Diagnostics of Power Generating Equipment with Semantic Rules (Extended Version)

Gulnar Mehdi
Siemens CT, TU Munich, Germany

Evgeny Kharlamov
University of Oxford, UK

Ognjen Savković
Free Univ. of Bozen-Bolzano, Italy

Guohui Xiao
Free Univ. of Bozen-Bolzano, Italy

Sebastian Brandt
Siemens CT, Germany

Ian Horrocks
University of Oxford, UK

Mikhail Roshchin
Siemens CT, Germany

Thomas Runkler
Siemens CT, TU Munich, Germany

ABSTRACT

Rule-based diagnostics of power generating equipment is an important task in industry. Diagnostics typically requires rules that process sensor signals, determine patterns in the signals, and produce notification messages when undesirable patterns are found. In Siemens such rules are data-dependent in the sense that they rely on specific characteristic of individual equipment and thus a complex diagnostic task may require a rule set with up to hundreds of such rules. Authoring and maintaining such rule sets is a challenging task that requires automation. In this demo 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 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 present our diagnostic system *SemDig*. The attendees will be able to write diagnostic programs in *SemDig* using *sigRL* over 50 Siemens turbines. We also present how such programs can be automatically verified for redundancy and inconsistency. Moreover, the attendees will see the provenance service that *SemDig* provides to trace the origin of diagnostic results.

ACM Reference format:
Gulnar Mehdi, Evgeny Kharlamov, Ognjen Savković, Guohui Xiao, Sebastian Brandt, Ian Horrocks, Mikhail Roshchin, and Thomas Runkler. 2017. Diagnostics of Power Generating Equipment with Semantic Rules (Extended Version) . In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Diagnostic systems play an important role in industry since they help to maximise equipment's up-time and minimise its maintenance and operating costs [10]. 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 [5], or checking vibration instrumentation [7], performance degradation [8], 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 discuss the challenges in mode details and then present our solution to address them.

Challenges with Authoring SPRs. The main challenge for authoring is that SPRs in most modern industrial 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 a typical diagnostic task to verify that the purging[2] is over in the main flame component of a given turbine 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.

Challenges with Management of SPRs. Development of a diagnostic rule-set 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

[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.

Conference'17, Washington, DC, USA

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 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* [1] to define a novel SPR language and on *reasoning* [3] 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 as the 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 such as HermiT [9]. We refer the reader to [1] for more details on ontologies and reasoning.

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 developed efficient algorithms to execute diagnostic programs in bottom-up fashion, verify redundancy and inconsistency in diagnostic programs, and to compute provenance that explains the reasons for diagnostic results. Moreover, we implemented our ideas in a system *SemDig* for diagnostics of power generating equipment that (1) allows to author diagnostic programs; (2) offers a tool-kit for management of diagnostic programs, that includes detection of redundancy and inconsistency, as well as provenance computation.

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.

Demo Overview. Demo attendees will be able to learn how to do diagnostics of Siemens turbines with *sigRL* diagnostic programs. To this end we prepared a deployment of our *SemDig* system on data from 50 Siemens power generating turbines, a diagnostic ontology, and a catalogue of 15 diagnostic tasks. The attendees will be able to load preconfigured diagnostic programs, deploy and execute them, author their own diagnostic programs, and try out our provenance computation and program verification services. See Section 3 for details on demo scenarios.

Due to space limit we put some formal definitions related to our *sigRL* language in the appendix.

2 OUR DIAGNOSTIC SOLUTION

We first introduce our diagnostic language *sigRL* and then describe our system *SemDig*.

2.1 *sigRL* Diagnostic Language

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. 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. We now define signal expressions that filter and manipulate basic signals and create new more complex signals. Intuitively, in our language we group signals in ontological concepts and signal expression are defined on the level of concepts. Then, a *signal processing expression* is recursively defined as follows:

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

where C, C_1, C_2 are concepts, $\circ \in \{+, -, \times, /\}$, $\alpha \in \mathbb{R}$, *agg* is one of min, max, avg, 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 {up, down}. Intuitively, 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 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 $f_{s'}(t) = \perp$. The formal meaning of signal processing expressions is defined in [4].

A *diagnostic program* is a tuple $\Pi = (\mathcal{S}, \mathcal{O}, \mathcal{M})$ where $\mathcal{S}$ is a set of basic signals, $\mathcal{O}$ is an ontology[3], $\mathcal{M}$ is a set of signal expressions. Each diagnostic program comes with a set of *message rules* that are defined on top of Boolean combinations of expressions:

$$
\begin{array}{rl}
& r = \mathit{msg}(m) \leftarrow D, \quad \mathit{where} \\
D := & C \mid \mathit{not}\ C \mid D_1\ \mathit{and}\ D_2
\end{array}
$$

We now illustrate our diagnostic programs on the following purging diagnostic task:

> *Verify that the purging 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, *(ii)* 15s after this, 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, i.e., by comparing the average speed of its speed sensors with purging thresholds that are specific for individual rotors. The purging diagnostic program can then consist of an ontology with one axiom:

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

[3] In order to guarantee efficiency of diagnostics with *sigRL*, we consider a tractable ontology language OWL 2 QL [3] that is standardised by W3C.

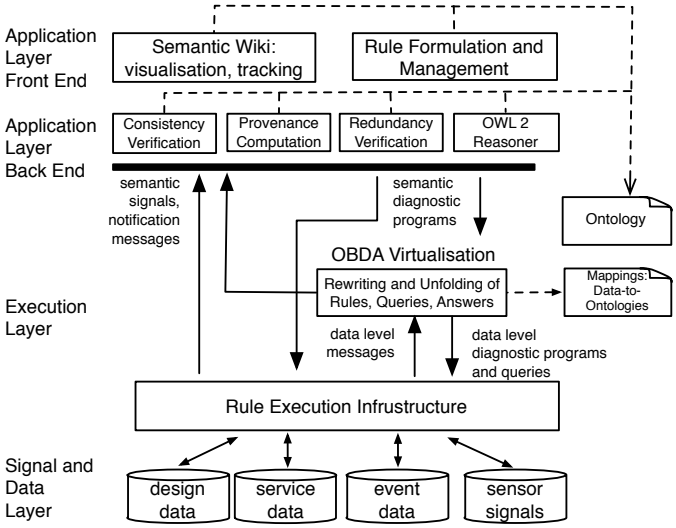

Figure 1: Architecture of *SemDig*.

two signal processing expressions and one message rule:

$$\text{PurgingStart} = \mathit{avg}\ \text{rotorStart} : \text{value}(>, \mathit{purgingSpeed}),$$
$$\text{PurgingStop} = \mathit{avg}\ \text{rotorStart} : \text{value}(<, \mathit{nonPurgingSpeed}),$$
$$\text{msg}(\text{“}\mathit{Purging\ over}\text{”}) = \text{FlameSensor} : \text{duration}(>, 10s) :$$
$$\mathit{after}[15s]\ \text{PurgingStart} : \mathit{after}[20s]\ \text{PurgingStop}$$

2.2 *SemDig* Diagnostic System

The main functionality of our semantic rule-based diagnostics system *SemDig* is to author and maintain *sigRL* diagnostic programs, to deploy them in turbines, to execute the programs, and to visualise the results of the execution. We now give details of our system by following its architecture in Figure 1 where the solid arrows indicate data flow and dashed—access to ontologies and mappings. There are three essential layers in the architecture: application, rule execution, and signal and data layers. Our system is mostly implemented in Java. We now discuss the system layer by layer.

Application Layer. On this layer, the system offers two front ends and several back end components. The first front end component allows engineers to author, store, load, and maintain diagnostic programs by formulating sets of SPRs as well as message rules in *sigRL* and sensor retrieving queries. Such formulation is guided by the domain ontology stored in the system. In Figure 2 (left) one can observe a screenshot of the diagnostic program editor which is embedded in the Siemens analytical tool-kit. Another front end component is the semantic Wiki that allows among other features to visualize signals and messages (triggered by programs), and to track deployment of programs in equipment. In Figure 2 (right) one can see visualisation of signals from two components of one turbine. The back end of the application layer consist of components that rely on HermiT [9] ontology reasoning and support consistency and redundancy verification as well as provenance computation. Diagnostic programs formulated in the application layer are converted into XML-based specifications and sent to the rule execution layer that returns back messages and signals. We rely on REST API to communicate between the application layer and the execution layer of our system and OWL API to deal with ontologies.

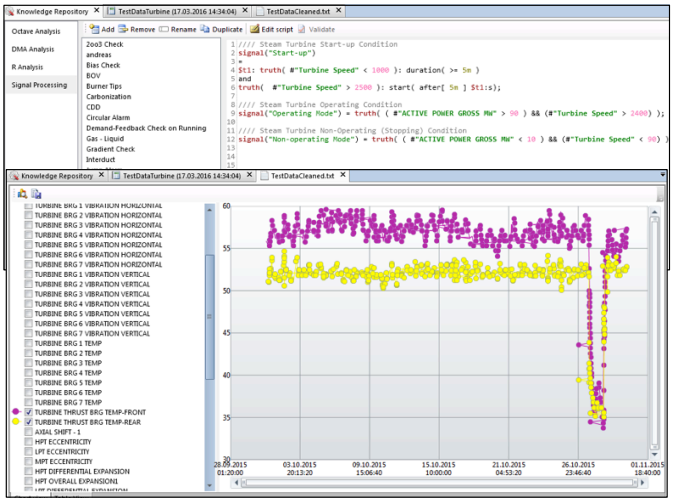

Figure 2: Screenshots: SPR editor (top), Diagnostic visualisation monitor (bottom)

Execution Layer. On this layer we support semantic signals that are either *native*, that is, represented in terms of the diagnostic ontology as RDF triple, or *virtual*, that is obtained through the Ontology Based Data Access (OBDA) [6] component of *SemDig*. This component allows to present signals stored in relational databases as if they were native semantic. This requires to connect the relational signals to an ontology via declarative mappings. For the OBDA layer we rely on the Ontop system [2] that takes care of transforming diagnostic programs written in *sigRL* into either SPRs written in the Siemens data-driven rule language 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. Moreover, the execution layer takes care of planning and executing rules and queries received either from the rule management or OBDA component. 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[4] 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.

Signal and Data Layer. On this 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. Currently *SemDig* support PostgresQL, Teradata, as well as Sparksee.

3 DEMONSTRATION SCENARIOS

Demo Setup. For the demonstration purpose we prepared the following ingredients:

- *diagnostic tasks*: 15 tasks (see Table 1) were defined during brainstorming sessions with Siemens diagnostic engineers and R&D personnel from Siemens Corporate Technology;
- *anonymised Siemens data*: from 50 power generating gas turbines gathered for 2 years that contains sensor signals, equipment

[4] http://drools.jboss.org/drools-fusion.html

Task Nr	Complexity	Diagnostic Tasks
T1	Low	Variable guided vanes analyses
T2	Low	Multiple start attempts
T3	Low	Lube oil system analyses
T4	Medium	Monitoring turbine states
T5	Medium	Interduct thermocouple analyses
T6	Medium	Igniter failure detection
T7	High	Bearing carbonisation
T8	High	Combustion chamber dynamics
T9	High	Gearbox Unit Shutdown
T10	High	Surge detection
T11		Does the turbine T100 reach purging and ignition speed for 30 sec?
T12		Is the purging over in the main flame component of the turbine T1.
T13		Does the turbine T100 reach purging and ignition speed for 30 sec?
T14		Does the turbine T100 go from ignition to stand still within 1min and then stand still for 30 sec?
T15		Is the turbine T100 ready to start?

Table 1: Demo diagnostic tasks for Siemens gas turbines.

specifications and configurations and maintenance data; all the data was anonymised for the demo purpose;

- *Siemens diagnostic ontology*: the ontology was inspired by the Siemens Technical System Ontology (TSO), Semantic Sensor Network Ontology (SSN), and the international standards IEC 81346 and ISO/TS 16952-10; the main module of our ontology in partially depicted in Figure 3 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.
- *SemDig deployment in Siemens*: over both materialised and ontology mediated Siemens data; *SemDig* is deployed on Teradata for signals and MS SQL for other information; for rule processing *SemDig* uses Drools Fusion; for the OBDA setting we developed 376 R2RML mappings.

During the demo we can either remotely connect to our deployment at Siemens or to run demo on a laptop. We now explain demo scenarios in details.

Scenario 1: Preconfigured Diagnostics. We plan to present the demo starting from this scenario where the attendees will get acquainted with *sigRL* and the main functionalities of *SemDig*. For this purpose we prepared 10 tasks of different complexity, where complexity is de ned using the number of ontological terms and lines of code. These tasks can be found in Table 1, they are indicated as T1–T10. The attendees will start with the tasks of *low* complexity by loading the corresponding diagnostic programs in the *SemDig* rule editor, learning the basic signal processing operations of *sigRL*, and then deploying the programs over the turbines. Then, the attendees will study the diagnostic results on the visualisation monitors. From the visualisation monitor the attendees will be able to request the provenience for specific answers. After the tasks with *low* complexity the attendees will proceed to the tasks with *medium* and *high* complexity. We have intentionally introduced inconsistencies and redundancy in the tasks so that the attendees will be able to detect them using the tooling support of *SemDig*.

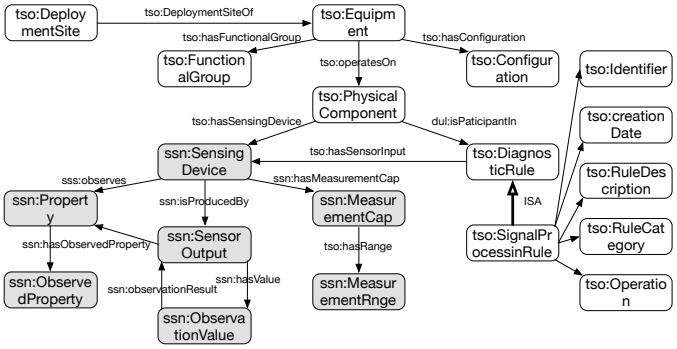

Figure 3: A fragment of the Siemens ontology that we developed to support turbine diagnostic SPRs.

Scenario 2: User Defined Diagnostics. In this scenario we assume that the demo attendees have already learned the basics of *sigRL* and now they are ready to author diagnostic programs in our editor depicted in Figure 2. For this purpose we prepared five tasks T11–T15 that can be found in Table 1. Note that for simplicity we phrased the tasks T11–T15 in such a way that the wording follows the actual grammer of *sigRL*. Moreover, most of semantic terms relevant for these tasks are explicitly mentioned in the tasks. Observe that the first four tasks are independant from each other, while the last task, T15, combines the other four. This will show the attendees the compositionality of our language *sigRL*. For illustration consider the following diagnostic program that corresponds to the task T13:

$$\text{Ignition} = \mathit{avg}\ \text{RotorSensor} : \text{value}(<, \mathit{ignitionSpeed}).$$
$$\text{PurgeAndIgnition} = \text{PurgingStart} : \text{duration}(>, 30s) : \mathit{after}[2m]\ \text{Ignition} : \text{duration}(>, 30s).$$
$$\text{msg}(\text{“Purging and Ignition”}) = \text{PurgeAndIgnition}.$$

And now the diagnostic program corresponding to T15:

$$\text{msg}(\text{“Ready to Start”}) = \text{RampChange} : \mathit{after}[5m]\ \text{PurgingOver} : \mathit{after}[11m]\ \text{PurgingAndIgnition} : \mathit{after}[15s]\ \text{IgnitionToStand}.$$

While the attendees formulate the five diagnostic tasks as diagnostic programs to check they will be asked to verify them for redundancy and consistency. After they finish formulation and verification, they can run the programs and check provenance.

REFERENCES

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (Eds.). 2003. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press.

[2] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. 2017. Ontop: Answering SPARQL queries over relational databases. *Semantic Web* 8 (2017).

[3] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *JAR* 39, 3 (2007).

[4] G. Mehdi, E. Kharlamov, O. Savković, G. Xiao, S. Brandt, I. Horrocks, M. Roshchin, and T. Runkler. 2017. Diagnostics of Power Generating Equipment with Semantic Rules (Extended Version). http://www.ghxiao.org/publications/2017-cikm-demo-tr.pdf. (2017).

[5] John S. Mitchell. 1993. *An introduction to machinery analysis and monitoring.* Pennwell Books.

[6] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. 2008. Linking Data to Ontologies. *J. Data Semantics* 10 (2008), 133–173.

[7] Robert Bond Randall. 2011. *Vibration-based condition monitoring: industrial, aerospace and automotive applications.* Wiley.

[8] B. K. N. Rao. 1996. *Handbook of condition monitoring.* Elsevier.
[9] Rob Shearer, Boris Motik, and Ian Horrocks. 2008. HermiT: A Highly-Efficient OWL Reasoner. In *OWLED*.
[10] George Vachtsevanos, Frank L. Lewis, Michael Roemer, Andrew Hess, and Biqing Wu. 2006. *Intelligent Fault Diagnosis and Prognosis for Engineering Systems.* Wiley.

A *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:

$$\text{ClassAssertion(RotorSensor 'S21R1T1')}.$$

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 $\mathbb{R} \cup \{\perp\}$, where $\perp$ 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:

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

where C, C_1, C_2 are concepts, $\circ \in \{+, -, \times, /\}$, $\alpha \in \mathbb{R}$, $agg \in \{\text{min}, \text{max}, \text{avg}, \text{sum}\}$, $\odot \in \{<, >, \leq, \geq\}$, $\mathit{filterAlign} \in \{\text{within}, \text{after}[t], \text{before}[t]\}$ where t is a period and *direction* is either {up, 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 $C_1 : \text{filterValue}(\odot, \alpha)$ 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 $f_{s'}(t) = \perp$. Due to the lack of space we illustrate other constructs of expressions with an example.

Example A.1. The following expression defines the start of a purging process from the running example:

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

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 $f_{s''}(t) > \mathit{purgingSpeed}$; otherwise $f_{s'}(t) = \perp$. □

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:

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

In order to guarantee efficiency of diagnostics with *sigRL*, we consider a tractable ontology language OWL 2 QL [3] 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:

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

Example A.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):

$$\text{PurgingStop} = \mathit{avg}\ \text{mainSpeedSensor} : \text{value}(<, \mathit{nonPurgSpeed}) \quad (3)$$

$$\text{msg}(\text{"}\mathit{Purging\ over}\text{"}) = \text{FlameSensor} : \text{duration}(>, 10s)\ \mathit{and}\ \text{PurgingStart} : \mathit{after}[20s]\ \text{PurgingStop}. \quad (4)$$

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 $\mathcal{S}$: $\mathcal{S}^{\mathcal{I}} = \{s_1^{\mathcal{I}}, \ldots, s_n^{\mathcal{I}}\}$ 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 $(o, f_\perp)$ where $f_\perp(t) = \{\perp\}$, 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.