given a set of signals for an interpretation $\mathcal{I}$: $\mathcal{S}^\mathcal{I} = \{s_1^\mathcal{I}, \ldots, s_n^\mathcal{I}\}$ s.t. $\mathcal{I}_{FOL}$ 'returns' the signal id, $s^{\mathcal{I}_{FOL}} = o_s$ and $\mathcal{I}_{\mathcal{S}}$ 'returns' the signal itself, $s^{\mathcal{I}_{\mathcal{S}}} = s$.

Now we can define how $\mathcal{I}$ interprets KBs. Interpretation of a KB $\mathcal{K}^\mathcal{I}$ extends the notion of first-order logics interpretation as follows: $\mathcal{K}^{\mathcal{I}_{FOL}}$ is a first-order logics interpretation $\mathcal{K}$ and $\mathcal{K}^{\mathcal{I}_{\mathcal{S}}}$ is defined for objects, concepts, roles and attributes following $S^\mathcal{I}$. That is, for each object o we define $o^{\mathcal{I}_{\mathcal{S}}}$ as s if o is the id of s from $\mathcal{S}$; otherwise $(o, f_\perp)$. Then, for a concept A we define $A^{\mathcal{I}_{\mathcal{S}}} = \{s^{\mathcal{I}_{\mathcal{S}}} \mid o_s^{\mathcal{I}_{FOL}} \in A^{\mathcal{I}_{FOL}}\}$. Similarly, we define $\cdot^{\mathcal{I}_{\mathcal{S}}}$ for roles and attributes.

Finally, we are ready to define $\mathcal{I}$ for signal expressions and we do it recursively following the definitions in Table 1. We now illustrate some of them. For example, if $C = \{s_1, \ldots, s_m\}$, then $C^\mathcal{I} = \{s_1^\mathcal{I}, \ldots, s_m^\mathcal{I}\}$; if $C = Q$ then $C^{\mathcal{I}_{FOL}} = Q^{\mathcal{I}_{FOL}}$ where $Q^{\mathcal{I}_{FOL}}$ is the evaluation of Q over $\mathcal{I}_{FOL}$ and $C^{\mathcal{I}_{\mathcal{S}}} = \{s \mid o_s^{\mathcal{I}_{FOL}} \in Q^{\mathcal{I}_{FOL}}\}$, provided that $\mathcal{I}_{FOL}$ is a model of $\mathcal{K}$. Otherwise we define $C^\mathcal{I} = \emptyset$. Similarly, we define interpretation of the other expressions.

Firing a Message. Let Π be a program and 'r : *message*$(m) = C$' a message rule. We say that Π *fires* message r if *for each* interpretation $\mathcal{I} = (\mathcal{I}_{FOL}, \mathcal{I}_{\mathcal{S}})$ of Π it holds $C^{\mathcal{I}_{FOL}} \neq \emptyset$, that is, the concept that fires r is not empty. Our programs and rules enjoy the *canonical* model property, that is, each program has a unique (Hilbert) interpretation [3] which is minimal and can be constructed starting from basic signals and ontology by following signal expressions. Thus, one can verify $C^{\mathcal{I}_{FOL}} \neq \emptyset$ only on the canonical model. This implies that one can evaluate *sigRL* programs and expressions in a bottom-up fashion. We now illustrate this approach on our running example.

Example 5. Consider our running program Π from Example 3 and its canonical interpretation $\mathcal{I}_\Pi$. First, for each query Q in $\mathcal{M}$ we evaluate Q over KB $\mathcal{K}$ by computing $Q^{\mathcal{I}_\Pi}$. In our case, the only query is $\mathsf{rotorStart}$ that collects all sensor ids for a particular turbine. Then, we evaluate the expressions in $\mathcal{M}$ following the dependency graph of definitions. We start by evaluation the expression from Eq. (6), again in a bottom-up fashion. Concept $\mathsf{rotorStart}^{\mathcal{I}_\Pi}$ contains sensor ids: '$\mathsf{S21R1T1}$','$\mathsf{S22R1T1}$' and '$\mathsf{S23R1T1}$'. At the same time, those sensors have signal functions assigned from $\mathcal{S}^{\mathcal{I}_\Pi}$. 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, value$(>, \mathit{purgingSpeed})$ eliminates all values of s_4 that are not $> \mathit{purgingSpeed}$. Similarly, we compute signal transformations for the expression from Eq. (6). Finally, we use those two expressions to evaluate the message rule from Eq. (8). If there exists at least one FlameSensor that aligns with one signal in evaluated expressions corresponding to Eqs. (6) and (7), then the message is fired. □

3 System Implementation and Deployment in Siemens

System Implementation. The main functionality of our Semantic Rule-based Diagnostics system is to author *sigRL* diagnostic programs, to deploy them in turbines, to execute the programs, and to visualise the results of execution. We now give details of our system by following its architecture in Fig. 2. There are four essential layers in the architecture: application, OBDA, rule execution, and data. Our system is mostly implemented in Java. We now discuss the system layer by layer.

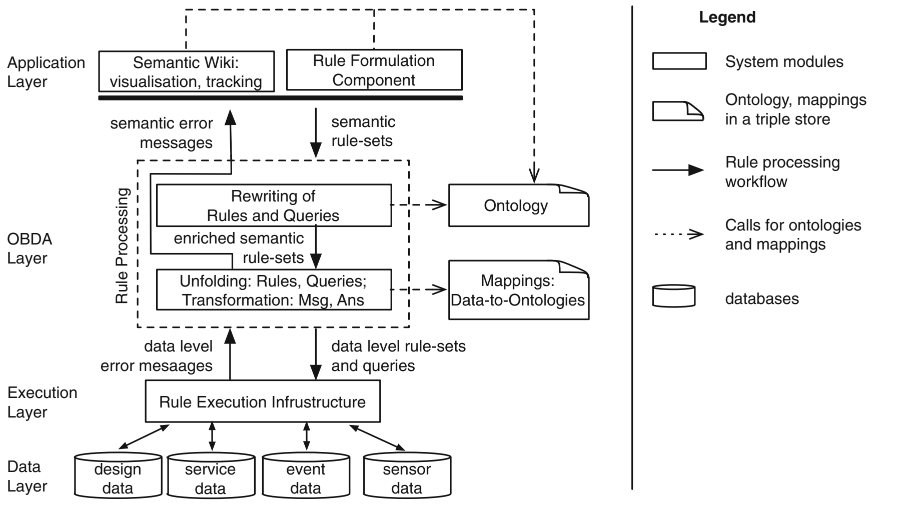
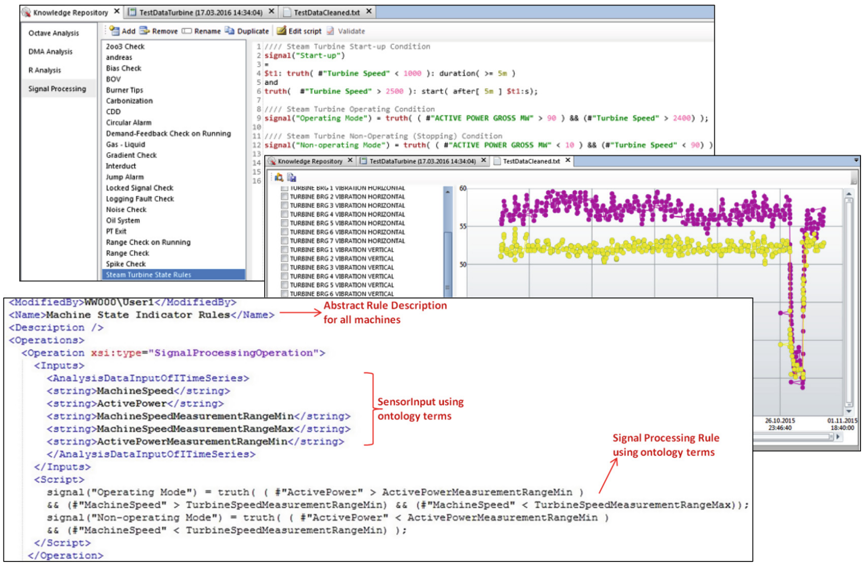

Application Layer
Semantic Wiki: visualisation, tracking
Rule Formulation Component
semantic error messages
semantic rule-sets
OBDA Layer
Rule Processing
Rewriting of Rules and Queries
enriched semantic rule-sets
Unfolding: Rules, Queries; Transformation: Msg, Ans
Ontology
Mappings: Data-to-Ontologies
data level error mesaages
data level rule-sets and queries
Execution Layer
Rule Execution Infrustructure
Data Layer
design data
service data
event data
sensor data
Legend
System modules
Ontology, mappings in a triple store
Rule processing workflow
Calls for ontologies and mappings
databases

Fig. 2. Architecture of our Semantic Rule-based Diagnostics system.

On the *application layer*, the system offers two user-oriented modules. The first module allows engineers to author, store, and load diagnostic programs by formulating sets of SPRs in *sigRL* and sensor retrieving queries. Such formulation is guided by the domain ontology stored in the system. In Fig. 3 (top) one can observe a screenshot of the SPR editor which is embedded in the Siemens analytical toolkit. Another module is the semantic Wiki that allows among other features to visualize signals and messages (triggered by programs), and to track deployment of SPRs in equipment. In Fig. 3 (center) one can see visualization of signals from two components of one turbine. Diagnostic programs formulated in the application layer are converted into XML-based specifications and sent to the OBDA layer, which returns back the messages and materialized semantic signals, that is, signals over the ontological terms. In Fig. 3 (bottom) one can see an excerpt from an XML-based specification. We rely on REST API to communicate between the application layer and the OBDA layer of our system and OWL API to deal with ontologies.


Knowledge Repository
TestDataTurbine (17.03.2016 14:34:04)
TestDataCleaned.txt
Octave Analysis
DMA Analysis
R Analysis
Signal Processing
Add
Remove
Rename
Duplicate
Edit script
Validate
2oo3 Check
andreas
Bias Check
BOV
Burner Tips
Carbonization
CDD
Circular Alarm
Demand-Feedback Check on Running
Gas - Liquid
Gradient Check
Interduct
Jump Alarm
Locked Signal Check
Logging Fault Check
Noise Check
Oil System
PT Exit
Range Check on Running
Range Check
Spike Check
Steam Turbine State Rules
TURBINE BRG 1 VIBRATION HORIZONTAL
TURBINE BRG 2 VIBRATION HORIZONTAL
TURBINE BRG 3 VIBRATION HORIZONTAL
TURBINE BRG 4 VIBRATION HORIZONTAL
TURBINE BRG 5 VIBRATION HORIZONTAL
TURBINE BRG 6 VIBRATION HORIZONTAL
TURBINE BRG 7 VIBRATION HORIZONTAL
TURBINE BRG 1 VIBRATION VERTICAL
TURBINE BRG 2 VIBRATION VERTICAL
TURBINE BRG 3 VIBRATION VERTICAL
TURBINE BRG 4 VIBRATION VERTICAL
TURBINE BRG 5 VIBRATION VERTICAL
TURBINE BRG 6 VIBRATION VERTICAL
60
55
50
26.10.2015 23:46:40
01.11.2015 18:40:00
Abstract Rule Description for all machines
SensorInput using ontology terms
Signal Processing Rule using ontology terms

```
//// Steam Turbine Start-up Condition
signal("Start-up")
=
$t1: truth( #"Turbine Speed" < 1000 ): duration( >= 5m )
and
truth(  #"Turbine Speed" > 2500 ): start( after[ 5m ] $t1:s);

//// Steam Turbine Operating Condition
signal("Operating Mode") = truth( ( #"ACTIVE POWER GROSS MW" > 90 ) && (#"Turbine Speed" > 2400) );

//// Steam Turbine Non-Operating (Stopping) Condition
signal("Non-operating Mode") = truth( ( #"ACTIVE POWER GROSS MW" < 10 ) && (#"Turbine Speed" < 90) )
```

```
<ModifiedBy>WW000\User1</ModifiedBy>
<Name>Machine State Indicator Rules</Name>
<Description />
<Operations>
  <Operation xsi:type="SignalProcessingOperation">
    <Inputs>
      <AnalysisDataInputOfITimeSeries>
      <string>MachineSpeed</string>
      <string>ActivePower</string>
      <string>MachineSpeedMeasurementRangeMin</string>
      <string>MachineSpeedMeasurementRangeMax</string>
      <string>ActivePowerMeasurementRangeMin</string>
      </AnalysisDataInputOfITimeSeries>
    </Inputs>
    <Script>
      signal("Operating Mode") = truth( ( #"ActivePower" > ActivePowerMeasurementRangeMin )
      && (#"MachineSpeed" > TurbineSpeedMeasurementRangeMin) && (#"MachineSpeed" < TurbineSpeedMeasurementRangeMax));
      signal("Non-operating Mode") = truth( ( #"ActivePower" < ActivePowerMeasurementRangeMin )
      && (#"MachineSpeed" < TurbineSpeedMeasurementRangeMin) );
    </Script>
  </Operation>
```

Fig. 3. Screenshots: SPR editor (top), Wiki-based visualisation monitor for semantic signals (centre), and a fragment of an XML-based specification of an SPR (bottom).

Note that during the course of the project we have developed an extension to the existing Siemens rule editor and a dedicated wiki-based visualisation monitor for semantic signals. Note that we use the latter for visualising query answers and messages formatted according to the Siemens OWL 2 ontology and stored as RDF.

The *OBDA layer* takes care of transforming SPRs written in *sigRL* into either SPRs written in the Siemens data-driven rule language (in order to support the existing Siemens IT infrastructure) or SQL. This transformation has two steps: rewriting of programs and queries with the help of ontologies (at this step both programs and queries are enriched with the implicit information from the ontology), and then unfolding them with the help of mappings. For this purpose we extended the query transformation module of the Optique platform [11–13, 15, 19] which we were developing earlier within the Optique project [10].

The OBDA layer also transforms signals, query answers, and messages from the data to semantic representation.[3]

The *rule execution layer* takes care of planning and executing data-driven rules and queries received from the OBDA layer. If the received rules are in the Siemens SPR language then the rule executor instantiates them with concrete sensors extracted with queries and passes them to the Drools Fusion (drools.jboss.org/drools-fusion.html) the engine used by Siemens. If the received rules are in SQL then it plans the execution order and executes them together with the other queries. To evaluate the efficiency of our system in Sect. 4 we assume that the received rules are in SQL. Finally, on the *data layer* we store all the relevant data: turbine design specifications, historical information about services that were performed over the turbines, previously detected events, and the raw sensor signals.

Deployment at Siemens. We deployed our Semantic Rule-Based Diagnostics system on the data gathered for 2 years from 50 gas power generating turbines. We rely on Teradata for signals and MS SQL for other information. For rule processing, we connected our system to the Siemens deployment of Drools Fusion.

An important aspect of the deployment was the development of a diagnostic ontology and mappings. Our ontology was inspired by the *(i)* Siemens Technical System Ontology (TSO) and Semantic Sensor Network Ontology (SSN) and *(ii)* the international standards IEC 81346 and ISO/TS 16952-10. The development of the ontology was a joint effort of domain experts from Siemens businesses units together with the specialist from the Siemens Corporate Technology. Our ontology consists of four modules and it is expressed in OWL 2 QL. In order to connect the ontology to the data, we introduced 376 R2RML mappings. Note that the development of the ontology and mappings is done offline and it does not affect the time the engineers spend to author rules when they do turbine diagnostics. We now go through the ontology modules in more detail.

The main module of our ontology in partially depicted in Fig. 4 where in grey we present SSN and with white TSO terms. This module has 48 classes and 32 object and data properties. The other three modules are respectively about equipment, sensing devices, and diagnostic rules. They provide detailed information about the machines, their deployment profiles, sensor configurations, component hierarchies, functional profiles and logical bindings to the analytical rule definitions. More precisely:

– *The Equipment module* describes the internal structure of an industrial system. The main classes of the module are *DeploymentSite* and *Equipment* and they describe the whole facility of system and machines that have been physically deployed and monitored. It also defines the system boundaries, substantial descriptions of the system environment, set of components it operates

[3] In this work we assume that RDF is the semantic data representation.

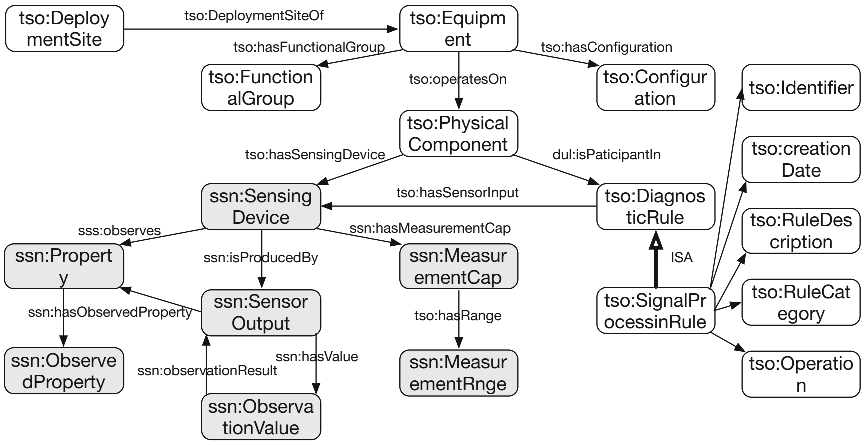


Fig. 4. A fragment of the Siemens ontology that we developed to support turbine diagnostic SPRs.

on, part-of relations, system configurations and functional grouping of components. For example, it encodes that every *Equipment* should have an *ID*, *Configuration* and at least one *Component* to operate on, and an optional property *hasProductLine*.

- *The Sensing Device module* is inspired by the SSN ontology. In particular, we reuse and extend the class *SensingDevice*. The module describes sensors, their properties, outputs, and observations. One of our extensions comparing to SSN is that the measurement capabilities can now include measurement property range i.e. maximum and minimum values (which we encode with annotations in order to keep the ontology in OWL 2 QL). For example, *VibrationSensor* is a sensing device that observes *Vibration* property and measures e.g. *BearingVibrations*.
- *The Diagnostic Rules module* introduces rules and relate them to e.g., *Description*, *Category*, and *Operation* classes. For example, using this module one can say that *SignalProcessingRule* is of a type *DiagnosticRule*, that it has certain *Operation* and it may have different sensor data input associated with its operation.

4 Siemens Experiments

In this section, we present two experiments. The first experiment is to verify whether writing diagnostic programs in *sigRL* offers a considerable time saving comparing to writing the programs in the Siemens data dependent rule language. The second experiment is to evaluate the efficiency of the SQL code generated by our OBDA component (see Sect. 3 for details on our OBDA component).

Preliminary User Study. We conducted a preliminary user study in Siemens with 6 participants, all of them are either engineers or data scientists. In Table 2 we summarise relevant information about the participants. All of them are mid age, most have at least an M.Sc. degree, and all are familiar with the basic concepts of the Semantic Web. Their technical skills in the domain of diagnostics are from 3 to 5. We use a 5-scale range where 1 means 'no' and '5' means 'definitely yes'. Two out of six participants never saw an editor for diagnostic rules, while the other four are quite familiar with rule editors.

During brainstorming sessions with Siemens power generation analysts and R&D personnel from Siemens Corporate Technology we selected 10 diagnostic tasks which can be found in Table 3. The selection criteria were: diversification on topics and complexity, as well as relevance for Siemens. The tasks have three complexity levels (Low, Medium, and High) and they are defined as a weighted sum of the number of sensor tags, event messages, and lines of code in a task.

Table 2. Profile information of participants.

#	Age	Occupation	Education	Tech. skills	Similar tools	Sem. web
P1	43	Design Engineer	Ph.D.	3	3	Yes
P2	46	Senior Diagnostic Engineer	Ph.D.	4	1	Yes
P3	37	Diagnostic Engineer	M.Sc.	5	4	Yes
P4	45	R& D Engineer	M.Sc.	4	4	Yes
P5	34	Software Engineer	B.Sc.	3	3	Yes
P6	33	Data Scientist	Ph.D.	3	1	Yes

Before the study we gave the participants a short introduction with examples about diagnostic programs and message rules in both Siemens and *sigRL* languages. We also explained them the constructs of *sigRL*, presented them our diagnostic ontology, and explained them the data. The data was from 50 Siemens gas turbines and included sensor measurement, that is, temperature, pressure, rotor speed, and positioning, and associated configuration data, that is, types of turbines and threshold values. During the study participants were authoring diagnostic programs for the tasks T1 to T10 from Table 3 using both existing Siemens rule language (as the baseline) and *sigRL*; while we were recording the authoring time. Note that all participants managed to write the diagnostic tasks correctly and the study was conducted on a standard laptop.

Figure 5 summarises the results of the user study. The four left figures present the average time that the six participants took to formulate the 10 tasks over respectively 1, 5, 10, and 50 turbines. We now first discuss how the authoring time changes within each of the four figures, that is, when moving from simple to complex tasks, and then across the four figures, that is, when moving from 1 to 50 turbines.

Observe that in each figure one can see that in the baseline case the authoring time is higher than in the semantic case, i.e., when *sigRL* is used. Moreover, in the

Table 3. Diagnostic tasks for Siemens gas turbines that were used in the user study, where complexity is defined using the number of sensor tags, event messages, and lines of code.

	Complexity	# sensor tags	# event msg	# code lines	Monitoring task
T1	Low	4	2	102	Variable guided vanes analyses
T2	Low	6	5	133	Multiple start attempts
T3	Low	6	3	149	Lube oil system analyses
T4	Medium	6	2	231	Monitoring turbine states
T5	Medium	18	0	282	Interduct thermocouple analyses
T6	Medium	16	2	287	Igniter failure detection
T7	High	17	3	311	Bearing carbonisation
T8	High	19	2	335	Combustion chamber dynamics
T9	High	15	4	376	Gearbox Unit Shutdown
T10	High	12	8	401	Surge detection

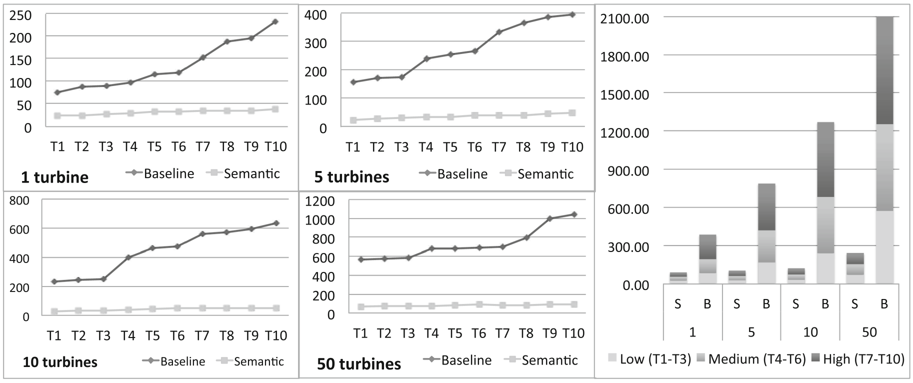


Fig. 5. Results of the user study. Left four figures: the average time in second that the users took to express the tasks T1–T10 for 1, 5, 10, and 50 turbines using existing Siemens rule language (Baseline or B) and our semantic rule language *sigRL* (Semantic or S). Right figure: the total time in seconds the user took to express the tasks grouped according to their complexity.

semantic case the time only slightly increases when moving from simple (T1–T3) to complex (T7–T10) tasks, while in the baseline case it increases significantly: from 2 to 4 times. The reason is that in the baseline case the number of sensor tags makes a significant impact on the authoring time: each of this tags has to be found in the database and included in the rule, while in the semantic case the number of tags does not make any impact since all relevant tags can be specified using queries. The number of event messages and the structure of rules affects both the baseline and the semantic case, and this is the reason why the authoring time grows in the semantic case when going from rules with low to high complexity.

Now consider how the authoring time changes for a given tasks when moving from 1 to 50 turbines. In the baseline case, moving to a higher number of turbines requires to duplicate and modify the rules by first slightly modifying the rule structure (to adapt the rules to turbine variations) and then replacing concrete sensors tags, threshold values, etc. In the semantic case, moving to a higher number of turbines requires only to modify the rule structure. As the result, one can see that in the semantic case all four semantic plots are very similar: the one for 50 turbines is only about twice higher than for 1 turbine. Indeed, to adapt the semantic diagnostic task T4 from 1 to 50 turbines the participants in average spent 50 s, while formulating the original task for 1 turbine took them about 30 s.

Finally, let us consider how the total time for all 10 tasks changes when moving from 1 to 50 turbines. This information is in Fig. 5 (right). One can see that in the baseline case the time goes from 400 to 2.100 s, while in the semantic case it goes from 90 to 240. Thus, for 10 tasks the semantic approach allows to save about 1.900 s and it is more than 4 times faster than the baseline approach.

Performance Evaluation. In this experiment, we evaluate how well our SQL translation approach scales. For this we prepared 5 diagnostic task, corresponding data, and verified firing of messages using a standard relational database engine PostgreSQL. We conducted experiments on an HP Proliant server with 2 Intel Xeon X5690 Processors (each with 12 logical cores at 3.47 GHz), 106 GB of RAM. We now first describe the diagnostic tasks and the data, and then report the evaluation results.

In Fig. 6 we present four of our 5 diagnostic tasks, and the task D_2 is our running example. Note that D_1–D_4 are independent from each other, while D_5 combines complex signals defined in the other four tasks. This is a good example of modularity of *sigRL*. On the data side, we took measurements from 2 sensors over 6 days as well as the relevant information about the turbines where the sensors were installed. Then, we scaled the original data to 2000 sensors; our scaling respect the structure of the original data. The largest raw data for 2000 sensors took 5.1 GB on disk in a PostgreSQL database engine.

During the experiments our system did two steps: translation of semantic diagnostic programs into SQL code and then execution of this SQL. During the first step our system generated SQL code that ranging from 109 to 568 lines depending on the diagnostic task and the code is of a relatively complex structure, e.g., for each diagnostic task the corresponding SQL contains at least 10 joins. The results of the second step are presented in Fig. 7. We observe that

Diagnostics Task D_1**:** "Is there a ramp change after 6 min in the turbine T100?":

$$\begin{aligned}\mathsf{SlowRotor} &= \min \mathsf{RotorSensor} : \mathsf{value}(<, \mathit{slowSpeed}) : \mathsf{duration}(>, 30s).\\ \mathsf{FastRotor} &= \max \mathsf{RotorSensor} : \mathsf{value}(>, \mathit{fastSpeed}) : \mathsf{duration}(>, 30s).\\ \mathsf{RampChange} &= \mathsf{FastRotor} : \mathit{after}[6m]\ \mathsf{SlowRotor}.\\ \mathsf{message}(\textit{"Ramp change"}) &= \mathsf{RampChange}.\end{aligned}$$

Diagnostic Task D_3**:** "Does the turbine T100 reach purging and ignition speed for 30 sec?":

$$\begin{aligned}\mathsf{Ignition} &= \mathit{avg}\ \mathsf{RotorSensor} : \mathsf{value}(<, \mathit{ignitionSpeed}).\\ \mathsf{PurgeAndIgnition} &= \mathsf{PurgingStart} : \mathsf{duration}(>, 30s) :\\ &\quad \mathit{after}[2m]\ \mathsf{Ignition} : \mathsf{duration}(>, 30s).\\ \mathsf{message}(\textit{"Purging and Ignition"}) &= \mathsf{PurgeAndIgnition}.\end{aligned}$$

Diagnostics Task D_4**:** "Does the turbine T100 go from ignition to stand still within 1min and then stand still for 30 sec?":

$$\begin{aligned}\mathsf{StandStill} &= \mathit{avg}\ \mathsf{RotorSensor} : \mathsf{value}(<, \mathit{standStillSpeed}).\\ \mathsf{IgnitionToStand} &= \mathsf{Ignition} : \mathsf{duration}(>, 1m) :\\ &\quad \mathit{after}[1.5m]\ \mathsf{StandStill} : \mathsf{duration}(>, 30s).\\ \mathsf{message}(\textit{"Ignition to Stand"}) &= \mathsf{IgnitionToStand}.\end{aligned}$$

Diagnostics Task D_5**:** "Is the turbine T100 ready to start?":

$$\begin{aligned}\mathsf{message}(\textit{"Ready to Start"}) &= \mathsf{RampChange} :\ \mathit{after}[5m]\ \mathsf{PurgingOver} :\\ &\quad \mathit{after}[11m]\ \mathsf{PurgingAndIgnition} :\\ &\quad \mathit{after}[15s]\ \mathsf{IgnitionToStand}.\end{aligned}$$

Fig. 6. Signal processing rules that we used for performance evaluation.

query evaluation scales well. Specifically, the running time grows sublinearly with respect to the number of sensors. The most challenging query D_5 can be answered in 2 min over 2000 sensors.

5 Related Work

The authors in [19] introduce temporal streaming language STARQL that extends SPARQL with aim to facilitate data analysis directly in queries. This and other similar semantic streaming languages, e.g., SPARQL$_{stream}$ [9], are different from our work, since we propose *(i)* a rule diagnostic language and *(ii)* focus on temporal properties of signals which are not naturally representable in those languages.

A preliminary idea on how to use semantic technologies in abstracting details of machines was presented in [21,22] where the authors use KBs to abstract away details of particular turbines in Siemens. Data about turbines is retrieved using OBDA and send to further analytical analysis (e.g., using KNIME system (www.knime.com)). This line of work does aims at using off-the-shelf analytical software instead of diagnostic rules.

Recent efforts have been made to extend ontologies with analytical and temporal concepts. Authors in [1,2] allow for temporal operators in queries and ontologies. Still, such approach use temporal logics (e.g., LTL) which in not adequate for our case since sensor data are organized based on intervals, e.g. [0s,10s].

Work in [16,19] introduces analytical operations directly into ontological rules in such a way that OBDA scenario is preserved. This line of work, we use an inspiration on how to define analytical functions on concepts, e.g. avg C, in OBDA setting. However, the authors do not consider temporal dimension of the rules.

Finally, our work is related to a well-studied Metric Temporal Logic [20]. One can show that *sigRL* is a non-trivial extension of the non-recursive Datalog language $\text{Datalog}_{\text{nr}}$MTL that has been introduced in [4]. Our rewriting techniques from *sigRL* rules into SQL follow similar principles as the ones for $\text{Datalog}_{\text{nr}}$MTL.

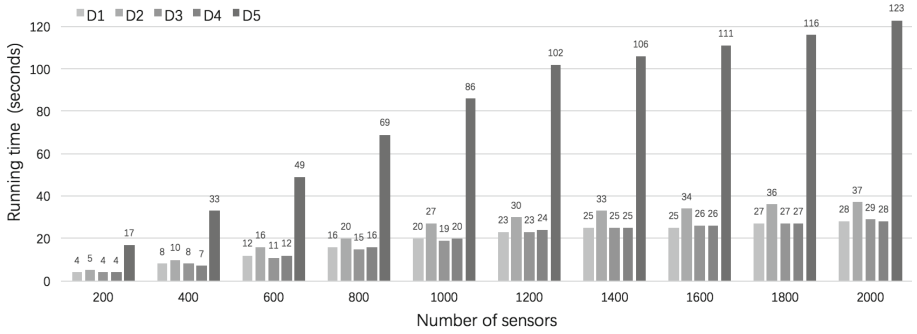


Fig. 7. Performance evaluation results for the Siemens use case.

6 Lessons Learned and Future Work

In this paper we showcase an application of semantic technologies for diagnostics of power generating turbines. We focused on the advantages and feasibility of the ontology-based solution for diagnostic rule formulation and execution. To this end we studied and described a Siemens diagnostic use-case. Based on the insights gained, we reported limitations of existing Siemens and ontology based solutions to turbine diagnostics. In order to address the limitations we proposed a signal processing rule language *sigRL*, studied its formal properties, implemented, and integrated it in an ontology-based system which we deployed at Siemens.

The main lesson learned is the effectiveness of our semantic rule language in dealing with the complexity of the rules and the number of turbines and sensors for rule deployment. The evaluation shows up to 66% of engineers time saving when employing ontologies. Thus, our semantic solution allows diagnostic engineers to focus more on analyses the diagnostic output rather than on data understanding and gathering that they have to do nowadays for authoring data-driven diagnostic rules. Following these experiments, we are in the process of setting up a deployment of our system into the Siemens remote diagnostic system to further evaluate the usability and impact. Second important learned lesson is that execution of semantic rules is efficient and scales well to thousands of sensors which corresponds to real-world complex diagnostic tasks.

Finally, note that our system is not included in the production processes. There are several important steps that we have to do before it can be included. First, it has to become much more mature and improve from the university-driven prototype to a stable system. Second, we have to develop an infrastructure for rule management, in particular, for rule maintenance and reuse—some of this work has already be done and one can our preliminary ideas in [18,23]. Third, we need more optimisations and evaluations that will include a performance comparison of our solution with the Siemens solutions that are based on the Siemens data-driven rule language. Moreover, we need techniques for abstracting (at lease some of) the existing 200k SPRs from the data driven rules into a much smaller number of sigRL. All of these is our future work.

Acknowledgements. This research is supported: EPSRC projects MaSI^3, DBOnto, ED^3; and the Free University of Bozen-Bolzano project QUEST.

References

1. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: The complexity of clausal fragments of LTL. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 35–52. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45221-5_3
2. Artale, A., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Temporal description logic for ontology-based data access. IJCAI **2013**, 711–717 (2013)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York (2003)
4. Brandt, S., Kalaycı, E.G., Kontchakov, R., Ryzhikov, V., Xiao, G., Zakharyaschev, M.: Ontology-based data access with a horn fragment of metric temporal logic. In: AAAI (2017)
5. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: answering SPARQL queries over relational databases. Semant. Web **8**(3), 471–487 (2017)
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semant. Web **2**(1), 43–53 (2011)

7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: the DL-Lite family. JAR **39**(3), 385–429 (2007)
8. Charron, B., Hirate, Y., Purcell, D., Rezk, M.: Extracting semantic information for e-Commerce. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) ISWC 2016. LNCS, vol. 9982, pp. 273–290. Springer, Cham (2016). doi:10.1007/978-3-319-46547-0_27
9. Corcho, O., Calbimonte, J.-P., Jeung, H., Aberer, K.: Enabling query technologies for the semantic sensor web. Int. J. Semant. Web Inf. Syst. **8**(1), 43–63 (2012)
10. Horrocks, I., Giese, M., Kharlamov, E., Waaler, A.: Using semantic technology to tame the data variety challenge. IEEE Internet Comput. **20**(6), 62–66 (2016)
11. Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I., Pinkel, C., Skjæveland, M.G., Thorstensen, E., Mora, J.: BootOX: practical mapping of RDBs to OWL 2. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 113–132. Springer, Cham (2015). doi:10.1007/978-3-319-25010-6_7
12. Kharlamov, E., Brandt, S., Giese, M., Jiménez-Ruiz, E., Kotidis, Y., Lamparter, S., Mailis, T., Neuenstadt, C., Özçep, Ö.L., Pinkel, C., Soylu, A., Svingos, C., Zheleznyakov, D., Horrocks, I., Ioannidis, Y.E., Möller, R., Waaler, A.: Enabling semantic access to static and streaming distributed data with optique: demo. In: DEBS (2016)
13. Kharlamov, E., Brandt, S., Jiménez-Ruiz, E., Kotidis, Y., Lamparter, S., Mailis, T., Neuenstadt, C., Özçep, Ö.L., Pinkel, C., Svingos, C., Zheleznyakov, D., Horrocks, I., Ioannidis, Y.E., Möller, R.: Ontology-based integration of streaming and static relational data with optique. In: SIGMOD (2016)
14. Kharlamov, E., et al.: Capturing industrial information models with ontologies and constraints. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) ISWC 2016. LNCS, vol. 9982, pp. 325–343. Springer, Cham (2016). doi:10.1007/978-3-319-46547-0_30
15. Kharlamov, E., et al.: Ontology based access to exploration data at statoil. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 93–112. Springer, Cham (2015). doi:10.1007/978-3-319-25010-6_6
16. Kharlamov, E., et al.: Optique: towards OBDA systems for industry. In: Cimiano, P., Fernández, M., Lopez, V., Schlobach, S., Völker, J. (eds.) ESWC 2013. LNCS, vol. 7955, pp. 125–140. Springer, Heidelberg (2013). doi:10.1007/978-3-642-41242-4_11
17. Kharlamov, E., et al.: Towards analytics aware ontology based access to static and streaming data. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) ISWC 2016. LNCS, vol. 9982, pp. 344–362. Springer, Cham (2016). doi:10.1007/978-3-319-46547-0_31
18. Kharlamov, E., Savković, O., Xiao, G., Mehdi, G., Penaloza, R., Roshchin, M., Horrocks, I.: Semantic rules for machine diagnostics: execution and management. In: CIKM (2017)
19. Kharlamov, E., et al.: How semantic technologies can enhance data access at siemens energy. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8796, pp. 601–619. Springer, Cham (2014). doi:10.1007/978-3-319-11964-9_38
20. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. **2**(4), 255–299 (1990)
21. Mehdi, G., Brandt, S., Roshchin, M., Runkler, T.A.: Semantic framework for industrial analytics and diagnostics. In: IJCAI, pp. 4016–4017 (2016)

22. Mehdi, G., Brandt, S., Roshchin, M., Runkler, T.A.: Towards semantic reasoning in knowledge management systems. In: AI for Knowledge Management Workshop at IJCAI (2016)
23. Mehdi, G., Kharlamov, E., Savković, O., Xiao, G., Kalaycı, E.G., Brandt, S., Horrocks, I., Roshchin, M., Runkler, T.: SemDia: semantic rule-based equipment diagnostics tool. In: CIKM (demo) (2017)
24. Mitchell, J.S.: An Introduction to Machinery Analysis and Monitoring. Pennwell Books, Tulsa (1993)
25. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semant. **10**, 133–173 (2008)
26. Randall, R.B.: Vibration-Based Condition Monitoring: Industrial, Aerospace and Automotive Applications. Wiley, Hoboken (2011)
27. Rao, B.K.N.: Handbook of Condition Monitoring. Elsevier, Amsterdam (1996)
28. Savković, O., Calvanese, D.: Introducing datatypes in DL-Lite. In: ECAI (2012)
29. Vachtsevanos, G., Lewis, F.L., Roemer, M., Hess, A., Wu, B.: Intelligent Fault Diagnosis and Prognosis for Engineering Systems. Wiley, Hoboken (2006)