TECHNISCHEN UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK

Semantically defined Analytics for Industrial Equipment Diagnostics

Gulnar Mehdi, M.Sc.

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation.

Vorsitzender:	Prof. Dr. Florian Matthes
Prüfer der Dissertation:	1. Hon.-Prof. Dr. Thomas Runkler 2. Prof. Dr. Helmut Kcrmar

Die Dissertation wurde am 20.01.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 22.05.2020 angenommen.

Dedicated to mom dad, my husband and angel baby 🦋

Abstract

In this age of digitalization, industries everywhere accumulate massive amount of data such that it has become the lifeblood of the global economy. This data may come from various heterogeneous systems, equipment, components, sensors, systems and applications in many varieties (diversity of sources), velocities (high rate of changes) and volumes (sheer data size).

Despite significant advances in the ability to collect, store, manage and filter data, the real value lies in the analytics. Raw data is meaningless, unless it is properly processed to actionable (business) insights. Those that know how to harness data effectively, have a decisive competitive advantage, through raising performance by making faster and smart decisions, improving short and long-term strategic planning, offering more user-centric products and services and fostering innovation. Two distinct paradigms in practice can be discerned within the field of analytics: semantic-driven (deductive) and data-driven (inductive).

The first emphasizes logic as a way of representing the domain knowledge encoded in rules or ontologies and are often carefully curated and maintained. However, these models are often highly complex, and require intensive knowledge processing capabilities. Data-driven analytics employ machine learning (ML) to directly learn a model from the data with minimal human intervention. However, these models are tuned to trained data and context, making it difficult to adapt.

Industries today that want to create value from data must master these paradigms in combination. However, there is great need in data analytics to seamlessly combine semantic-driven and data-driven processing techniques in an efficient and scalable architecture that allows extracting actionable insights from an extreme variety of data.

In this thesis, we address these needs by providing:

- A unified representation of domain-specific and analytical semantics,in form of ontology models called TechOnto Ontology Stack. It is highly expressive, platform-independent formalism to capture conceptual semantics of industrial systems such as technical system hierarchies, component partonomies etc and its analytical functional semantics.

- A new ontology language *Semantically defined Analytical Language* (SAL) on

top of the ontology model that extends existing DatalogMTL (a Horn fragment of Metric Temporal Logic) with analytical functions as first class citizens.

- A method to generate semantic workflows using our SAL language. It helps in authoring, reusing and maintaining complex analytical tasks and workflows in an abstract fashion.

- A multi-layer architecture that fuses knowledge- and data-driven analytics into a federated and distributed solution.

To our knowledge, the work in this thesis is one of the first works to introduce and investigate the use of the semantically defined analytics in an ontology-based data access setting for industrial analytical applications. The reason behind focusing our work and evaluation on industrial data is due to (i) the adoption of semantic technology by the industries in general, and (ii) the common need in literature and in practice to allow domain expertise to drive the data analytics on semantically interoperable sources, while still harnessing the power of analytics to enable real-time data insights. Given the evaluation results of three use-case studies, our approach surpass state-of-the-art approaches for most application scenarios.

Zusammenfassung

Im Zeitalter der Digitalisierung sammeln die Industrien überall massive Datenmengen, die zum Lebenselixier der Weltwirtschaft geworden sind. Diese Daten können aus verschiedenen heterogenen Systemen, Geräten, Komponenten, Sensoren, Systemen und Anwendungen in vielen Varianten (Vielfalt der Quellen), Geschwindigkeiten (hohe Änderungsrate) und Volumina (reine Datengröße) stammen.

Trotz erheblicher Fortschritte in der Fähigkeit, Daten zu sammeln, zu speichern, zu verwalten und zu filtern, liegt der eigentliche Wert in der Analytik. Rohdaten sind bedeutungslos, es sei denn, sie werden ordnungsgemäß zu verwertbaren (Geschäfts-)Erkenntnissen verarbeitet. Wer weiß, wie man Daten effektiv nutzt, hat einen entscheidenden Wettbewerbsvorteil, indem er die Leistung steigert, indem er schnellere und intelligentere Entscheidungen trifft, die kurz- und langfristige strategische Planung verbessert, mehr benutzerorientierte Produkte und Dienstleistungen anbietet und Innovationen fördert. In der Praxis lassen sich im Bereich der Analytik zwei unterschiedliche Paradigmen unterscheiden: semantisch (deduktiv) und Daten-getrieben (induktiv).

Die erste betont die Logik als eine Möglichkeit, das in Regeln oder Ontologien kodierte Domänen-wissen darzustellen, und wird oft sorgfältig kuratiert und gepflegt. Diese Modelle sind jedoch oft sehr komplex und erfordern eine intensive Wissensverarbeitung. Datengesteuerte Analysen verwenden maschinelles Lernen (ML), um mit minimalem menschlichen Eingriff direkt ein Modell aus den Daten zu lernen. Diese Modelle sind jedoch auf trainierte Daten und Kontext abgestimmt, was die Anpassung erschwert.

Branchen, die heute Wert aus Daten schaffen wollen, müssen diese Paradigmen in Kombination meistern. Es besteht jedoch ein großer Bedarf in der Datenanalytik, semantisch und datengesteuerte Verarbeitungstechniken nahtlos in einer effizienten und skalierbaren Architektur zu kombinieren, die es ermöglicht, aus einer extremen Datenvielfalt verwertbare Erkenntnisse zu gewinnen.

In dieser Arbeit, die wir auf diese Bedürfnisse durch die Bereitstellung:

- Eine einheitliche Darstellung der Domänen-spezifischen und analytischen Semantik in Form von Ontologie modellen, genannt TechOnto Ontology Stack. Es ist ein hoch-expressiver, plattformunabhängiger Formalismus, die konzep-

tionelle Semantik industrieller Systeme wie technischer Systemhierarchien, Komponentenparonomien usw. und deren analytische funktionale Semantik zu erfassen.

- Eine neue Ontologie-Sprache *Semantically defined Analytical Language* (SAL) auf Basis des Ontologie-Modells das bestehende DatalogMTL (ein Horn fragment der metrischen temporären Logik) um analytische Funktionen als erstklassige Bürger erweitert.

- Eine Methode zur Erzeugung semantischer Workflows mit unserer SAL-Sprache. Es hilft bei der Erstellung, Wiederverwendung und Wartung komplexer analytischer Aufgaben und Workflows auf abstrakte Weise.

- Eine mehrschichtige Architektur, die wissens- und datengesteuerte Analysen zu einer föderierten und verteilten Lösung verschmilzt.

Nach unserem Wissen, die Arbeit in dieser Arbeit ist eines der ersten Werke zur Einführung und Untersuchung der Verwendung der semantisch definierten Analytik in einer Ontologie-basierten Datenzugriff Einstellung für industrielle analytische Anwendungen. Der Grund für die Fokussierung unserer Arbeit und Evaluierung auf industrielle Daten ist auf (i) die Übernahme semantischer Technologien durch die Industrie im Allgemeinen und (ii) den gemeinsamen Bedarf in der Literatur und in der Praxis zurückzuführen, der es der Fachkompetenz ermöglicht, die Datenanalyse auf semantisch interoperablen Quellen voranzutreiben, und nutzen gleichzeitig die Leistungsfähigkeit der Analytik, um Echtzeit-Dateneinblicke zu ermöglichen. Aufgrund der Evaluierungsergebnisse von drei Anwendungsfällen übertrifft unser Ansatz für die meisten Anwendungsszenarien modernste Ansätze.

Acknowledgement

Over the past four and half years, my PhD career has been an exciting and an eventful journey. It has influenced greatly and positively in shaping my professional, academic and personal life. I was fortunate enough to collaborate with the renowned researchers in the field of semantics as well as professionals from the industry who guided me throughout this journey.

My deepest gratitude goes to my supervisor, Prof. Dr. Thomas Runkler. His expert guidance and encouragement especially during the difficult times have been a driving force for me. All the success and achievements e.g. Best Paper Award that I achieved during my PhD studies are in regards to his consistent motivation. I am also very thankful to him for introducing and helping me understand the art of scientific research, critical analysis, strategic decision making and peculiarities of academic writing. I am very grateful and fortunate enough to have had Prof. Dr. Runkler as my PhD supervisor.

My sincere thanks also go to my team at Siemens AG, especially Dr. Steffen Lamparter, Dr. Mikhail Roshchin and Dr. Sebastian Brandt, who did make themselves available for advising and supporting me technically and professionally during the initial days of my PhD. They were generous enough to spend countless hours in planning and reviewing my conference paper submissions that ultimately help me in improving the quality of my research.

I am also grateful to my use-case partners from Siemens business namely Dr. Davood Naderi, John Ayotte, Erik Aerlebaeck, Giuseppe Fabio Ceschini, Dr. Martin Klimmek, Micheal Taylor, Yvonne Quacken, Dr. Daniel Dagnelund, Dr. Francesco Ferroni, Dr. Sindhu Suresh and Dr. Quang Nguyen for sharing the data and their expert knowledge of multiple domains, its problem space and technical system technologies. Their interest and contributions have inspired my work on this dissertation and help me develop a working solution for real-world applications.

I would also like to mention my colleagues at Siemens Corporate Technology Dr. Thomas Hubauer, Dr. Siegmund Duell, Bernhard Lang, Dr. Alexey Fishkin and Dr. Martin Ringsquandl, for their support and positive spirited attitude, that made working at Siemens one of the best and most rewarding experiences for me. I am also grateful to the collaborating partners from University of Oxford namely Prof. Dr. Ian Horrocks, Prof. Dr. Evgeny Kharlamov, Dr. Ernesto Jimenez Ruiz, Dr. Dmitriy Zheleznyakov, from University of Bolzano, Prof. Dr. Diego Calvanese, Prof.

Dr. Guohui Xiao, Elem Guzel, Dr. Ognjen Savkovic and from University of Oslo Prof. Dr. Arild Waaler, Dr. Martin G. Skjaeveland and Prof. Dr. Martin Giese. Their expertise help me to improve the theorem-proving and reasoning functionality in my solution application.

I would like to extend my gratitude to the members of my dissertation committee, Prof. Helmut Kcrmar and Prof. Florian Matthes. Their comments and feedback were very insightful and beneficial in improving the overall quality of my dissertation. I would also like to thank the committee chairman, Dr. Wolfgang Worndl and Ms. Manuela Fischer for the administrative support.

The past few years have been roller-coaster ride and I am extremely grateful to my husband Dr. Danish Rafique for his immense support and encouragement. He has been like a rock, an inspiration and a sole motivator behind all my academic contributions.

Finally, special recognition and all my heart goes out to my parents, my brother, my sister, my niece and nephew and my extended family for understanding why it took me so long to visit them and in supporting me emotionally throughout my study period and my life in general.

Contents

1 Introduction

The emergence of Industry 4.0 has given opportunities to collect, process and analyze equipment data across the entire industry value chain that can significantly enhance maintenance, improve fault management, asset utilization, automation, performance and so forth. Large heterogeneous data sets from the industrial equipment such as trains, power generating turbines, smart grid components with nearly 16 Terabytes of data generating everyday [1], has become a gold mine for industries that can enable data-driven analytics such as for condition monitoring and diagnosis. However, integrating and aggregating data to implement analytics across different distributed data sets and different engineering domains is non-trivial and requires new efficient methods [2]. This phenomenon has served as an opportunity and motivated many research initiatives, where natural language processing [3], information retrieval and ontology-based data access (OBDA) are utilized to build models and approaches to extract and analyze relevant data.

Most existing approaches to data access and analytics [4, 5, 6, 7, 8, 9, 10, 11] have proved effective when a specific characteristic of individual asset such as equipment or sensor identifiers or a dataset query for a specific equipment is explicitly and unambiguously encoded in an tool-dependent analytical workflow. In such case, either analyst has to rely on IT specialist to develop a corresponding database query or he makes effort to understand each of the underlying data model. One typical task of an analytical workflow is to detect potential faults of a turbine equipment caused by, e.g., an undesirable pattern in pressure behaviour within various components of the turbine. Consider a (simplified) example of such a analytical task:

For a given turbine, list all pressure sensors that are operating reliably, i.e., they are operating within the average score of validation tests of at least 90%, and whose measurements are within the last 20 min were similar, i.e., Pearson correlated by at least 0.75, to measurements reported last month by a reference design sensor that had been functioning in a critical mode.

Such task requires to extract, aggregate, and correlate static data of a particular turbine, for which data is generally produced by up to 2,000 sensors that are installed in different parts of the turbine. In addition to this, there also exist historical operational data of the reference design sensor that may be stored in multiple data sources and have several versions or different product configurations. Executing and analyzing such a task currently requires to construct hundreds of queries, the majority of which are semantically the same (they ask about pressure), but syntactically differ

(they are over different schemata,formats). Implementing and executing so many datasource-specific queries and then integrating the computed answers in current state-of-the-art may take up to 80% of the overall diagnostic and analysis time that an engineer typically have to spend [12]. In such a scenario, the adoption of OBDA has proven to save a lot of time since only one domain-specific query can help to "hide" the technical details of how the data is stored, represented, and accessed in data sources, and to show only what this data is all about. However, any OBDA system faces two major challenges, i) lack of unified semantic representation of industrial or domain-specific system and its underlying data model, that provides a generic schema but allows for domain-specific building blocks to support knowledge sharing and information integration, and ii) inefficient and costly analytical operations in ontological queries, or in data queries specified in the mappings. In the case of ontological or domain-specific semantic queries, all relevant values from the source database must be retrieved prior to performing aggregations, arithmetic or any other analytical operation. Such operation is highly inefficient because it fails to exploit source capabilities (e.g., access to pre-computed averages). Data retrieval could also be a bottleneck because the retrieval could be slow and/or costly in case where these values are stored remotely. Moreover, such semantic queries also adds to the complexity of equipment-specific queries, and thus limits the benefits of the abstraction layer which attracts the user towards any OBDA system. Moreover, in the case of source queries, aggregation functions and comparison operators can be used in mapping queries. However, this is brittle and inflexible, as values such as 90% and 0.75, which are used to define "reliable sensor" and "similarity", cannot be specified in the ontological query, but must be "hard-wired" in a mapping language (e.g. R2RML), unless an appropriate extension to the query language or the ontology are developed.

In this thesis, we aim to address these issues by investigating, first, the role of unified modular ontology model and second, the use of semantically defined analytical OBDA that can support declarative representations of a industrial equipment together with basic analytical operations and using these to efficiently answer higher level queries for industrial equipment diagnosis. In particular, we research and evaluate several methods for access and analyzing two types of semantics for equipment diagnostics: *domain-specific semantics*, i.e. semantics extracted from background engineering ontologies and technical system specifications to capture equipment knowledge bases, and *analytics-aware semantics* i.e. semantics required to develop a analytical language and workflows.

Third, we use Siemens equipment data from three different domains i.e. power generation, mobility and smart grid as a representative case study of semantically defined analytical services in the experimental work conducted in this thesis. Specifically, we study the usability and applicability of both types of semantics in multiple analytical tasks for each domain. This mainly considers *equipment-level* analytical workflows, i.e., sequence of analytical task for detecting the faults of individual equipment (e.g., "Turbines", "Train", "Door" etc.), and *system-level* analytics, i.e., exchanges outcomes of individual analytical workflows to analyze e.g. the overall

performance of a given plant. Lastly, we investigate the use of our language to address the data-dependency challenges with authoring, reuse, and maintenance of analytical workflows.

We explain the motivation behind our thesis in the following subsections, and detail our research questions, thesis methodology, contributions and list of publications produced on our work.

1.1 Motivation

An industrial system is a network of intelligent industrial equipments such as trains and power generating turbines that collect and share large amounts of data. This data is either generated by various sensors deployed at the equipment or captures equipment specific meta-data such as configurations, history of use, design and manufacturing details. Exploitation of such large-scale data resources has the potential to revolutionize the competitiveness of the data-intensive industries where for example, intelligent diagnostics is critical to maximise equipment's up-time and minimise its maintenance and operating costs [13, 14]. With the advancements in Big Data technologies, significant progress has been made in addressing problems related to the volume and velocity of data, but still they often lag behind in meeting the *variety challenge*, which has emerged as the top data priority for mainstream companies [15]. As a result, the integration of information from multiple sources is often left to humans, making it difficult and time consuming for decision makers to obtain a coherent operational overview (Figure. 1.1).

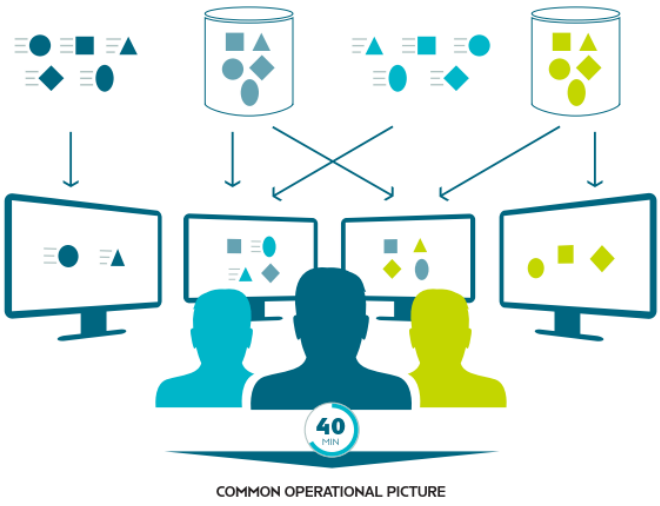

Figure 1.1: Existing approach: humans deal with information integration.

Nevertheless, where methods such as data virtualization, middleware integration, datawarehousing fall into prey of design and implementation challenges, Ontology-

based data access (OBDA)[16, 17] has emerged as a winner for many industrial use-cases (Figure. 1.2).

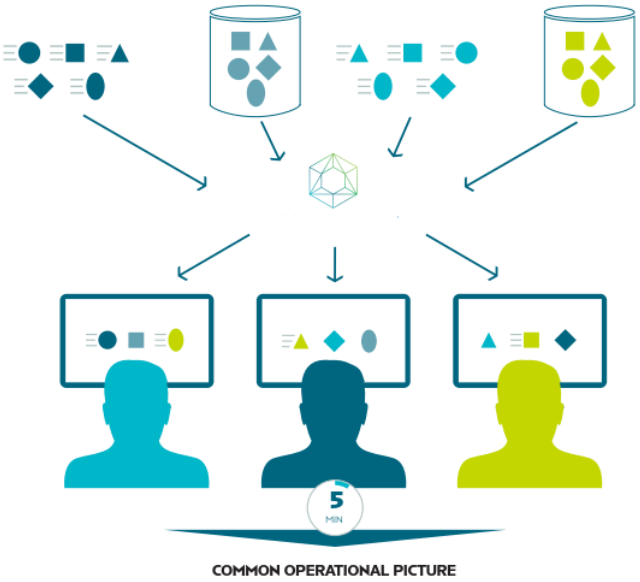

Figure 1.2: OBDA approach: decision makers provided with coherent operational overview

It provides an abstraction layer using an ontology that mediates between the data sources and data consumers. The ontology is a uniform conceptual schema that describes the domain of the underlying data independently of how and where the data is stored, and declarative mappings to specify how the ontology is related to the data by relating elements of the ontology to queries over data sources. The ontology and mappings are used to transform queries over ontologies, i.e., ontological queries, into data queries over data sources. As well as abstracting away from details of data storage and access, the ontology and mappings provide a declarative, modular and query independent specification of both the conceptual model and its relationship to the data sources; this simplifies development and maintenance and allows for easy integration with existing data management infrastructure. Large number of systems that at least partially implement OBDA have been recently developed; they include D2RQ [18], Mastro [19], morph-RDB [20], Ontop [21], OntoQF [22] and others [23]. Some of them are successfully used in various applications including cultural heritage [24], governmental organisations [25], IT benchmarking [26] and industry [1, 27, 28, 29]. Despite the success, OBDA systems come with some strengths and weaknesses when applied on complex industrial systems for analytical tasks including temporal concepts as will be explained in the following paragraphs.

1.1.1 OBDA for Industrial Analytics: Gaps and Challenges

The process industry accounts for more than 3% of European GDP, is under severe competitive pressure, and urgently needs to reduce costs and increase productivity

along the entire value chain. Heavy investment in digitalization and automation is bringing with it an explosion in the volume and velocity of available data. With an expected growth of of $7.3 billion in 2018, the market size of big data will break past the $40 billion mark in 2018 (as per Technical Report on Big Data Development (see https://bigdata-madesimple.com/4-critical-big-data-developments-to-prepare-for-in-2018/). This huge variety of data systems and data sources in use along the supply chain makes it a challenge to integrate and analyse this data. Narrowing down the gap between available data and execution capability (see Figure 1.3) has become a key objective for many industrial sectors, including condition monitoring [30, 31], predictive maintenance [32], smart manufacturing and IoT domains to provide decision makers with the information they need to optimise operating models and business processes.

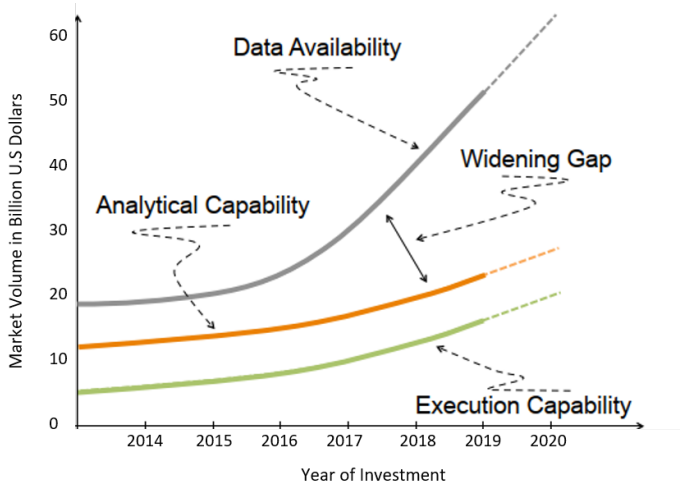

Figure 1.3: Gaps between available data, analytical capability, and execution capability [33].

Most of the industry sector is itself targeting this challenge with an increased focus on standards. This includes international, European, and national standards within the technical domains, as well as information and system integration standards such as OPC UA, Industry 4.0, ISA, IEC and ISO standards. The standards, however, are often imprecisely specified and inconsistently implemented, and their role in addressing the variety challenge has yet to be convincingly demonstrated.

With the emergence of semantic-based approaches, much effort has been made to develop upper-level and domain-specific ontologies that best describe the domain and its underlying data model. Overall, most existing ontologies adapt fairly well to some of the specific characteristics and use-cases of the industrial equipment, and therefore produce relatively higher performances than those which use the conventional generic model. Nevertheless, such models are still semantically weak, because do not represent the real semantics of equipments when analysing its behaviour or performance criteria that occur within them [34]. Semantics generally inclined towards exploring what an equipment, system or a component is supposed to mean in a given scenario. In the data mining and analytics world, representing and encoding

the semantics of the equipment will likely strengthen the understanding of the type of the data it produces and the behaviour or function that this data represents in the context it occurs.

1.1.2 From Equipment Semantics to Analytical-aware Semantics

In the previous paragraph we discussed that the existing approaches to equipment analysis (standard and semantic-based) usually address some of the pitfalls and limitations imposed by the industrial data. However, most of these solutions encounter similar problems as they are not tailored towards analytical tasks that are naturally based on data aggregation and correlation. A typical industrial scenario that requires both analytics and access to static and streaming data is diagnostics and monitoring of equipment. Traditional approaches of temporal streaming language STARQL extends SPARQL with aim to facilitate data analysis directly in queries. This and other similar semantic streaming languages, e.g., SPARQLstream [35], lack the support of rule-based structures and temporal representation of data signals. Recent efforts have been made to extend ontologies with analytical and temporal concepts. However, such approach use temporal logics (e.g., LTL, MTL) which in not adequate to capture time-series data which are often analyzed based on intervals, e.g. [0s; 10s]. Introduction of analytical operations directly into ontological rules (i.e. SWRL [36], datalogMTL [37]) in such a way that OBDA scenario is preserved, is the line of work, we use as inspiration. We aspire to investigate on how to define analytical functions on concepts, e.g. avg C, in an OBDA setting.

However, traditional methods lack the support of defining temporal dimensions to the rules. Thus considering both conceptual semantics of the equipment and analytical semantics is rather important when retrieving and analyzing the data. This is especially crucial to allow engineers to focus more on analysing the analytics output rather than on understanding the data and implementing an analytical workflows or combining various workflows for a specific use-case scenario.

The above limitations of different approaches either semantic or non-semantic has recently brought an immense interest in research and industrial use of semantics for data analysis (aka, semantically driven data analysis). However, semantic approaches (domain-specific and analytics-aware) are generally not equipped to handle complex industrial systems. First, they are restricted by the underlying semantic resources or domain ontologies, which is especially problematic when processing data across different systems using a unified model. Secondly, these approaches are tailored towards the use of conventional mathematical operations and temporal concepts that is trivial when analysing industrial data.

Our work in this dissertation addresses the problem of developing and managing analytical-aware ontology model and language for analytics. Our models and lan-

guage captures the semantics of industrial data to a large extent and features both domain-specific and analytical semantics in their analytical processes, aiming to capture the analytical tasks for equipments with regards to their semantics, and consequently improve the overall data analytics workflow. In the following sections, we present our research questions that we address in this thesis, we discuss our contributions, and we provide the outline of the thesis.

1.2 Research Questions and Contributions

The main research question investigated in this thesis is:

Could the semantics of industrial systems and analytical operations boost data analysis performance on industrial equipment?

Our main focus, as discussed in the previous section is to improve the performance of data analytics tasks and workflow by developing solutions that incorporates the core semantics of industrial systems and their analytical characteristics in an analytical workflow. Given the state of the problem discussed earlier (i.e., the type of the domain model and the type of semantics used), we have broken-down our main research question into following four sub-questions in order to improve data analysis for industrial data in a systematic fashion.

- **[RQ1]Can domain-specific and analytical-aware ontology models for industrial equipment enhance data analysis performance?**

  Adoption of wide range of standards for domain specification, communication and interface, life-cycle and system integration by the industries have not yet serve the purpose to access, integrate, exchange and/or analyse the relevant heterogeneous data. Key obstacles include:

  - overlapping and mutually inconsistent standards;
  - lack of precision in the specification of standards leaves them open to different interpretations;
  - legacy systems, some of which are up to 30 years old, implement standards in an ad-hoc way, if at all;
  - shortage of appropriately trained IT personnel means that standards may be poorly understood and inconsistently implemented;
  - the high cost of developing and maintaining the necessary standards; and

- the inability of Big Data technologies to exploit such standards in order to meet the variety challenge.

Semantic technologies and in particular OBDA approaches have been successful in providing a semantic declarative representation of the domain together with comprehensive and timely access to data, answering real user queries with response times in the range of seconds over TB-size federated databases with very complex structure; in contrast, existing systems require hours or even days to answer such queries, if they can answer them at all. However, the foundation of any OBDA system lies on its ability to capture the semantics of its domain and analytical characteristics. Thus, we propose a semantic representation of industrial equipment to improve OBDA and analytical workflows. Furthermore, we specify our concrete contributions for this research question as follows:

- Build a semantic ontology model, called TechOnto, that captures the conceptual and contextual semantics of industrial equipment and their diagnostic characteristics.

- Introduce several semantic-driven methods, based on TechOnto, for system and component levels fault analysis.

- Build and test a new ontology model manager to author, reuse and manage such models.

All these contributions are presented in Chapter 3.

- **[RQ2]Can an analytical-aware ontology language for analytical tasks enhance data analysis performance?**

Engineers create, use and deploy various diagnostic functions that include complex rule-sets and/or sophisticated analytical models to detect abnormalities of the equipment and may further combine these abnormalities with models of physical aspects of equipment for example, thermodynamics and energy efficacy. Based on the available resources and expertise of the engineer, he may use different analytical platforms, each of those have a specific conceptualization or schema for representing data and meta-data. This scenario leads to an extra coding-effort to achieve both the desired interoperability and a better provenance level. In addition to this, these functions are often data-dependent in the sense that specific characteristic of individual sensors and pieces of equipment are explicitly encoded in the models defined in a specific language.

To reduce the gap, data analytics vocabularies and ontologies have been proposed. Our contributions under this research question incorporates extraction

and inclusion of semantic concepts for analytical operations to enhance their performance. Our contributions are stated as follows:

- Propose a semantic driven ontology language, called SAL as a common language to enable data analysis across heterogeneous data sets, support interoperability among analytical tools and provide more automatized environment for obtaining the analytical results.

- Implement analytical operations that can filter, aggregate, combine, and compare data signals and is expressive enough and computationally efficient.

- Prove the proposed ontology language to be effective in an industrial setting and FO rewritable.

All these contributions are presented in Chapter 4.

- **[RQ3]Can semantically driven analytical workflows boost data analysis performance?**

  Engineers tend to produce a large number of analytical workflows using dedicated data-sets and models specified in a tool-dependent language. And more often these results are stored locally or shared via traditional interfaces. Such analysis are time-consuming and requires a certain level of expertise and resources. On the other hand, data and tool-specific dependences makes authoring, composition, reuse and maintenance of such workflows difficult and error-prone.

  To address these challenges, we propose building a new approach to extract the meta-data of these workflows and support authoring and composition of analytical workflows in a cost-effective way. Many existing approaches use syntactic structure of rules or pre-defined sets of templates in order to access data. Whereas, our solution provides flexibility and promotes reusability of workflows. We have also evaluated our proposed approach to determine its effectiveness in a system- and equipment-level fault analysis tasks. Contributions for this research question are:

  - Propose a novel approach that automatically extracts relevant data from the contextual semantic and generate corresponding analytical workflows for a given task.

  - Formulate and execute workflows using our proposed ontology language that combine relevant data and analytics together.

  - Perform quantitative and qualitative analysis on a test cases of our extracted analytics-aware semantics and show the potential of our approach

for finding the right models and composing analytical workflow in a user-friendly manner.

All these contributions are presented in Chapter 5.

- **[RQ4]Can a semantically defined analytical system boost data analysis performance?**

  Any data analytics application today requires sufficient amount of resources together with a rigorous collaboration of domain experts, engineers, IT specialist, statisticians/data miners and software developers. This large resource commitment makes it difficult to incorporate analytics as part of an overall business process and ultimately create a direct financial investment link. There is great need for a unified semantically defined data analytics solution that is able to combine the knowledge- and data-driven processing techniques in an efficient and scalable architecture and where experts can collaborate on their desired data analysis task rather than managing data challenges and software components.

  Contributions for this work in particular are:

  – Implementation of a scalable and efficient semantic framework to manage data analysis task, algorithms, their implementations and executions, as well as inputs (e.g., data) and outputs (e.g., models) they specify.

  – Conduct feature analysis of our semantic system and the state-of-the-art.

  All these contributions are presented in Chapter 6.

1.3 Thesis Methodology and Outline

Our thesis aims to present our contributions towards improving the performance of data analysis tasks and workflows by using both, the domain-specific and analytical-aware ontology models and language. To achieve the purpose, we developed a generic methodology that is adopted in the different phases of our work. Abstraction, incorporation and assessment are three building blocks of our methodology as depicted in Figure 1.4 and details are presented below:

1. Abstraction: design methods for capturing domain-specific and analytical-aware semantics for industrial equipment.

2. Incorporation: investigate state-of-the-art ontology models and languages for incorporating and using domain and analytical-driven semantics in data analysis tasks and workflows.

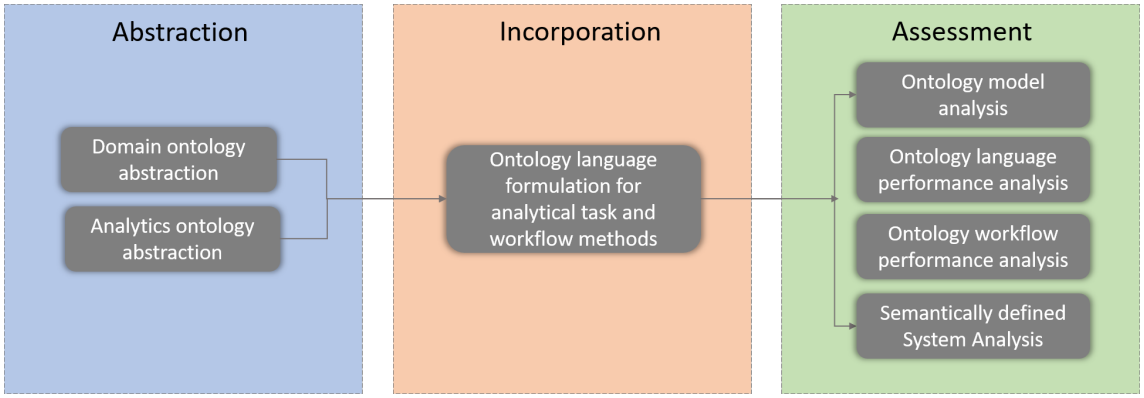

Figure 1.4: Our thesis methodology for abstracting, incorporating and assessing the use of analytical-aware semantics in industrial diagnostic scenarios.

3. Assessment: measure the performance of our proposed approach of modelling and use of analytical-driven semantic language in multiple analytical tasks on industrial data as well as analytical workflow composition.

Figure 1.5 presents an overview of our core contributions that are described in each chapter of this thesis. We have used these three pillars of the methodology as presented above in each of these chapters. Our work in this thesis comprises of the following chapters:

In Chapter 2 we presents a background knowledge of the semantic technologies and data analysis task. After that, we present details on the building blocks and existing work in the the area of OBDA and semantic-driven analytics. We also present the challenges and limitations of the current state-of-the-art models and languages.

In Chapter 3 we present results on our work of using the domain-specific and analytical ontology models for industrial equipment for improving the performance of data analysis methods for diagnostics. We also explore the use of ontology model manager to maintain such models. Our first research question is addressed in this chapter.

In Chapter 4 we present results on our work of using the analytical-aware ontology language to improve the performance of data analysis approaches, addressing our second research question of this dissertation.

In Chapter 5 we present our proposed approach to extract relevant information for data-access and generate workflows in an automated way which addresses the third research question. We also present our ontology language to support authoring and analysis of such semantic-driven workflows.

In Chapter 6 we present the resulting architecture, system implementation of our solution and its deployment at Siemens power generation business. This addresses the fourth research question of our thesis.

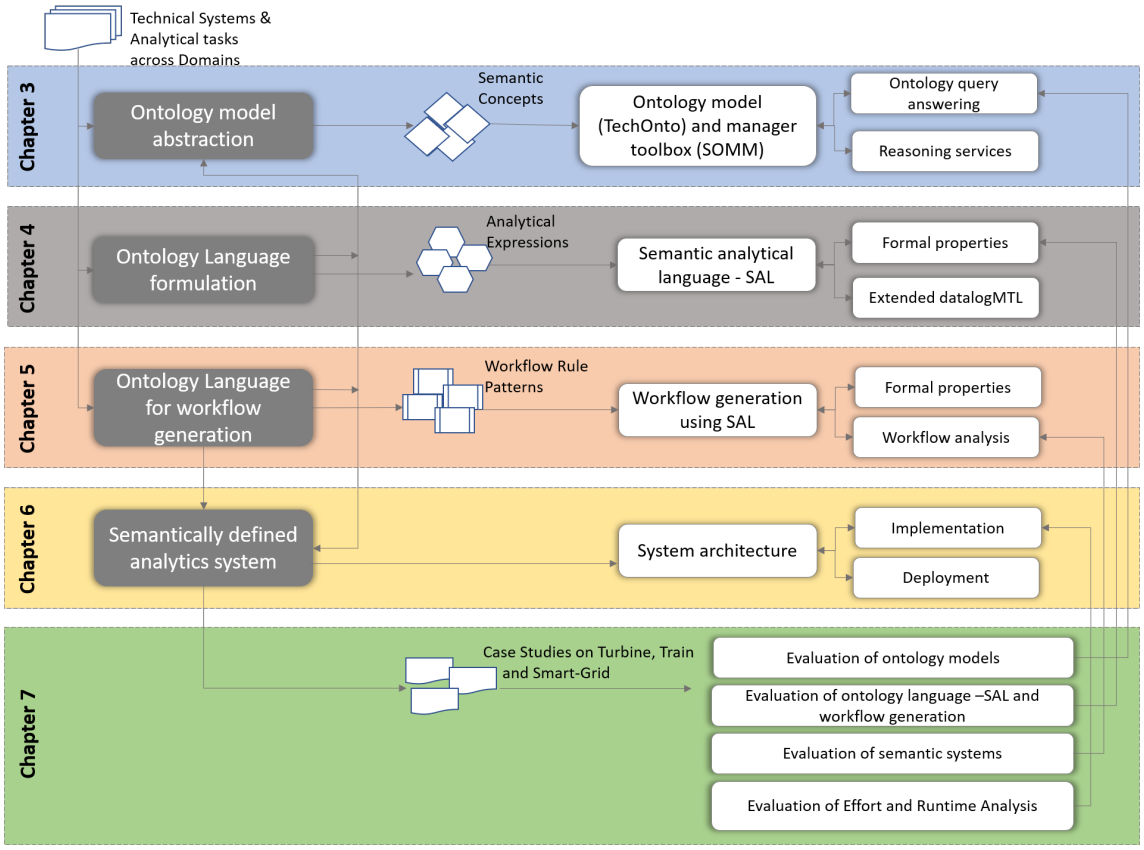

Figure 1.5: Overview of our work under each chapter along with our contributions. Arrows crossing two different chapters represent that results presented in one chapter is used in the other one.

In Chapter 7 we describe case studies and conduct evaluations on three different industrial use-cases and device a number of analytical task and workflows to proof the effectiveness of our approach.

In Chapter 8 we present our main conclusions, our contributions, research limitations and provide insights to our future work.

1.4 Publications

Within this thesis, we have achieved following publications (in international conferences and journals with peer-review process) and in patent applications:

[1] Gulnar Mehdi, Evgeny Kharlamov, Ognjen Savković, Guohui Xiao, Elem Güzel Kalaycı, Sebastian Brandt, Ian Horrocks, Mikhail Roshchin, and Thomas Runkler. "Semantic rule-based equipment diagnostics." In International Semantic Web Conference, pp. 314-333. Springer, Vienna, Austria, 2017. ***Best Paper Award***.

[2] Gulnar Mehdi, Evgeny Kharlamov, Ognjen Savković, Guohui Xiao, Elem Güzel Kalaycı, Sebastian Brandt, Ian Horrocks, Mikhail Roshchin, and Thomas Runkler. "SemDia: Semantic rule-based equipment diagnostics tool." In ACM Conference on Information and Knowledge Management, pp. 2507-2510. ACM, Pan Pacific Singapore, 2017.

[3] Gulnar Mehdi, Thomas Runkler, Mikhail Roshchin, Sindhu Suresh, and Nguyen Quang. "Ontology-based integration of performance related data and models: An application to industrial turbine analytics." In IEEE 15th International Conference on Industrial Informatics (INDIN), pp. 251-256. IEEE, Emden, Germany 2017.

[4] Gulnar Mehdi, Giuseppe Ceschini, Davood Naderi and Mikhail Roshchin. "A method and apparatus for performing a model-based failure analysis of a complex industrial system." US Patent Application 15/579,972 , Jun. 21 2018.

[5] Gulnar Mehdi, Evgeny Kharlamov, Ognjen Savković, Guohui Xiao, Elem Güzel Kalaycı, Sebastian Brandt, Ian Horrocks, Mikhail Roshchin, and Thomas Runkler. "Semantic rules for Siemens turbines." In International Semantic Web Conference, CEUR Workshop Demo and Poster Proceedings, Vienna, Austria, 2017.

[6] Gulnar Mehdi, Sebastian Brandt, Mikhail Roshchin, and Thomas Runkler. "Towards semantic reasoning in knowledge management systems." In IFIP International Workshop on Artificial Intelligence for Knowledge Management, pp. 132-146. Springer, New York, USA, 2016.

[7] Gulnar Mehdi, Sebastian Brandt, Mikhail Roshchin, and Thomas Runkler. "Semantic framework for industrial analytics and diagnostics." In International Joint Conferences on Artificial Intelligence, pp. 4016-4017. Springer, New York, USA 2016.

[8] Gulnar Mehdi, Giuseppe Ceschini, Davood Naderi and Mikhail Roshchin. "Model-based reasoning approach for automated failure analysis: An industrial gas turbine application." In Annual Conference of the Prognostics and Health Management Society, San Diego, California, USA 2015.

[9] Gulnar Mehdi, Giuseppe Ceschini, Davood Naderi and Mikhail Roshchin. "Model-based approach to automated calculation of key performance indicators for industrial turbines." In Annual Conference of the Prognostics and Health Management Society, San Diego, California, USA 2015.

[10] Gulnar Mehdi, Thomas Runkler, Mikhail Roshchin, Sindhu Suresh, and Nguyen Quang. "Semantic-aware analytics for smart grids." In IEEE Power and Energy Society's Transmission and Distribution Conference and Exposition, Dallas, TX, USA, 2018.

[11] Evgeny Kharlamov, Gulnar Mehdi, Ognjen Savković, Guohui Xiao, Steffen Lam-

parter, Ian Horrocks, and Arild Waaler. "Towards simplification of analytical workflows with semantics at Siemens." In IEEE International Conference on Big Data (Big Data), pp. 1951-1954. IEEE, Seattle, WA, USA, 2018.

[12] Ognjen Savković, Evgeny Kharlamov, Martin Ringsquandl, Guohui Xiao, Gulnar Mehdi, Elem Güzel Kalaycı, Werner Nutt and Ian Horrocks. "Semantic Diagnostics of Smart Factories." In Joint International Semantic Technology Conference, pp. 277-294. Springer, Awaji City, Hyogo, Japan, 2018.

[13] Evgeny Kharlamov, Gulnar Mehdi, Ognjen Savković, Guohui Xiao, Elem Güzel Kalaycı, and Mikhail Roshchin. "Semantically-enhanced rule-based diagnostics for industrial Internet of Things: The SDRL language and case study for Siemens trains and turbines." Journal of Web Semantics (2018).

[14] Evgeny Kharlamov, Ognjen Savković, Martin Ringsquandl, Guohui Xiao, Gulnar Mehdi, Elem Güzel Kalaycı, Werner Nutt, Mikhail Roshchin, Ian Horrocks, and Thomas Runkler. "Diagnostics of trains with semantic diagnostics rules." In International Conference on Inductive Logic Programming, pp. 54-71. Springer, Ferrara, Italy, 2018.

[15] Ognjen Savković, Evgeny Kharlamov, Guohui Xiao, Gulnar Mehdi, Elem Güzel Kalaycı, Werner Nutt, Mikhail Roshchin, and Ian Horrocks. "Theoretical characterization of signal diagnostic processing language." In Description Logic Workshop (DL 2018), pp. 1-11. Tempe, Arizona, USA, 2018.

[16] Evgeny Kharlamov, Ognjen Savković, Guohui Xiao, Rafael Penaloza, Gulnar Mehdi, Mikhail Roshchin, and Ian Horrocks. "Semantic rules for machine diagnostics: Execution and management." In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 2131-2134. ACM,Pan Pacific Singapore, 2017.

[17] Evgeny Kharlamov, Theofilos Mailis, Gulnar Mehdi, Christian Neuenstadt, Ozgür Ozcep, Mikhail Roshchin, Nina Solomakhina, Ahmet Soylu, Christoforos Svingos, Sebastian Brandt, Martin Giese, Yannis Ioannidis, Steffen Lamparter, Ralf Möller, Yannis Kotidis, Arild Waaler. "Semantic access to streaming and static data at Siemens." In Journal of Web Semantics: Science, Services and Agents on the World Wide Web, pp. 54-74, 2017.

[18] Evgeny Kharlamov, Bernardo Cuenca Grau, Ernesto Jimenez-Ruiz, Steffen Lamparter, Gulnar Mehdi, Martin Ringsquandl, Yavor Nenov, Stephan Grimm, Mikhail Roshchin, and Ian Horrocks. "Capturing industrial information models with ontologies and constraints." In International Semantic Web Conference, pp. 325-343. Springer, Kobe, Hyogo, Japan, 2016.

[19] Ian Horrocks, Bernardo Cuenca Grau, Ernesto Jimenez-Ruiz,Yavor Neno, Evgeny Kharlamov, Steffen Lamparter, Gulnar Mehdi, Martin Ringsquandlv and

Sebastian Brandt. "Somm: Industry oriented ontology management tool." In International Semantic Web Conference. Springer, Kobe, Hyogo, Japan, 2016.

2 Fundamentals of Ontology Models and Languages for Data Access and Analytics

In this chapter we present the foundations of Semantic Web Technologies together with formal definitions. We also explore the existing state-of-the-art ontology models, languages and their constructs. We introduce the building blocks of ontology based data access systems and its state-of-the-art together with available analytics aware systems.

2.1 Background

The world wide web (WWW) has provided an exponential increase of information and has revolutionized the production and use of data. Until now the WWW has been known and is accessible via search engines and browsers. However, this immense volume of web content is not understandable by any computer. The vision behind the Semantic Web (SW) is to make this content understandable and provide semantic meaning to enrich information for effective services.

With the advancement in Semantic Web Technology and knowledge based graph structures, all of the web content can be available in machine readable formats and therefore would be possible for automated processing. There are two foundational blocks to SW. Firstly, a formal ontology model which is mainly domain specific background information that is expressed, formalized across different partners. It provides semantic descriptions and a shared vocabulary for a particular domain and is expressed in form of object classes, predicate classes and their interdependencies. All these semantic descriptions that represent a certain background information is formalized in logical statements and axioms. Secondly, this web content is annotated by constraints that can be read and processed by machines.

There are number of SW applications using this semantic content, amongst which context and user-driven information retrieval is a prominent application. However, this information retrieval is highly users dependent, and needs integration of information from multiple sources that uses smart filters to gather user centric information. Therefore, such smart engines becomes more autonomous that captures and is aware

of semantic knowledge, as well as is capable of interpreting user request, knows where the content can be originated from and later represents the requested information in an appropriate user-friendly form. The advantage of semantic annotation features allows to obtain better search ranking and therefore the semantic matching between query and the content can be evaluated with high confidence. The second set of applications are Semantic web services. These services present texts, web content and multimedia data in a machine readable form. Such services are of great interest both for academia and industry. Well-defined SW standards are formulated to be understood by semantic search engines and web applications. Service requests and interfaces follow such SW standards to serve user-driven information retrieval. In the third set of SW applications comes the web of data. SW technologies provides infrastructure, standards and representational languages to integrate and exchange diverse set of informations. The domain of Biomedicine is a prominent candidate of SW application with almost 1000 databases publicly available today. One can formulate a common ontological vocabulary to integrate and publish this huge volume of data for querying and for analysis. The vision of WWW is to bring the knowledge of the world to our applications, readily available and accessible for processing and analysis. Lastly, in the fourth set of applications, SW technologies are required to support expert systems, modelling complex industrial domains and supporting advanced decision making process.

One of the key selling point of SW application is the support for reasoning capabilities. The reasoning capability relies on ontological background knowledge and the set of asserted statements to derive new set of knowledge. However, it has some limitations in practice. First, logical reasoning does not easily scale up to huge volumes of data which is required by many industrial applications. Projects like the EU FP 7 Scalable End-user Access to Big Data (see. http://www.optique-project.eu/) address this issue and provide implementations and infrastructure to access information on the fly supporting Ontology-based Data Access approaches. Secondly, until now logical reasoning does not easily support temporal and spatial information. However, with recent research contributions, such representation of temporal and spatial information on the SW and reasoning with temporal constraints on an OBDA system have been recently addressed. Third, logical reasoning completely relies on axiomatic prior knowledge. It does not explore patterns/ knowledge statements in the data that are not present as ontological background statements. This brings opportunities for new researchers to explore solutions of combining both analytics, learning as well as temporal constraints to access data and explore new frontiers of hidden information. The analysis of the potential of data analytics for the SW is the topic of this thesis. More and more information is made available in SW formats and machine learning and data mining are the basis for the analysis of the combined data sources.

2.1.1 Basics and History

There exist some common terminologies in SW community. For example, terms knowledge-base, semantic data representation or ontology are used interchangeably. Applications implementing SW technologies usually adopt some form of (semi-)formal abstract/meta representation of knowledge. Whereas, the traditional technologies allow the storage of raw data. The term ontology usually used in Artificial Intelligence (AI) is a formal abstraction of a given domain that includes a set of domain-specific concetps and their relationships. Formal ontologies are well equiped in supporting instance data with reasoning services that considers the use of background knowledge represented in form of formal logic axioms and constraints. The semantics of a formal logical representation refers to the fact that the meaning is unambiguous and making such information machine processable. Originally, the term ontology comes from the science of philosophy which refers to a particular system of categories which are supporting a specific vision of the world. Since the early 70s ontologies emerged as a prominent technology and was used in the field of Computers and AI. The technology was required to store knowledge in a machine readable format and to and process user queries. The charm of this technologiy was that it was able to process the inserted information as well as deduce additional new and unknown facts. Thus, the use in AI mainly reflects to an engineering artifact describing a specific domain at hand by defining its list of concepts together with their semantics including inference rules. An example for one of the first "semantic representations" is SIR (Semantic Information Retrieval) [38]. In SIR, the sentences represents entities and relationships among these entities. Traditionally semantic applications deal in domain taxonomies rather than arbitrary relations with specific semantic. Taxonomies are generally best understood in the form of trees. Such data structures are typical known as frames, termed by Marvin Minsky [39]. There exist a number of frame-based systems such as KL-ONE [40] that became the popular choice of knowledge representation like description logic (DL). Most of the knowledge representations DL is a subset of first-order logic. It highly supports reasoning and reasonable number of inference algorithms to deduce additional knowledge. Figure 2.1 lists terms popularly known to different research communities for related concepts of components of knowledge representations along with an informal description from the perspective of machine learning. This abbreviated format over simplifies and over generalizes most terms. However such simplification makes information access much easier and manageable in many contexts.

2.1.2 Definition of Formal Ontologies

In the field of AI, an ontology is considered as an engineering artifact, that comprises of a specific vocabulary which describes and represent a certain domain. In addition, the artifact also includes a set of explicit facts regarding the intended semantics of the world in the formal representation [41]. The term formal ontology is generally used to differentiate the applications of ontologies in the field of AI from the science

Term	Description
individual, *instance*, *fact*, *statement*, *ground(ed) atom*	real world evidence or observation; measurable or countable concrete event or entity
relation, *predicate*, *property*, *role*	a relation between individuals (relation instance) or concepts
resource, *individual*, object	instance of an entity class
domain, universe	set of possible entities that can be quantified
extension of a class	set of all instances of one class
ABox	set of observed instantiations (e. g. , of entity classes and relation classes)
concept, *entity (class)*, class, entity, type, category	a grouping of objects
interpretation, possible world	valid and consistent concrete assignment of instances to relations and concepts
vocabulary	set of terms naming concepts, individuals and relations
schema, terminological knowledge, concept level, type level, *TBox*	vocabulary without individuals
taxonomy	tree like hierarchy of classes; special case of ontologies where all relations have the same semantic, namely the *subsumption* relation.
model	In the context of formal semantics a model is an interpretation that satisfies a logical theory (evaluates an interpretation to be true). When working with probabilistic representations, e. g. , in machine learning, model is used to refer to the format how the data and the hypotheses are represented. Model is also commonly used in the context of software engineering to describe data representations or in databases where it defines database model or database schema which is the structure or format of a database.

Figure 2.1: Commonly used ontology terms.

of philosophy. However, the field of computer science specifies to the term ontologies as a knowledge representation of both non-formal or semi-formal knowledge bases. It is important to note what do we mean by formal representation. The widely accepted definition of formal is that the constructed ontology is expressed in a formal language, that contains a set of terminologies (entities with a certain vocabulary), a syntax (formal grammar) and formal semantics. These foundation elements makes the knowledge representation precise and unambiguous, allowing for an automated machine processing mechanism.

However, one can conclude that formal representation requires an ontology to be computationally expressive.

There have been many attempts to define the ontology term in many different context and domains. However, the most cited definition of ontologies in information systems is:

"An ontology is an explicit specification of a conceptualization" [42].

This definition encapsulates a generic and abstract conceptualization of the ontology artifact. It emphasizes on a construction of an abstract model that describes certain aspect of the world, represented in form of entities, properties and the relationships between them [43]. Studer et al made some efforts to provide a more intuitive explanation of this definition:

"An ontology is a formal, explicit specification of a shared conceptualization".

Here, conceptualization means an abstract world view. Explicit means that the model belongs to some aspect of the work and their intend is explicitly defined. Formal represents that the ontology must be computationally expressive, have formal semantics and should be machine-readable. Shared meaning the notion the vocabulary captured in the ontology is agreed amonst different parties, that is, it is not restrictive to any set of individuals, but mutually agreed by a group of experts. Guarino [41] provides another intuitive definition:

"An ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models."

In addition, Guarino [41] provides a formal definition of the terms used (cmp. to the informal definition in figure 2.1):

Formal vocabulary: $\mathcal{V}$ is an ontological vocabulary, i.e., a set of classes and properties of ontology language .

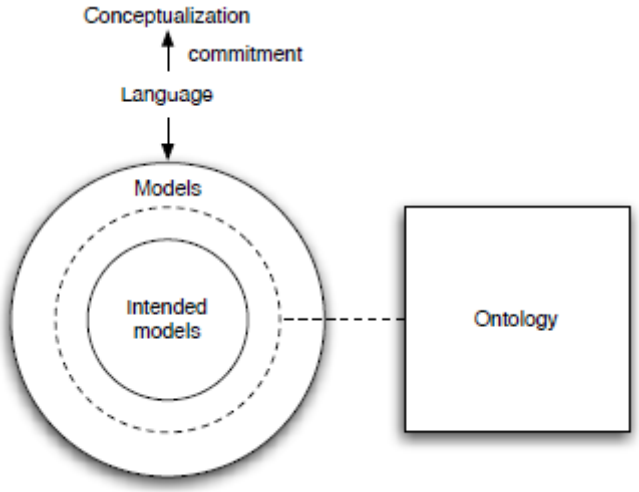

Figure 2.2: "The intended models of a logical language reflect its commitment to a conceptualization. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating this set of intended models" [41].

Conceptualization: "$C =< \mathcal{D}, \mathcal{W}, \mathcal{R}$ where C is a concept description, $\mathcal{D}$ is a set of facts of a domain or database, $\mathcal{W}$ are the possible worlds and R is the set of conceptual relations on the domain space $\mathcal{D}, \mathcal{W}$. Thus, a conceptualization is defined by its intentional/conceptual relations instead of a domain space with ordinary mathematical relations. Specifically the intentional relations are defined on possible worlds $\mathcal{W}$ not generally on the domain $\mathcal{D}$" [41].

Intentional / conceptual relations: "ρ is defined as a function from $\mathcal{W}$ to the set of all relations on $\mathcal{D}$. All conceptual relations of a possible world will contain the admittable extensions of ρ".

Ontological commitment: "$\mathcal{K}$ is a intensional interpretation of a conceptualization C known as Knowledge Base (KB). An interpretation is defined by assigning elements of the set of conceptual relations $\mathcal{R}$ in C to predicate symbols of $\mathcal{V}$" .

Intended models: "The set of all models of a given ontology language that are compatible with $\mathcal{K}$ will be called the set of intended models. Models of are extensional interpretations in the form of assignments of elements of $\mathcal{D}$ and $\mathcal{R}$ to $\mathcal{V}$. Intended models can be seen as the subset of models that are consistent with the conceptualization".

Ontology: "$\mathcal{O}$ for a language approximates a conceptualization C if there exists an ontological commitment $\mathcal{K}$ such that the intended models of according to $\mathcal{K}$ are included in the models of $\mathcal{O}$. In other words, an ontology for L is a set of axioms designed in a way such that the set of its models approximates the set of intended models of according to $\mathcal{K}$". An illustration is shown in Figure 2.2.

It is relevant that the detailed formal definition of [41] shows that foundational parts of the ontology are defined as axioms and are used as an approximate reality of a domain. An important characteristic of defining an ontology is the use of formal semantics. This means that an appropriate ontology language must be chosen and its purpose well-defined to represent a concrete set of specification capturing certain aspect of the world.

Here, we would emphasize that using formal ontologies requires a well-defined formal ontology language because it can support reasoning functionality. Without reasoning capabilities there is no advantage of any ontology languages in the context of automated processing. Computers cannot support automated reasoning if the machine readable artifacts are not using formal representation. This notion must be a deciding factor in choosing a right representation while formulating knowledge-base. However, reasoning can be considered and choosen differently in different phases of the ontology life cycle [44]. Now we present the use of reasoning capabilities in different phases of the ontology. Firstly, it can be utilized in the design phase i.e. conceptualization of an artifact. Ontology reasoning can be used to check contradictions or unintended interpretations and consequences for a set of axioms, like synonymous concepts or annotations, subsumption relationships etc. Secondly, when different ontologies are aligned and integrated then it is important to compute the integrated concept hierarchy and check for consistency. Third use of reasoning occurs during deployment phase of the ontology. For example, in determining subsumption hierarchy of concepts for certain set of facts might be checked for consistency with the corresponding ontology axioms. There exists a large number of ontology languages for formal knowledge representation. The two most popular logical formalisms for ontology languages are based on are First Order Logic (FOL) and Description Logic (DL). FOL is more expressive and powerful than DL. However, there is a trade-of between expressivity and complexity, making DL computationally less demanding. The DL on the other hand provides a number of reasoning services which allows easy construction of subsumption hierarchies and the checking of consistency of the semantic descriptions. The DLs provides clear semantics which makes it easier to encapsulate and use all the knowledge in form of an ontology and to make it consistent and complete representation of the world.

By simplest definition, any formal language can be used to define formal semantics. A formal ontology has a formal semantics if and only if it defines an entailment relation precisely for a statement in a language that entails an unambiguous interpretations of those statements. In very simple terms, a formal ontology has a formal semantics only if it supports deductive inference. We will provide a detailed definition of formal semantics using the examples of an ontology language construct of OWL DL in the following sections. For research purposes, formal semantics serves as a backbone of the SW community. It allows for ontology sharing, fusion and translation in an efficient way. However, we identify here some of the key factors which ontologies need to fulfill to be applicable to the analytical framework.

1. The ontology must contain list of domain-specific terms defining that can be

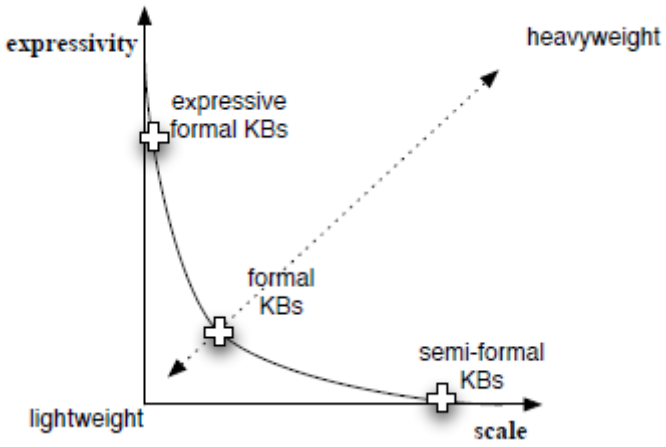

Figure 2.3: Expressivity and complexity of heavy-weight vs. light-weight ontologies and existing datasets.

represented in form of concepts and properties of concepts as well as relations. Furthermore, a domain of interest must be supported by instance data. In addition, there must be a mechanism to determine the membership of instances to concepts and relations. The DL term for that inference service is realization of a certain class that an individual belongs to.

2. The ontology must define a logical theory that can be checked for consistency. It must support tools that can be used to identify and compute any contradictory facts in the model. In DL terminology, this is called an operation to check the consistency of an ABox with respect to a TBox.

2.1.3 Existing Ontology Models

The dilemma of most adopted formal ontologies used in research or in industry is that if they are highly expressive than they have small set of axioms and if they are poorly expressive then they are large scale with huge set of axioms. Figure 2.3 visualizes the current status of existing ontology models.

Input and output in a formal representation is complex and unintuitive for an uninformed user that can be potentially processed for automated reasoning. However, it requires a large group of users to provide information in order to make a knowledge base available for use. Another problem lies in the requirements imposed by a large group of users to engineer an ontology which ultimately results in difficulty to maintain reasonable level of consistency of a knowledge base. It is an important observation that a representation of knowledge in a logical format can be extremely difficult especially in cases where uncertainty, contradictions and rapid changes to the semantic descriptions needs to be supported. Such requirement may arise as a result of the way the data is being captured or integrated from distributed systems as

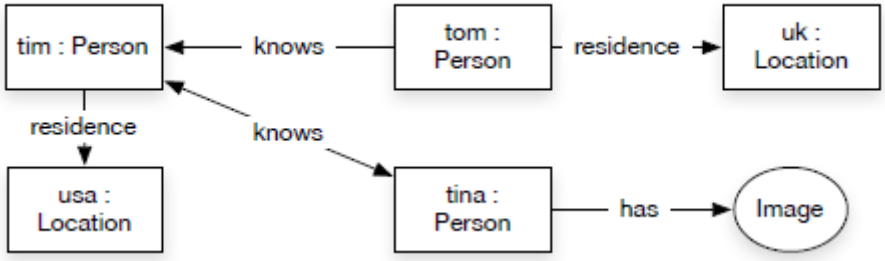

Figure 2.4: Example of a typical RDF-graph.

well as potentially from sources with varying quality. Overlapping schemas, different formats of same information an also result in such requirements. However, relaxing the assumption and formulation of a consistent ontology can reduce the requirements on automated reasoning and the computational expressivity together. This results in an inherent contradiction of an ontology being either formally expressive or a large scale ontology model.

2.1.4 State-of-the-art of the Ontology Languages

The World Wide Web Consortium (W3C) is one of the prime international standards organization for the WWW. Its main responsibility is to develops standards and recommendations for the SW. In the following sections, we will discuss the main ontology languages supported by the SW standards, that are, RDF, RDFS and OWL.

In general, RDF constructs are useful in making statements about data itself whereas RDFS supports schema formulations and subclass hierarchies, and OWL is widely used to formulate additional background knowledge. In a very simplistic way, statements in RDF, RDFS and OWL can all be represented as one combined directed graph depicted in figure 2.4

A common semantics of these languages rely of languages constructs of RDFS and OWL that support in defining domain-independent interpretations.

Ontologies in RDF(S)

The most recommended ontology language construct in the world of Semantic Web is the resource description framework (RDF). It is mainly used to locate and express information about web resources available on WWW (e.g, meta data/annotations). However, it is also well-suited to describe all other forms of structured data, e.g. data from legacy relational databases and applications. In the context of RDF, each data resource is defined by using a unique identifier. This unique identifier

is constructed using a uniform resource identifier, URI. Each statement in RDF is of a triple form (subject, predicate, object). For example tim of a type Person, tim has a full name Tim Miller is defined as a triple. A triple can be conceived as a directed arc, labeled by the property (predicate) and directing from a subject node to a property value node. The subject of any sentence is always a URI, the property value is either also a URI or a literal (e.g, String, Boolean, Float). In the first case, one can denotes a property as object property and a statement as an object-to-object statement. In the latter case one defines a datatype property corresponding to an object-to-literal statement. A complete knowledge base (triple store) can then be represented as a directed graph, a semantic knowledge graph as in figure 2.4. One can consider a triple as a tuple of a binary relation property (subject, property values). A triple can only represent a binary relation involving a subject and a property value. Each resource either a subject or an object can be associated wit one or more resources (i.e., classes) by defining them as a type-property. On the other hand, every concept can also be interpreted as a property value in a type-of statement. Conversely, each and every concept or a resource represents all instances belonging to a certain concept or its type. Concepts are defined in the RDF vocabulary description language, RDF Schema known as (RDFS). Both, RDF and RDFS formulate to a combined RDF/RDFS graph. By defining all concepts in RDF language, the corresponding RDFS schema graph also contains additional properties that have a predefined semantics, that implement some formal entailment rules respectively.

Ontologies in OWL DL

As discussed in the previous section, RDF/RDFS are mainly used as formal ontology language to model resources with low expressivity. There is still a need of a more expressive ontology language to model and represent complex knowledge structures with formal semantics. In this section, we present a very popular ontology language named OWL DL which is based on strong and well-defined description logic (DL). DL constructs consists of classes and properties same as defined and used in case of RDFS. However, in OWL DL these classes and properties can be presented and structured in complex ways. It comprises of a subset of first-order logic, which is considered more efficient and effective in case of decision making and problem solving. Examples of facts expressed in OWL DL are:

- A person has exactly one date of birth.
- A person is either female or male.
- Persons that can drive a car, are not in the age group 0-16.

These complex statements are described by logical constructors. We will briefly present these constructs in the following paragraph together with the formal semantics of OWL DL.

Constructor	DL Syntax
intersectionOf	$C_1 \sqcap ... \sqcap C_n$
unionOf	$C_1 \sqcup ... \sqcup C_n$
complementOf	$\neg C$
oneOf	$\{o_1, ..., o_n\}$
allValuesFrom	$\forall R.C$
someValuesFrom	$\exists R.C$
value	$\exists R.\{o\}$
minCardinality	$\geq nR.C$
maxCardinality	$\leq nR.C$
cardinality	$= nR.C$

Figure 2.5: DL Constructors.

OWL DL Syntax: "OWL DL is a language that provides a good balance between expressivity and complexity of logical inferencing tasks. Ofcourse, it is less expressive than first order logic but its inferencing mechanism is still computationally acceptable and decidable. However, in practical sense OWL DL is considered more expressive than RDFS. OWL DL does not align with some RDFS constructs and thus its is clearly not a superset of RDFS family. The description logic equivalent to OWL DL is called SHOIN(D). The basic foundational elements of OWL DL ontologies are classes C, properties R and individuals o. Here the properties are also termed as roles in OWL DL. *Person(tim)* represents that an individual named tim belongs to the class Person. The relation *knows(tim, tom)* is an abstract property. Individuals are considered as constants in FOL, classes as unary predicates and roles as binary predicates. Generally, OWL DL classes and concept constructs support in defining complex sets of axioms or relations. A list of all logical constructors that can be used in OWL DL are presented in figure 2.5" [45].

"The empty class is denoted as $\bot \equiv C \sqcap \neg C$ and the class that contains all individuals $\top \equiv C \sqcap \neg C$. In RDFS, the range constraint are expressed by $\top \sqsubseteq \forall R.C$ whereas the RDFS domain constraint is expressed as $\exists R.T \sqsubseteq C$. OWL DL supports a number of additional axioms that are used to define restrictions on classes or concepts (see Fig. 2.6)" [45].

Semantic: Formal semantics of SHOIN(D) mainly comprises of definition of entailment relation. Here, entailment relation is defined firstly by providing an interpretation for individuals, classes and roles and secondly by providing interpretation for axioms. "The interpretation of individuals, classes and roles is a function to elements of a particular domain $\mathcal{D}$, respectively. Hereby, identifiers of individuals can be assigned to elements of $\mathcal{D}$, identifiers of classes to $2^{\mathcal{D}}$ and identifiers of roles to $2^{\mathcal{D}} \times \mathcal{D}$. Here the formal interpretation of each item of the vocabulary and the interpretations of its complex constructors are listed in Fig. 2.5. For instance, $C1 \cup C2$ denotes the union $\cup$ of all identifiers of the individuals of C and $\mathcal{D}$. Secondly, the interpretation for every DL axiom (see Fig. 2.6) is determined. In this case, the identifiers are not mapped to domain descriptions, but assignments of individuals

Axiom	DL Syntax
SubClassOf	$C_1 \sqsubseteq C_n$
EquivalentClasses	$C_1 \equiv ... \equiv C_n$
SubPropertyOf	$R_1 \sqsubseteq R_n$
SameIndividual	$o_1 = ... = o_n$
DisjointClasses	$C_i \sqsubseteq \neg C_j$
DifferentIndividuals	$o_i \neq o_j$
inverseOf	$R_1 \equiv R_2^-$
Transitive	$R^+ \sqsubseteq R$
Symmetric	$R \equiv R^-$

Figure 2.6: DL Axioms.

are mapped to truth values. For instance $C(a)$ is interpreted as true if the individual identified with a is element of class C Or $C_1 \sqsubseteq C_2$ is considered true if each identifier of an individual is member of C_1 and C_2. In order to determine a complete closure to the definition of formal semantics of OWL DL, one is obliged to define the satisfiability of the entailment relation. Given a SHOIN(D) Knowledge Base $\mathcal{K}$ in form of a set of classes, properties, individual and axioms, we define an interpretation to be a model of this $\mathcal{K}$ if every axiom is assigned with a truth value. Now If such model exists then this $\mathcal{K}$ is specified to be satisfiable" [45].

Reasoning with OWL DL

Reasoning is an important feature of using ontology languages. In this section, we will present some inference tasks that are supported by reasoners using OWL DL constructs. In addition, we will also discuss methods used to provide reasoning services in SW application. Ontologies based on OWL DL language are formally represented where truth values for assertions not present in ABOX can also be computed. This means that such formalism can be used to ask user queries about the concepts and instances together. The most generic decision problems are user queries to the knowledge base are based on membership checking of individuals. The more advanced queries solve for tasks like subsumption and concept consistency of the knowledge base. In the following, we list and present most generic inference tasks support by OWL DL:

Instance Membership: "checks whether an instance a is member of a class C. $C(a)$ can be entailed if $\mathcal{K} \cup \neg C(a)$ is unsatisfiable".

Realization: "is the retrieval of all instances that are members of a specific class. It finds the most specific classes that an individual belongs to. Or, in other words, computes the direct types for each of the individuals. Using the classification hierarchy, it is also possible to get all the types for that individual".

Subsumption: "is used to find out if C_1 is a subclass of C_2. This is the case if $\mathcal{K} \cup (C1 \cap C2(a))$ is unsatisfiable. It is used to compute the subclass relations between every named class to create the complete class hierarchy. The class hierarchy is essential to answer queries such as getting all or only the direct subclasses of a class. There are similar problems that are focused on the TBox like checking if two classes are equivalent or two classes are disjoint".

Concept Satisfiability: "checks whether a concept is meaningful or more precisely if it is possible for a concept to have any instances. If class is unsatisfiable, then defining an instance of the class will cause the whole ontology to be inconsistent".

Consistency: "is the most essential inference task. To check whether a certain KB is consistent, it needs to be decided if the $\mathcal{K}$ is not unsatisfiable. This ensures that an ontology does not contain any contradictory facts. In DL terminology, this is the operation to check the consistency of an ABox with respect to a TBox. All problems listed can be reduced to the last task of $\mathcal{K}$ consistency".

The inference algorithm proposed in this thesis also uses domain $\mathcal{K}$ consistency checks during the inference process. The standard inference algorithms checks the unsatisfiability of a $\mathcal{K}$ using tableaux algorithms. It tries to build a tree-like model, the tableaux, by starting with an empty Tableaux and iteratively adds logical entailments of the $\mathcal{K}$. It terminates either when a contradiction occurs or no more rules are applicable. If each branch in the tableaux contains a contradiction, there is no model and the KB is considered to be inconsistent.

Querying

The most popular and recommended query language in the world of SW is SPARQL (SPARQL Protocol and RDF Query Language). The SPARQL syntax is similar to that of relational database query languages. It mainly comprises of a search template which is a directed graph. Such graph may have variable nodes (such as a graph pattern). The result of a SPARQL query can be retrieved in form of a list of binding variables or in a form of an RDF-graph structure.

2.2 Ontology-based Data Access

With the recent advancement in SW, data access and retrieval is a prime application. Ontology Based Data Access (OBDA) is one of the prominent approaches for such applications in which an ontology is used to mediate between user queries and data sources. The ontology provides a unified view of the world as well as a single point of access. In addition, it also allows users to formulate queries using the vocabulary of the conceptual model. This conceptual model provides an abstraction

and hides away the complex implementation of the underlying database schemata. In such a case, domain experts are able to express their information needs and queries using their own domain terminologies without having any prior knowledge about the way the data is captured and stored at the source and receive answers in the same expressive domain language. In the OBDA setting, this ontological concepts are connected to data columns by a set of mappings. These mappings are declarative specifications that connects each ontological term with queries over the underlying data. The magic of OBDA system is that once the mappings are in place, it automatically translates ontological queries, i.e., SPARQL, into database queries, i.e., SQL, and delegate execution of SQL queries to the database systems hosting the data. Such system like OBDA is a natural fit to address industrial data access challenges. In cases, when a complex database is presented to users via an ontology, then it is easier for users to formulate queries in terms of classes and properties in an object-centric fashion. Moreover, OBDA is a popularly known as a virtual approach, where an abstract access layer lies on top of databases while leaving the data in its original database. However, OBDA has potential to further improve data access with a minimal change to existing data management infrastructure.

2.2.1 Definition of OBDA

Generally, an OBDA instance is defined as "a quadruple $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$ where $\mathcal{D}$ is an database, $\mathcal{V}$ is an ontological vocabulary, i.e., a set of classes and properties, $\mathcal{O}$ is an ontology over $\mathcal{V}$, i.e., a set of axioms expressed in a fragment of first-order logic, and $\mathcal{M}$ is a set of mappings between $\mathcal{D}$ and $\mathcal{V}$, i.e., assertions of the form: $P(f(x), f(y)) \leftarrow SQL(x, y)$ or $C(f(x)) \leftarrow SQL(x)$ where C and P are class and property names from $\mathcal{V}$, $SQL(x)$ and $SQL(x, y)$ are SQL queries over $\mathcal{D}$ with one and two output variables, and f is a function casting values returned by SQL into URIs and values (e.g., strings, dates)" [27].

Here an ontological query Q expressed in terms of $\mathcal{V}$ over $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$, where the user can executes data queries over $\mathcal{D}$ where usually SQL queries occurs in $\mathcal{M}$, then the engine uses the computed answers to populate the extensions of the corresponding classes and properties occurring in $\mathcal{M}$, which ultimately constitutes as a set of ontological facts $\mathcal{A}$. Then, the last and final set of evaluation occurs for the query Q over $\mathcal{A} \cup \mathcal{O}$. Since $\mathcal{A} \cup \mathcal{O}$ is based on a logical theory, the query answering mechanism, over this corresponds to logical reasoning and is defines into a concrete and precise set of answers. Consequently, these answers are known as set of tuple t to the query Q over $\mathcal{A} \cup \mathcal{O}$ if $Q(t)$ holds in every first-order model of $\mathcal{O} \cup \mathcal{A}$ [23]. Sometimes the computation of $\mathcal{A}$ and precise answers, can be computationally very expensive, with worst-case complexity depending on both ontology axioms and the corresponding expressivity of the query language. In particular, work presented in [23] shows that the computation is tractable in data complexity (i.e., in the size of $\mathcal{D}$) if following two conditions are met. Firstly if the ontological queries Q are conjunctive (CQs) in nature. Secondly if the ontologies $\mathcal{O}$ are expressed in OWL

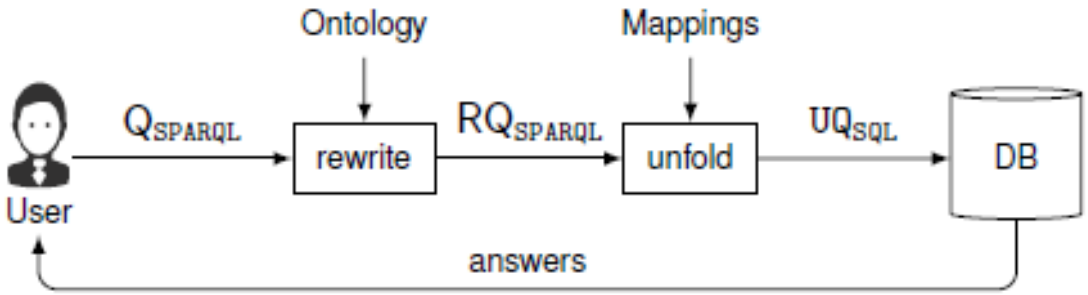

Figure 2.7: Query processing in OBDA.

2 QL. However, computation and query answering mechanism to obtain precise answers can be accomplished by implementing two step method of (i) rewriting and (ii) unfolding as depicted in Figure 2.7.

The rewriting method considers ontological queries. It mainly constitutes the compilation of relevant ontological information into the query Q. Technically, it can be considered as a resolution procedure used in Prolog language, and can be achieved by a perfect reformulation algorithm [23]. Such resolution takes a conjunctive query Q and an OWL 2 QL ontology $\mathcal{O}$ as input and returns another union of conjunctive queries RQ. Computation of certain answers for RQ over $\mathcal{A}$ will eventually return the same answers as for Q over $\mathcal{A} \cup \mathcal{O}$. During unfolding procedure, the inputs RQ and $\mathcal{M}$ are considered and RQ is translated into an SQL query UQ by mainly substituting occurrences of classes and properties in RQ with the SQL queries that they correspond and are mapped to in $\mathcal{M}$. Evaluation of UQ over $\mathcal{D}$ will effectively returns the certain answer computed by RQ over $\mathcal{A}$ and thus by Q over $\mathcal{A} \cup \mathcal{O}$.

2.2.2 State-of-the-art OBDA Systems

In this section, we present the most important state-of-the-art OBDA systems and their constructs [2]. A typical OBDA system investigates query answering over a given ontology, implements and uses mappings to fetch the data from the original data resources to the given ontology concepts. The main idea behind these system is to reduce the demanding data access problems to a model checking problem over the data sources, which in most cases are traditional relational databases. The main focus of such reduction is also motivated by the demand to enable computationally feasible reasoning services over large instance data sets i.e. ABoxes. Relatively, the size of the TBox (and the queries) are much smaller as compared to the size of the ABoxes. However, the ABox alone contributes to measure the computational feasibility of the data set, thereby fixing all other parameters (TBox, query respectively). In this context, we introduce a type of complexity is called data complexity. The notion of reduction is popularly known as first order logic (FOL) rewritability, details of which are explained in the next paragraph. Here, it is important to note that

the data complexity of answering first order logic queries w.r.t. DL-Lite ontologies is considered as a low Boolean circuits complexity class AC^0, which is roughly the class of problems that can be decided in constant time with the help of polynomially many processors.

Ontologies in DL-Lite

DL-Lite Description logics [17] is one of the most prominent and adopted representational language for ontologies. The reason for this is its formal semantics and polynomial computational properties that can support various standard reasoning services such as subsumption testing, satisfiability testing, query answering etc. As introduced in the previous section, query answering serves as a foundational component of any OBDA system and related to that is satisfiability testing of ontologies. An ontology is defined as a pair $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ that is fundamentally a TBox and an ABox. In all DLs this pair is made up by subsets of a set of concept symbols N_C , a set of role symbols N_R, and a set of individual constant symbols N_i. Additional constants (and predicates) with precise meanings over a given domain may also support for DLs with concrete domains or datatypes. Different variants of DLs differ in the set of concept/role constructors they offer and in the constraints for building TBox and ABox axioms. Typically TBox axioms are concept subsumptions $C \sqsubseteq D$ or role subsumptions $R \sqsubseteq S$ whereas ABox axioms have the form $C(a)$ or $R(a, b)$, where C, D stand for concept descriptions, R, S for role descriptions and a, b for individual constants. In the context of an OBDA system, there exist a family of DLs called DL-Lite [23] because it supports and allows for FOL rewritability. DL-Lite is a family language for another very popular representational language known as the OWL 2 QL profile which is currently W3C recommended web ontology language (OWL). FOL rewritability is a very strong property that is the sole reason for adoption at industry and also a prominent research interest. Because of this lightweight logics such as DL-Lite are used as representation language for the ontology. However, it has its own limitations on the query language such as restrictions w.r.t unions of conjunctive queries (UCQs). (But note, that the limits of expressivity under FOL rewriting can be easily managed by extending family of Datalog language.) To make the discussion more concrete we present the syntax of a DL-Lite language and its semantics in Fig. 2.8.

The TBox axioms are additionally constrained by the language that functional roles are not allowed to occur on the right hand side of role axioms. The semantics of concept descriptions is defined recursively on the basis of an interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$, which consists of a domain Δ and a denotation function $\cdot^{\mathcal{I}}$. The denotation of concept symbols (atomic concepts) A are subsets $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ of the domain; role symbols P are denoted by binary relations $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and constants a are denoted by elements of the domain $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

There is a modeling relation which is denoted by $\models$ and one defines that an interpretation $\mathcal{I}$ models or makes true an axiom $ax if f \mathcal{I} \models ax$. Any ontology is called

$$
\begin{array}{lll}
 & & (P^-)^{\mathcal{I}} = \{(d,e) \mid (e,d) \in P^{\mathcal{I}}\} \\
R \longrightarrow P \mid P^- & & (\exists R)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists e.(d,e) \in R^{\mathcal{I}}\} \\
B \longrightarrow A \mid \exists R & & (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
C \longrightarrow B \mid \neg B & & \mathcal{I} \models B \sqsubseteq C \text{ iff } B^{\mathcal{I}} \subseteq C \\
\textit{TBox:} & B \sqsubseteq C, (\text{func } R), & \mathcal{I} \models R_1 \sqsubseteq R_2 \text{ iff } R_1^{\mathcal{I}} \subseteq R_2 \\
 & R_1 \sqsubseteq R_2 & \mathcal{I} \models B(a) \text{ iff } a^{\mathcal{I}} \in B^{\mathcal{I}} \\
\textit{ABox:} & A(a), R(a,b) & \mathcal{I} \models R(a,b) \text{ iff } (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \\
 & & \mathcal{I} \models (\text{func } R) \text{ iff } R^{\mathcal{I}} \text{ is a (partial) function}
\end{array}
$$

Figure 2.8: DL-Lite language and its semantics.

satisfiable if there is an interpretation $\mathcal{I}$ that makes all axioms in the TBox and the ABox true. An ontology $\mathcal{O}$ entails an axiom ax, in short- $\mathcal{O} \models axiff$ all models of $\mathcal{O}$ are also models of ax.

Query Answering and Rewritability

Any first-order-logic query $Q = q(x)$ is a first-order logic formula $q(x)$ whose free variables are the ones in the n-ary vector of variables x; the variables in x are called distinguished variables. If x is empty, the query is called Boolean. Let a be a vector of constants from the pair of the ontology. The semantics of n-ary FOL queries with respect to an interpretation $\mathcal{I}$ is given by the set $Q^{\mathcal{I}}$ of n-ary tuples t over the domain $\Delta^{\mathcal{I}}$ such that $\mathcal{I}_{[}x \rightarrow t] \models q(x)$ Here, $\mathcal{I}_{[}x \rightarrow t]$ extends $\mathcal{I}$ by interpreting the variables in x by the elements in t. The precise set of answers w.r.t. an ontology is managed by an certain answer semantics coming from the database theory. We are not going to discuss the appropriateness of this kind of semantics but just state its definition. (For an adequateness discussion of certain answer semantics in particular w.r.t aggregation we refer the reader to [46]). At ontological level, FOL queries are too complex to be used. Hence, in order to guarantee FOL rewritability, we introduce two well known weaker subclasses of FOL queries that is conjunctive queries (CQ) and unions of conjunctive queries (UCQ) which will be used in this thesis.

Mapping

In any traditional OBDA system, the instance data / ABox is not given or materialized in advance but produced on-the-fly by using mappings [47] such as RDB to RDF mappings. These mappings are formally presented as rules with two parts. Part one are the queries of the ontological level that uses ontology concepts, called as the target. This is the head of the rule (here the left-hand side). Part two are the queries in the the data source language (in most cases SQL) and this serves as the body of the rule, which is noted here always on the right-hand side. Now, we

```
SENSOR(SID, CID, Sname, TID, description)
SENSORTYPE(TID, Tname)
COMPONENT(CID, superCID, AID, Cname)
ASSEMBLY(AID, AName, ALocation)
MEASUREMENT(MID, MtimeStamp, SID, Mval)
MESSAGE(MesID, MesTimeStamp, MesAssemblyID,
        catID, MesEventText)
CATEGORY(catID, catName)
```

Figure 2.9: Part of the relational schema in a measurement DB.

will present the definition of a mapping in a logical notation. A recent W3C recommended mapping language in machine readable form is R2RML, a mapping language from relational databases to RDF (http://www.w3.org/TR/r2rml/). As constructing mappings is a non-trivial task, recent research considers also bootstrapping or learning these mappings.

We exemplify a mapping for a sensor measurement scenario, assuming that there is one central DB with sensor measurement data and also sensor descriptions w.r.t. the DB schema in Fig. 2.9. The ontology is assumed to model sensors, measurements, events etc. in the same manner as the nearly standard semantic sensor networks (SSN) ontology, authored by the members of the W3C Sensor Network Incubator Group (see http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/.) It is worth mentioning that any such general ontology can be reused and/or extended for specific sensor measurement scenarios by introducing new names to the signature of the ontology and adding new ontology axioms. Here, we assume that there is a concept symbol Sens (for sensors) and an attribute symbol name. ABox assertions saying which element is a sensor and what their names are, are produced by the following mapping:

$$m : \mathsf{Sens}(x) \wedge \mathsf{name}(\mathsf{x},\mathsf{y}) \leftarrow SELECT\ f(SID)\ as\ x,\ Sname\ as\ y \\ FROM\ SENSOR$$

Thus information is basically a row in the measurement table that is mapped to unary facts (Sens(x)) and binary atomic facts (name(x, y)). If the table SENSOR contains a row:

$$(123, \mathsf{comp45}, \mathsf{TC255}, \mathsf{TempSens}, \mathsf{Atemperaturesensor})$$

then the mapping produces the conjunction of ABox assertions $\mathsf{Sens}(\mathsf{f}(123)) \wedge \mathsf{name}(\mathsf{f}(123), \mathsf{TempSens123})$.

In the DL-Lite notation, these mappings have in general on their left-hand side the conjunctive queries and on their right-hand side, data source specific queries such as SQL. The mappings are logical and complete when all variables used on the left-hand side occur as columns on the right-hand side. However, the source query may contain additional variables. Here, the term f(SID) denotes an individual constant, represented as a functional term indicating the value of the attribute SID of the sensor. All expressions f(SID) could be mapped to the more convenient atomic names of the form, e.g., si. If the ontology language allows for datatypes as we assume here then we can use attribute values directly without the need of an additional functional symbol. This is constructed above for the column Sname containing strings. To ease the construction and management of mappings, they can be split up into a simpler form where the target consist of an atomic query only. Within the splitting the source query is projected to the variables occurring in the atom; in the case of the query above the resulting split mappings would be as follows:

$$m1 : \mathsf{Sens(x)} \leftarrow \ SELECT\ f(SID)\ as\ x,\ FROM\ SENSOR$$
$$m2 : \mathsf{name(x,y)} \leftarrow \ SELECT\ f(SID)\ as\ x,\ Sname\ as\ y\ FROM\ SENSOR$$

For a given database $\mathcal{D}$ and a set of mappings $\mathcal{M}$, the induced ABox $A(\mathcal{M},\mathcal{D})$ is just the union of the ABox assertions produced by the mappings in $\mathcal{M}$ over the $\mathcal{D}$. The semantics of query answering w.r.t. a set of mappings over a DB and a TBox is just the certain answer semantics introduced above and applied to the ontology $(T, A(\mathcal{M},\mathcal{D}))$. Here, an important criteria for using such mappings in an OBDA setting is that the induced ABox is not materialized i.e. transformed into graph database, for query answering instead it is kept virtual. These queries over the induced ABox are unfolded to queries over the DB on demand basis. Therefore, in a traditional approach any UCQ over a TBox and the induced ABox of mappings w.r.t. a DB is first rewritten into a FOL query, then this query is unfolded into an SQL query over the DB (using these mappings) and then the unfolded query is evaluated over the DB, given the set of answers to the original query. There is no canonical way for unfolding a UCQ into a SQL query, and, indeed, different strategies for unfolding a UCQ w.r.t DL-Lite ontologies are presented and applied in the literature, e.g., as one strategy is introduced in [48, 49]. The common idea of many strategies used in industry and presented in literature is to present the mappings as logical rules and later use them for logical programming paradigm such as resolution to get the unfolded query. Sometimes, the rewriting of queries may even lead to an exponential blow-up, in such cases optimizations can be achieved at different levels (rewriting, unfolding and mappings) that are crucial in case of any OBDA system. Different optimization strategies are presented in [50, 51] and implemented, e.g., ontop OBDA system (http://ontop.inf.unibz.it/).

2.3 Semantically-defined Analytics Data Access

Recent efforts have been made to enrich and extend ontology models and languages with analytical and temporal constructs to meet the industrial requirements. RuleML (Rule Markup Language) is used to implement some basic aggregation functions. "RuleML is a rule language formulated in XML and primarily uses datalog constructs. Datalog is a function-free fragment of Horn clausal logic". In principle, RuleML allows the formulation of if-then-else types of rules. Here is important to note that both RuleML and OWL DL are different subsets of first-order logic (FOL). Another potential candidate to define analytics is SWRL (Semantic Web Rule Language). It is a popular and simplest Semantic Web rule language, that combines sub-languages of OWL (OWL DL and Lite) with those of the Rule Markup Language (Unary/Binary Datalog). Datalog clauses are usually important for modelling background knowledge in cases where DL might be inappropriate, for example in many industrial applications with integrity constraints.

Authors in [52, 53] support for temporal operators in queries and ontologies. Still, their approaches using temporal logics (e.g., LTL) are not adequate in cases where sensor data are organized based on intervals, e.g. [0s; 10s]. Works in [1, 12] introduce analytical operations directly into ontological rules in such a way that OBDA scenario is preserved. They define analytical functions on concepts, e.g. avg C, in OBDA setting. However, the authors do not consider temporal dimension of the rules. As discussed above, our work is strongly related to the work on well-studied Metric Temporal Logic [54]. In particular, we use a non-trivial extension of nonrecursive Datalog language DatalognrMTL which suitable for OBDA scenario. DatalognrMTL is introduced in [37] where the authors conduct a theoretical and experimental study investigating computational characteristics of the language. They show how query answering over a program in DatalognrMTL can be rewritten into the problem of query answering in SQL. In [55], they also describe how to leverage DatalognrMTL in a full-fledged temporal OBDA system. Following similar principles, we define rewriting of our proposed language into SQL and show that such rewriting performs reasonably well on sensor data. Another related direction is real-time processing of signal data streams. In this direction, most of the work done so far mainly focused on querying RDF stream data. Many different approaches such as C-SPARQL [35], SPARQL stream [56] and CEQLS [57] have surfaced in recent years, introducing SPARQL based query processors. Most of them, apart from C-SPARQL, follow the Data Stream Management Systems (DSMSs) paradigm and do not provide support for stream reasoning. EPSPARQL [58] combines SPARQL with complex event processing features, and includes sequencing and simultaneity operators. Unlike the others, LARS [59] is an Answer Set Programming based framework, which enables reasoning by compiling a Knowledge Base together with a SPARQL-like query into a more expressive logic program. In the future, we plan to investigate how to incorporate the real-time computation aspect into our framework.

2.4 Summary and Discussion

The demand for industrial data analytics is exponentially growing, providing opportunities to researchers with substantial incentives. However, there exist a number of challenges that must be taken into account when dealing with industrial data sets. Most of these challenges can be clustered into two main areas of interest, i.e. data and domain.

Data: Industrial data are very diverse and heterogeneous in the sense that they often contain different formats, ill-formed semantics and complex data structures. Moreover, analytical workflows are usually written using various programming languages and composed of poorly-structured workflows. Such non-uniform characteristics often affects the performance of traditional data analysis approaches.

Domain: Industrial systems are usually of interdisciplinary nature where experts from various domains combine there expertise for problem-solving. Approaches to analysing data sets from a given use-case corpus, are therefore required to adapt to the domain and semantics of the data set in that corpus. As discussed earlier, this is because the intent of the data analysis often changes with respect to their context in a given use-case scenario.

The above challenges demands for developing semantic-aware analysis approaches that happen to be more adhesive to the diverse and sparse nature of data and more flexible in adapting to newer domains and features of data analytics.

In this chapter, we introduced the fundamentals of semantic technology and discussed the key components of existing ontology languages. These component are the foundation and govern the rules by which we have constructed our semantic language for analytics. Furthermore, we presented the ingredients of traditional ontology-based data access that is an essential part of our solution. We also discussed the building blocks of OBDA systems and highlight its research challenges. Lastly, we presented and reviewed the latest development in the area of analytics-aware OBDA technology which still lacks inclusion of analytical and temporal operators into a single semantic framework and flexibility to cater complexity, interoperability and data challenges.

In the following chapters we present our results and details our proposed solutions addressing each of our research question, along with our evaluation sets that measure the efficiency of our proposed models, language and system as a whole.

3 Ontology Models for Domain-specific and Analytics-aware Semantics

In this chapter we present the use of ontology models to represent domain-specific models for industrial applications and extract analytical aware semantics to develop analytical ontology. The representation of industrial information models and standards using ontologies has been widely acknowledged as a non-trivial task [60, 61, 62, 63, 64]. However, we present a new approach that entails the semantics of large and complex technical systems and to find synergies with their underpinning models in applications. Their design has been driven towards fulfilling the same purposes as the models they originate from, that is, to act as schema-level templates for data generation and exchange, and to enable the formulation and execution of analytical queries. Our conclusion is that semantic machine-readable models that considers domain-specific and analytics aware semantics produce, in most cases, a higher performance w.r.t data access and integration as compared to those that merely rely on static models or standards.

The material in this chapter has been published in [65, 66, 67, 68, 69, 70].

3.1 Introduction

Software systems in the industries have become increasingly important in recent years. Production machines, such as assembly line robots or industrial turbines, are equipped with and controlled by complex and costly pieces of software, according to a recent survey, over 40% of the total production cost of such machines is due to software development and the trend is for this number only to continue growing [71]. Additionally, many critical tasks within business, engineering, and production departments (e.g., control of production processes, resource allocation, reporting, business decision making) have also become increasingly dependent on complex software systems. Recent global initiatives such as Industry 4.0 [72, 73, 74, 75] aim at the development of smart factories based on fully computerised, software-driven, automation of production processes and enterprise-wide integration of software components. In smart factories, software systems monitor and control physical processes, effectively communicate and cooperate with each other as well as with humans, and are

in charge of making decentralised decisions. The success of such ambitious initiatives relies on the seamless (re)development and integration of software components and services. This poses major challenges to an industry where software systems have historically been developed independently from each other. There has been a great deal of research in recent years investigating key aspects of software development in industrial manufacturing domains, including life-cycle costs, dependability, compatibility, integration, and performance (e.g., see [76] for a survey). This research has highlighted the need for enterprise-wide information models that are machine readable conceptualisations describing the functionality of and information flow between different assets in a plant, such as equipment and production processes. The development information models based on ISA and IEC standards has now become a common practice in modern companies [77]. In practice, however, many types of models co-exist, and applications typically access data from different kinds of machines and processes designed according to different models. These information models have been independently developed in different (often incompatible) formats using different types of proprietary software, furthermore, they may not come with a well-defined semantics, and their specification can be ambiguous. As a result, model development, maintenance, and integration, as well as data exchange and sharing pose major challenges in practice.

Adoption of semantic technologies has been a recent development in many large companies such as IBM [78], the oil and gas company Statoil [27], and Siemens [79, 80, 12, 81]. An important application of these technologies has been the formalisation of information models using OWL 2 ontologies and the use of RDF for storing application data. OWL 2 provides a rich and flexible modelling language that seems well-suited for describing industrial information models: it not only comes with an unambiguous, standardised, semantics, but also with a wide range of tools that can be used to develop, validate, integrate, and reason with such models. In turn, RDF data can not only be seamlessly accessed and exchanged, but also stored directly in highly scalable RDF triple stores and effectively queried in conjunction with the available ontologies. Moreover, legacy and other data that must remain in its original format and cannot be transformed into RDF can be virtualised as RDF using domain-specific ontologies following the Ontology-Based Data Access (OBDA) approach. Domain-specific ontologies such as SSN (https://www.w3.org/2005/Incubator/ssn/ssnx/ssn), QUDT (http://www.qudt.org/qudt/owl/1.0.0/qudt.owl), Formal Ontology [41] have previously been developed to capture some aspects of technical systems. For example, SSN only describes the capabilities of sensors, measurement processes, and resulting observations. Closer to our work is the upper ontology based on ISO 15926 [82]. However, it entails slightly loose definitions that can hamper its applicability for a specific domain use-case and fails to model the deployment, configuration, operational and analytical functional aspects of the industrial system.

In this chapter, we investigate extracting and using the domain-specific and analytical semantics of technical systems on industrial data, aiming mainly at addressing the above limitations of traditional semantic and non-semantic-based approaches

and consequently improving their data analysis performances. The research question we aim to address in this chapter is:

[RQ1]Can domain-specific and analytical-aware ontology models for industrial equipment enhance data analysis performance?

In order to address this research question, we propose a generic technical system ontology model called TechOnto, that integrates the existing models for sensor networks and quantity-related attributes and extends them with important concepts of technical systems such as deployment, functions, configuration, analytical processes and so forth. The ontology adopts a modular approach to enable sharing of knowledge and integrating information across the industries for multiple use-cases. Domain experts may extend the domain model by linking to their own ontologies, or existing knowledge bases. In our discussion, we stress the modelling choices made when formalising these models as ontologies and identify the key OWL constructs required in this setting. Our analysis revealed the need for integrity constraints for data validation [83, 84], which are not available in OWL 2. Hence, we discuss in detail what kinds of constraints are needed in industrial use cases in general and how to incorporate them. We then illustrate the use of reasoning services, such as concept satisfiability, data constraint validation, and query answering for addressing application requirements.

Our proposed ontology model is currently being maintained and used in industry (i.e. at Siemens Power Generation business). In order to widen the scope of application of semantic technologies in the company it is crucial to make ontology development accessible to teams of engineers. To this end, we have developed the Semantic Ontology Model Manager (SOMM) a tool that has been designed to fulfil industrial requirements and which supports engineers with little background on semantic technologies in the creation and use of ontologies.

The rest of the chapter is organized as follows: In Section 3.1 we discuss the background and motivation of the proposed ontology, Section 3.2 discuss the methodology and modelling choices underpinning the design of the model and identify a fragment of OWL 2 QL that is sufficient to capture the basic aspects of the information models. Our analysis of the model, however, also revealed the need to incorporate database integrity constraints for data validation, which are not supported in OWL 2. Thus, we also discuss the kinds of constraints that are relevant to data analysis tasks. We present how the OWL 2 QL axioms and integrity constraints can be captured by means of rules with stratified negation for the purpose of data validation and query answering. In Section 3.3 describes the upper ontology in more detail and highlight its design constructs and modules for knowledge representation and data analytics. In Section 3.4, we describe the developed tool - SOMM that provides a simple interface for ontology development and enables the introduction of instance data via automatically generated forms that are driven by our TechOnto ontology.

Background and Motivation

A technical system refers to a system (or a network of systems) with a high degree of complexity embedded within a larger infrastructure [85]. Various kinds of systems such as turbine systems, rail systems, manufacturing systems, smart grids are recognized as important artifacts of modern technology. They are functionally integrated into the process of industrialization and economic growth. The focus of today's information technology is to analyze the development and function of these technical systems to enhance management, quality control and intelligent decision making.

Industries today have adopted a common strategy of "Monitor, Assess, Predict and Optimize" to serve their purpose [13][4]. This means that at first, solutions must monitor the feature of interest and observe related data properties and processes of the system or component. Secondly, a solution within a system reasons about its current state based on the information model and observation data. Third, it predicts the future health states based on current assessment and finally infers actionable information to optimize the life cycle cost of the monitored system or component versus the owners desired value driver such as uptime. These solutions utilize the power of information and expert knowledge to provide reliable results. But even today they are split over many different models, software solutions, and processes. This heterogeneity and complexity of information systems makes it difficult for system stakeholders to share, exchange, trace and sustain relevant information. In this context, recent studies [10] have highlighted the need of an *industry-wide* knowledge representation approach. This means that these information models should be a machine-readable specification of the technical artifacts and represent key concepts by means of their set of properties, relationships, rules, and constraints. To this end, the aim of modern data management and modelling approaches is to capture different facets of the system development life cycle and changing requirements. For example, the quantities and properties of a component, its design related meta-data or different operating modes of sensor device together with its measurements.

A gap has developed between system complexity and the users ability to model information that the system provides. To bridge this gap, a common specification for 1) domain-centric knowledge and 2) data-centric knowledge (e.g. information entities, protocols, data formats, frameworks, and architectures) needs to be developed. There is a need to share knowledge and integrate a rich diversity of the generic and specific domains, machines and software agents to enable automation and intelligent decision making. Consequently, these entities must be equipped with specification on how to interact with and understand the semantics of the exchanged information.

However, modelling technical systems in a modular way is not a trivial task. Comprehensive study of the literature as presented in Chapter 2 and interviews and sessions conducted with domain experts from different domains of turbo-machinery,

mobility, manufacturing and smart grid units highlight the need to develop a common model and to find synergies between a variety of existing models within an industry and outside in order to promote knowledge sharing and ease of information exchange to accomplish efficient analytics. In the following, we explicitly specify our motivation of developing an domain ontology model - TechOnto for technical systems. More specifically, we discuss the background of investigating ontology based large technical system, highlighting the key problems which could be addressed by such a modular approach, and potential solution space.

1. An application dependent specification and a localized storage of the concepts related to any technical system's equipment and life-cycle processes severely limits the scalability. For such an architecture, a semantic model may be reused for logical system design, diagnosis of anomalies, maintenance planning of components etc. across a wide range of industrial assets. As opposed to the industrial standards (e.g. ISO 15926 [82]) that either define the physical side of the system or process side of the system (as in ISO/IEC 15288).

2. Data resources (DB models, logs, sensor data, texts, pictures) are typically treated in isolated platforms leading to isolated non-actionable knowledge. A semantic model may provide an ontological foundation for the various types of equipment models, their deployment profiles, design configurations, component hierarchies, part-whole relationships, functional profiles and logical bindings to other functional profiles. Likewise, state-of-the-art systems require manual intervention of the expert and collaboration with IT personnel to access available resources. The model may be used to provide access and a consistent view across industrial data resources already available in product design, engineering, operation, and service processes.

3. One of the key operational limitations of the existing methods is that the applications such as analytics or decision support lack the understanding of the system structure, e.g. where a sensor is installed, what data points it provides, and what physical quantity it measures. Semantic meta-data resolves such issues to a large extent.

4. Scalability is another issue for current systems. Analysis and adaptation are severely complicated when an existing system is extended by a new device or the configuration is changed, owing to a fixed and overloaded number of standards. A model-based approach may flexibly use logic-based reasoning to cater new system extensions and revisions.

5. The use of reasoning services [84], such as concept satisfiability, data constraint validation, and query answering is difficult because solution consist of isolated parts, which may be addressed by a unified model for various industrial automation applications including big-data solutions.

6. Furthermore, complex query pose limitations on the usability of advanced

technologies. A model may provide a schema for constructing and executing complex queries [84]. For example, a monitoring task that requires sensor measurements along with the configuration of a component within a platform. This may even lead to essential information sharing within the technical staff and promote knowledge sharing.

7. Finally, todays systems have restrictive capabilities to support fault detection, diagnosis [13], etc., typically requiring equipment know-how to explain the underlying process. Furthermore, during maintenance and service operation, technicians can only make decisions based on experience or local knowledge: A semantic model helps solve these problems.

In summary, as opposed to the state-of-the-art, the above-mentioned problems motivate that the model needs to be more than a static file with the controlled vocabulary of the desired system. Specifically, from usability perspective, the ontology solution framework should at least be:

- defined by general-purpose terms of the technical system in a modular fashion so that more specific classes and relations can be defined.
- acted upon as schema-less templates for data access, integration, and interoperability, and to enable the formulation and execution of queries.
- supportive of modifications, collaborative development, and an interactive model management [67].
- available online and support web-services, interfaces (such as REST API) utilized by different users and applications.
- extensible for future use cases based on commercial analytical and reporting solutions e.g. KNIME (https://www.knime.org/) workflows, Spotfire (http://spotfire.tibco.com/) Reports.

Hence we propose that the application of the semantic models provides a machine-readable format including other formats such as XML. It furthermore elaborates on contextual reasoning capabilities, where experts can retrieve and explore contextual information relevant (semantically similar) to their task at hand e.g. for diagnosis, control, and optimization.

3.2 Methodology

In this section, we first present the knowledge representation(KR) methodology and modelling approaches used in our work.

3.2.1 Ontology Development Methodology

We used customised NeOn framework to develop our TechOnto ontology [86]. NeOn framework provides nine different scenarios comprising of 59 activities. The NeOn core scenario lists each ontology development process separately. The execution of each scenario relates to various phases of the underlying life cycle model. There are two life cycle models included in the framework. First one is waterfall model which consist of variable number of phases that may depend on the scenarios to be executed. Secondly, an iterative and incremental model where a sequence of waterfall models can be supported. Each model here can belong to a different set of scenario. The framework stores scenarios and activities in a glossary of terms, aiming to give commonly accepted definitions for certain activities. Each activity has a set of comprehensive descriptions consisting of functional descriptions (e.g., definition, goals, and input/output). The technical system ontology as presented in this thesis is the result of a number of iterations of the overall ontology engineering process, which is based on an iterative and incremental life cycle model. So far, both the NeOn core scenario and the NeOn scenario for the reuse of ontological resources have been used as part of the thesis. In addition to this, we also adapted some of the NeOn activities to meet our requirements therein keeping the engineering process as lightweight as possible. In the following, subsequently performed activities are described in more detail in the order of their execution.

Knowledge Acquisition: We employ different activitites during the knowledge acquisition phase. Firstly, we gathered domain descriptions, their structures and instances from the domain experts. Secondly, we employ ontology learning approaches to automatically derive data descriptions from the unstructured, semi-structured and structured data sources. Within the technical system ontology engineering process, the ontology population activity is not performed during the ontology design phase, as it solely contains domain-specific conceptual knowledge.

Ontology Requirements Specification: The main challenge during the specification activity was to identify a set of appropriate competency questions (CQs). These questions help to describe the requirements set by the domain experts in a systematic way. In general, our focus here is to set requirements neccessary for accessing external data sources and data analytics tasks. The summary of the requirement specifications are already discussed in details in the previous section, which serves also as a motivation of the work in this chapter.

Ontology Conceptualization: It is suggested by the NeOn framework to create a conceptual representation of the domain in order to align the requirements lied down by the domain expert. This was achieved by listing the terms obtained during our discussions with the experts along with the documentation of their respective semantic meanings. This domain representation was iteratively enhanced until we reached a semi-formal, graphical model description of the intended ontology. Here we also considered third party ontology models that are available online and checked them against the given data sources. This activity helped us in improving the quality

of the model and reuse existing standard ontology models and engineering concepts as proposed by the NeOn framework [86]).

Ontology Reuse and Aligning: Existing (non-)ontological resources are used for the development of the TechOnto ontology. These resources encompass industrial data collected from different industrial applications mainly Siemens power generation, mobility and smart grid businesses, details can be found in Chapter 6. Moreover, existing relevant domain ontologies are identified and evaluated for their suitability in the context of TechOnto. In this context, the TechOnto ontology reuses for example, Sensor Network Ontology and QUDT ontologies to align the desired domain concepts and reuse their semantic model descriptions as much as possible.

Ontology Implementation: During this activity, we implemented the obtained conceptual model using OWL 2 QL. In addition to this, OWL 2 QL had to be extended to include complex constraints posed by the domain specifications. Such modelling challenges and approaches are presented in the following section. Due to a large number of considerations, the implementation process is supported by our in-house developed model manager. Model manager specifications are discussed also discussed in the following sections.

Ontology Annotation: In order to make our model readable across multiple industrial experts as well as across the equipment lifecycle, we provided various annotations satisfying each user context. In addition to general information (e.g., the ontology version), concepts and properties are supported using rdfs:label, rdfs:comment as well as domain specific labels.

Ontology Evaluation: Before the ontology was published, ontology evaluation is performed. First the ontology was evaluated against the requirements listed during the specification activity. Then, with use of HermiT [86] reasoner, we ensured both the consistency and general quality of our ontology models. Details on ontology evaluation and different metrics are presented in Chapter 7.

Ontology Documentation: Documentation is an important aspect of the overall ontology engineering methodology. Design decisions and code fragments must be properly registered in order to support transparency and furture extensions. TechOnto ontology is implemented in OWL 2 in order to be machine-processable and compatible for OBDA system. Thus, the OWL ontology constitutes the following: (i) classes as sets of individuals, (ii) individuals as instances of classes (i.e., real-world objects in the domain) and (iii) properties as binary relations between individuals. It also defines where possible cardinality restrictions such as domain and ranges as well as other constructs (e.g., taxonomies) to support reasoning services. The corresponding TechOnto ontology was modelled using the open-source ontology editor Protege [87] which is one of the most common tools for ontology development. To incorporate further contraints that were not supported by the standard tool, we implemented and used our in-house developed model manager (SOMM).

3.2.2 Modelling Approach

In the context of ontology formalisms, we consider the most recent and prominent variant OWL 2 QL axioms in combination with integrity constraints. This is because the standard OWL 2 QL axioms [83, 84] support subsumption hierarchies of classes and properties, transitivity, inverse properties, universal restrictions, cardinality restrictions of relationships and more. Our study of the requirements of information models revealed that many key aspects of information models naturally correspond to integrity constraints and hence cannot be captured by standard OWL 2 ontologies. This demonstrates intrinsic limitations of OWL 2 for industrial modelling and gives a clear evidence of why constraints are essential for such modelling. We propose the use of OWL 2 QL with integrity constraints to be sufficient for capturing the conceptual design of the technical system with scalable reasoning, greater expressibility, and efficient query answering. From the design point of view, our major consideration was to support modularity, such that the classes can be easily extended and integrated with more specific domain ontologies.

Modelling with Standard OWL 2 QL Axioms

From an ontological point of view, most building blocks of the the typical industrial information models are rather standard in conceptual design and naturally correspond to OWL 2 classes (e.g., *Turbine*, *Process*, *Product*), object properties (e.g., *hasPart*, *hasFunction*, *locatedIn*) and data properties (e.g., *ID*, *hasRotorSpeed*).

The specification of the models suggests the arrangement of classes and properties according to subsumption hierarchies, which represent the skeleton of the model and establish the basic relationships between their components. For instance, in the energy plant model a Turbine is specified as a kind of Equipment, whereas hasRotorSpeed is seen as a more specific relation than hasSpeed. The models also suggest that certain properties must be declared as transitive, such as hasPart, hasSpeed and locatedIn. Similarly, certain properties are naturally seen as inverse of each other (e.g., hasPart and partOf). These requirements are easily modelled in OWL 2 using the following axioms written in functional-style syntax:

$$\mathsf{SubClassOf}(\mathsf{Turbine}\ \ \mathsf{Equipment}) \tag{3.1}$$

$$\mathsf{SubDataPropertyOf}(\mathsf{hasRotorSpeed}\ \ \mathsf{hasSpeed}) \tag{3.2}$$

$$\mathsf{TransitiveObjectProperty}(\mathsf{hasPart}) \tag{3.3}$$

$$\mathsf{InverseObjectProperties}(\mathsf{hasPart}\ \ \mathsf{partOf}) \tag{3.4}$$

These axioms can be readily exploited by reasoners to support query answering; e.g., when asking for all equipment with a rotor, one would expect to see all turbines that contain a rotor as a part (either directly or indirectly).

Additionally, the models describe *optional relationships* between entities. In the manufacturing model certain materials are optional to certain processes, i.e., they

are compatible with the process but they are not always required. Similarly, certain processes can optionally be followed by other processes (e.g., conveying may be followed by packaging). Universal (i.e., *AllValuesFrom*) restrictions are well-suited for attaching an optional property to a class. For instance, the axiom

$$\begin{aligned}&\mathsf{SubClassOf}(\mathsf{Conveying}\\&\quad\mathsf{ObjectAllValuesFrom}(\mathsf{followedBy}\ \ \mathsf{Packaging}))\end{aligned}\tag{3.5}$$

states that only packaging processes can follow conveying processes; that is, a conveying process can be either terminal (i.e., not followed by any other process) or it is followed by a packaging process. As a result, when introducing a new conveying process we are not forced to provide a follow-up process, but if we do so it must be an instance of Packaging.

All the aforementioned types of axioms are included in the OWL 2 QL profile. This has many practical advantages for reasoning since OWL 2 QL is amenable to efficient implementation using rule-based technologies.

Modelling with Constraint Axioms

The main challenge that we encountered was to capture the constraints of the models using ontological axioms. We next describe how this was accomplished using a combination of OWL 2 QL axioms and integrity constraints.

In addition to optional relationships, the information models also describe relationships that are inherently *mandatory*, e.g., when introducing a new turbine, the energy model requires that we also provide its rotors.

This behaviour is naturally captured by an integrity constraint: whenever a turbine is added and its rotors are not provided, the application should flag an error. Integrity constraints are not supported in OWL 2; for instance, the axiom

$$\begin{aligned}&\mathsf{SubClassOf}(\mathsf{Turbine}\\&\quad\mathsf{ObjectSomeValuesFrom}(\mathsf{hasPart}\ \ \mathsf{Rotor}))\end{aligned}\tag{3.6}$$

states that every turbine must contain a rotor as a part; such rotor, however, can be possibly unknown or unspecified.

The information models also impose cardinality restrictions on relationships. For instance, each double rotor turbine in the energy plant model is specified as having exactly two rotors. This can be modelled in OWL 2 using the axioms

$$\begin{aligned}&\mathsf{SubClassOf}(\mathsf{TwoRotorTurbine}\\&\quad\mathsf{ObjectMinCardinality}(2\ \ \mathsf{hasPart}\ \ \mathsf{Rotor}))\end{aligned}\tag{3.7}$$

$$\begin{aligned}&\mathsf{SubClassOf}(\mathsf{TwoRotorTurbine}\\&\quad\mathsf{ObjectMaxCardinality}(2\ \ \mathsf{hasPart}\ \ \mathsf{Rotor}))\end{aligned}\tag{3.8}$$

Such cardinality restrictions are interpreted as integrity constraints in many applications: when introducing a specific double rotor turbine, the model requires that we also provide its two rotors. The semantics of axioms (3.7) and (3.8) is not well-suited for this purpose: on the one hand, (3.7) does not enforce a double rotor turbine to explicitly contain any rotors at all; on the other hand, if more than two rotors are provided, then (3.8) non-deterministically enforces at least two of them to be equal.

There have been several proposals to extend OWL 2 with integrity constraints [83, 84]. In these approaches, the ontology developer explicitly designates a subset of the OWL 2 axioms as constraints. Similarly to constraints in databases, these axioms are used as checks over the given data and do not participate in query answering once the data has been validated. The specifics of how this is accomplished semantically differ amongst each of the proposals; however, all approaches largely coincide if the standard axioms are in OWL 2 QL.

Data Validation and Query Answering

Our approach to data validation and query answering follows the standard approaches in the literature [88, 84]: given a query Q, dataset $\mathcal{D}$, and OWL 2 ontology $\mathcal{O}$ consisting of a set $\mathcal{R}$ of standard OWL 2 QL axioms and a set ρ of axioms marked as constraints, we proceed according to Steps 1–4 given next.

1. Translate the standard axioms $\mathcal{R}$ into a Datalog program $\Pi_{\mathcal{R}}$ using the well-known correspondence between OWL 2 QL and Datalog.

2. Translate the integrity constraints ρ into a Datalog program Π_{ρ} with stratified negation-as-failure containing a distinguished binary predicate Violation for recording the individuals and axioms involved in a constraint violation.

3. Retrieve and flag all integrity constraint violations. This can be done by computing the extension of the Violation predicate.

4. If no constraints are violated, answer the user's query Q using the query answering facilities provided by the reasoner.

Steps 3 and 4 can be implemented on top of RDF triple stores with support for OWL 2 QL and stratified negation (e.g., [89]), as well as on top of generic rule inference systems (e.g., [90]). In the remainder of this Section we illustrate Steps 1 and 2, where standard axioms and constraints are translated into rules.

Standard Axioms: Table 3.1 provides the standard OWL 2 QL axioms needed to capture the information models of Section and their translation into negation-free

OWL 2 Axiom	Datalog Rules
SubClassOf$(A\ B)$	$B(x) \leftarrow A(x)$
SubPropertyOf$(P_1\ P_2)$	$P_2(x,y) \leftarrow P_1(x,y)$
TransitiveObjectProperty(P)	$P(x,z) \leftarrow P(x,y) \wedge P(y,z)$
InverseObjectProperties(P_1, P_2)	$P_2(y,x) \leftarrow P_1(x,y)$ and $P_1(y,x) \leftarrow P_2(x,y)$
SubClassOf$(A$ AllValuesFrom$(P\ B))$	$B(y) \leftarrow P(x,y) \wedge A(x)$

Table 3.1: OWL 2 QL axioms as rules. All entities mentioned in the axioms are named. By abuse of notation, we use SubPropertyOf and AllValuesFrom to refer to both their Object and Data versions in functional syntax.

rules. In particular, the axioms (3.1)–(3.5) are equivalent to the following rules:

$$\mathsf{Equipment}(x) \leftarrow \mathsf{Turbine}(x) \tag{3.9}$$
$$\mathsf{hasSpeed}(x,y) \leftarrow \mathsf{hasRotorSpeed}(x,y) \tag{3.10}$$
$$\mathsf{hasPart}(x,z) \leftarrow \mathsf{hasPart}(x,y) \wedge \mathsf{hasPart}(y,z) \tag{3.11}$$
$$\mathsf{Packaging}(y) \leftarrow \mathsf{Conveying}(x) \wedge \mathsf{followedBy}(x,y) \tag{3.12}$$

Constraint Axioms: Table 3.2 provides the constraint axioms required to capture the models of Section 3.3 together with their translation into rules with negation. Our translation assigns a unique id to each individual axiom marked as an integrity constraint in the ontology, and it introduces predicates not occurring in the ontology in the heads of all rules. Constraint violations are recorded using the fresh predicate Violation relating individuals to constraint axiom ids.

The constraint (3.6) from Section 3.2.2 is captured by the rules:

$$\mathsf{hasPart\_Rotor}(x) \leftarrow \mathsf{hasPart}(x,y) \ \wedge \ \mathsf{Rotor}(y) \tag{3.13}$$
$$\mathsf{Violation}(x,\alpha) \leftarrow \mathsf{Turbine}(x) \wedge \mathbf{not}\ \mathsf{hasPart\_Rotor}(x) \tag{3.14}$$

Rule (3.13) identifies all individuals with a rotor as a part, and stores them as instances of the auxiliary predicate hasPart_Rotor. In turn, Rule (3.14) identifies all turbines that are not known to be instances of hasPart_Rotor (i.e., those with no known rotor as a part) and links them to the constraint α they violate.

Integrity constraints based on cardinalities require the use of the OWL 2 equality

predicate For instance, the constraint axiom (3.7) from Section 3.2.2, to which we assign the id β_1, is translated into the following rules:

$$\begin{aligned} \mathsf{hasPart\_2\_Rotor}(x) \leftarrow & \bigwedge_{1 \leq i \leq 2} \big(\mathsf{hasPart}(x, y_i) \wedge \mathsf{Rotor}(y_i)\big) \\ & \wedge (\mathbf{not}\ \mathsf{sameAs}(y_1, y_2)), \\ \mathsf{Violation}(x, \beta_1) \leftarrow & \mathsf{TwoRotorTurbine}(x) \\ & \wedge \mathbf{not}\ \mathsf{hasPart\_2\_Rotor}(x). \end{aligned}$$

The first rule infers an instance of the auxiliary predicate hasPart_2_Rotor if it is connected to two instances of Rotor that are not known to be equal; in turn, the second rule infers that all instances of TwoRotorTurbine that are not known to be instances of the auxiliary predicate violate the constraint (3.7). Similarly, axiom (3.8), to which we assign the id β_2, is translated as follows:

$$\begin{aligned} \mathsf{hasPart\_3\_Rotor}(x) \leftarrow & \bigwedge_{1 \leq i \leq 3} \big(\mathsf{hasPart}(x, y_i) \wedge \mathsf{Rotor}(y_i)\big) \\ & \wedge \bigwedge_{1 \leq i < j \leq 3} (\mathbf{not}\ \mathsf{sameAs}(y_i, y_j)), \\ \mathsf{Violation}(x, \beta_2) \leftarrow & \mathsf{TwoRotorTurbine}(x) \\ & \wedge\ \mathsf{hasPart\_3\_Rotor}(x). \end{aligned}$$

Analogously to the previous case, the first rule infers that an individual is an instance of hasPart_3_Rotor if it is connected to three instances of Rotor that are not known to be equal; in turn, the second rule infers that every such individual that is also an instance of TwoRotorTurbine violates the constraint axiom (3.8).

To conclude this section, we note that our translation in Table 3.2 yields a stratified program for any set $\mathcal{R}$ of constraints. We can always define a stratification where the lowest stratum consists of the predicates in $\mathcal{R}$ and the intermediate stratum contains all predicates of the form R_B, R_n_B, and R_n, and the uppermost stratum contains the special Violation predicate.

3.3 Ontology Descriptions (TechOnto)

Our technical system ontology is built upon systematic concepts generalizable for any complex system of any scale and provides a rationale for delineating technolog-

OWL Axiom	Datalog rules
SubClassOf (A SomeValuesFrom(R B))	$R\_B(x) \leftarrow R(x,y) \ \wedge \ B(y)$ and $Violation(x,\alpha) \leftarrow A(x) \ \wedge \ \textbf{not} \ R\_B(x)$
SubClassOf (A HasValue(R b))	$Violation(x,\alpha) \leftarrow A(x) \ \wedge \ \textbf{not} \ R(x,b)$
FunctionalProperty(R)	$R\_2(x) \leftarrow R(x,y_1) \ \wedge \ R(x,y_2) \ \wedge \ \textbf{not} \ sameAs(y_1,y_2)$ and $Violation(x,\alpha) \leftarrow R\_2(x)$
SubClassOf (A MaxCardinality(n R B))	$R\_B(x) \leftarrow \bigwedge_{1\le i\le n+1} (R(x,y_i) \wedge B(y_i)) \bigwedge_{1\le i<j\le n+1} (\textbf{not} \ sameAs(y_i,y_j))$ and $Violation(x,\alpha) \leftarrow A(x) \ \wedge \ R\_B(x)$
SubClassOf (A MinCardinality(n R B))	$R\_n\_B(x) \leftarrow \bigwedge_{1\le i\le n} (R(x,y_i) \wedge B(y_i)) \bigwedge_{1\le i<j\le n} (\textbf{not} \ sameAs(y_i,y_j))$ and $Violation(x,\alpha) \leftarrow A(x) \ \wedge \ \textbf{not} \ R\_n\_B(x)$

Table 3.2: Constraints axioms as rules. All entities are named, $n \geq 1$, and α is the unique id for the given constraint. *SomeValuesFrom*, *HasValue*, *FunctionalProperty*, *MaxCardinality* and *MinCardinality* denote both their Object and Data versions.

ical systems from other social systems, small or large. Usually, technical structures are coherent and comprise interacting and interconnected components. Primarily it can be viewed as a hierarchy of systems where each system is a large, complex, customized and engineered-intensive product of its own kind to meet the requirements of its customer.

To capture the semantic of such systems for data access and analytics task, many ontologies exist, but not all of them are suitable for reuse. Here a critical knowledge engineering task is to select the representative reference ontologies that are generic enough to unify different domain context and scope. This thesis builds reference ontologies that are adopted for the cross-industry domain of managing data integration and analytics.

Figure 3.1 shows the core reference ontologies that were used on client engagements, which are the subject of this thesis.

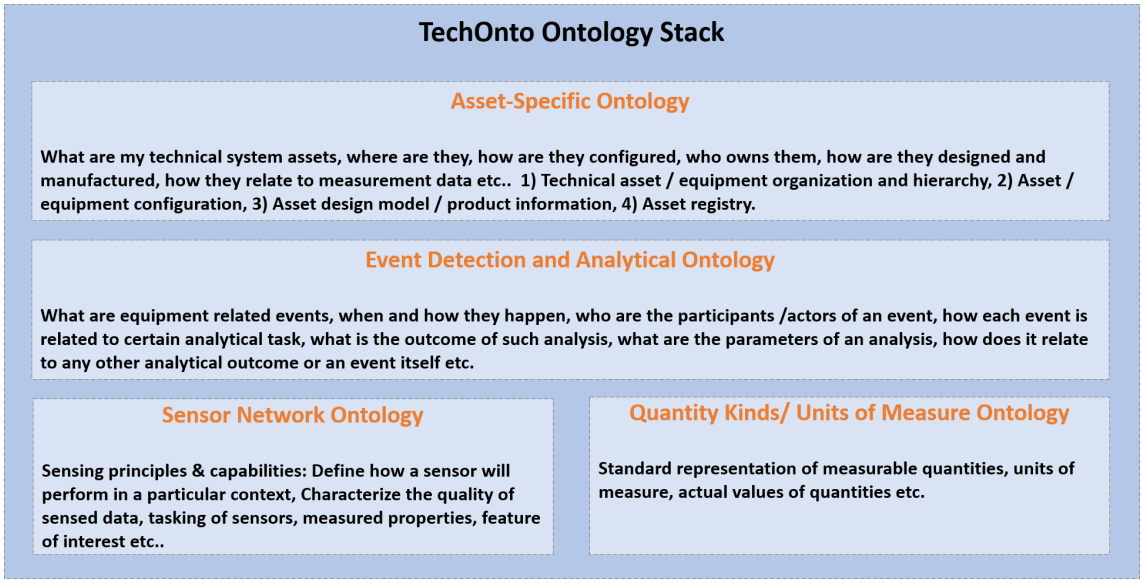

Figure 3.1: TechOnto Ontology stack including reference ontologies.

The solution technical system ontology builds on and extends existing reference ontologies and attempts to reuse knowledge that was developed by domain experts within the solution domain as much as possible. A typical solution requirement is to use the flexibility, extensibility, and openness that is promised by semantic web technologies with open methods for data access. The strategy is to adopt widely used vocabularies and ontologies to define concepts and terms within the solution reference semantic model. To meet these goals, this thesis identifies relevant reference ontologies, including the Semantic Sensor Network (SSN) ontology and the Quantity - Unit - Dimension - Type (QUDT) ontology, and develops Asset-specific domain (ASD) and Event and Analytics ontologies. The solution domain ontology uses, extends, and harmonizes these ontologies, and additionally defines its own concepts, which are not covered by these ontologies.

3.3.1 Domain-specific Ontology Models

Asset-specific Domain Ontology

Asset specific domain ontology is the foundational ontology of TechOnto ontology and defines competency questions like What are my assets, where are they, how are they configured, who owns them, how they relate to measurement data, how they can be analysed for a particular use-case etc. Therefore, the ontology in Figure 3.2 focuses on the following aspects:

- Technical asset/equipment organization and hierarchy
- Asset/equipment configuration

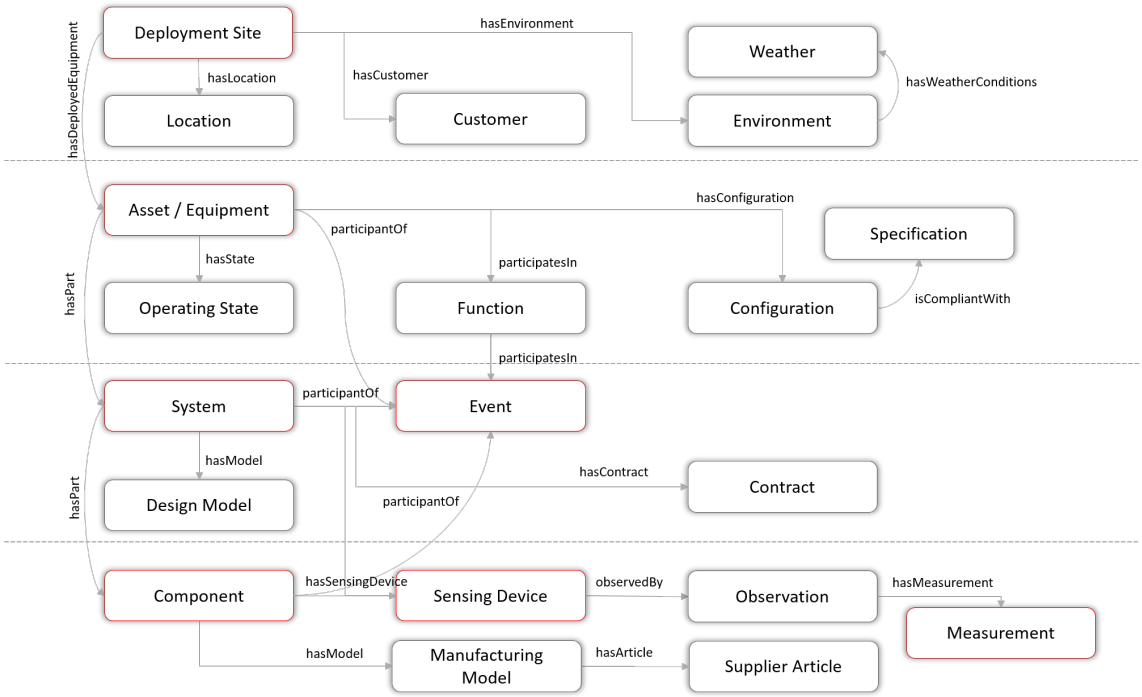

Figure 3.2: TechOnto Ontology - Asset-specific main classes and properties.

- Asset design model / product information
- Asset registry

Technical asset/equipment organizational hierarchy: defines the formal hierarchy and organization of any equipment and mainly follows the following four facets in any type of industrial system.

Deployment: facet describes the whole facility such as the 'Deployment Site' of the asset/ equipment that have been physically deployed. It also defines the asset application, system boundaries, substantial descriptions of the system environment and interaction with external entities such as 'Customers', 'External application interfaces'. Some real-world examples would be of a drive-train, Monticello power plant in Texas, gas power-station in Dresden or wind stations in Baltic Sea.

Our main contribution is the high-level abstraction of the deployment and high-level asset meta-data, especially regarding its internal processes and states. Asset-level processes define the technical functionalities throughout its life-cycle, which are not informative entities and therefore its type must be inferred by the information entities of the system itself e.g. inference of its operational status based on sensor measurements and observation. Another contribution is the knowledge about the plant's deployment. It is represented via geographical objects such as location. Figure 3.2 shows the main classes and relations of this facet.

System: is conceptualized as a system of material-technical artifacts of some materialized action of a specific technical type. Therefore, system module has a central position in the ontology. This facet corresponds to the 'machine level' specifications in the hierarchic level of automation. The concept has been inspired by the 'platform' concept in SSN ontology.

The main contribution of this facet is first to capture system related concepts so that any application for data access, processing or analytics can extract knowledge about the real world. Secondly, the model establishes a direct link between the system and its configuration. We argue that this system configuration (design time or real-time) provides a distinct architecture of interconnected components in a functional chain and is highly important across life-cycle functions. The System concept further breaks down to capture its compositional and functional knowledge.

Component: view is captured by the component facet. Various types of components are operated by the system. Some could be physical while others could be virtual components to fit in the architecture of the system. They can be identified by some component type, manufacturer model and inventory information such as article number, SAP description and/or serial number.

Our main contribution is that part-whole relationship (i.e. component hierarchies) are being captured by an object property *hasPart* which is transitive in nature. Secondly, component actions describe all the events across the lifetime of a component. For example, its installation or removal or inspection of one or more system and plants. This information is also utilized when the design or any activity of one component impacts other components in the same system. For example, a turbine stator is being affected by a failure in its sub-component Stage 3.

Lastly, the most important aspect is the modelling of sensing devices that are mounted on the components to measures various properties for monitoring and analytical tasks. The upper ontology describes the sensors, their properties, measurements and observations by utilizing the SSN ontology. Measurement capabilities have been extended to include measurement property configuration (e.g. set-point values at design time and real-time). We also represent sensor meta-data including its reference designation system tag to infer its location, its sampling method, and data transfer method. Such meta-data is required to make the sensor and its output more meaningful for lookup, discovery and analytical applications.

Function: defines the behavioural semantics of the system and its components. Functions are defined as all interactions that occur during the life-cycle. It provides a multi-dimensional functional view of the complex system as well as translates the functional aspects of any given component or a group of components by its type, location or related processes. Each function attributes

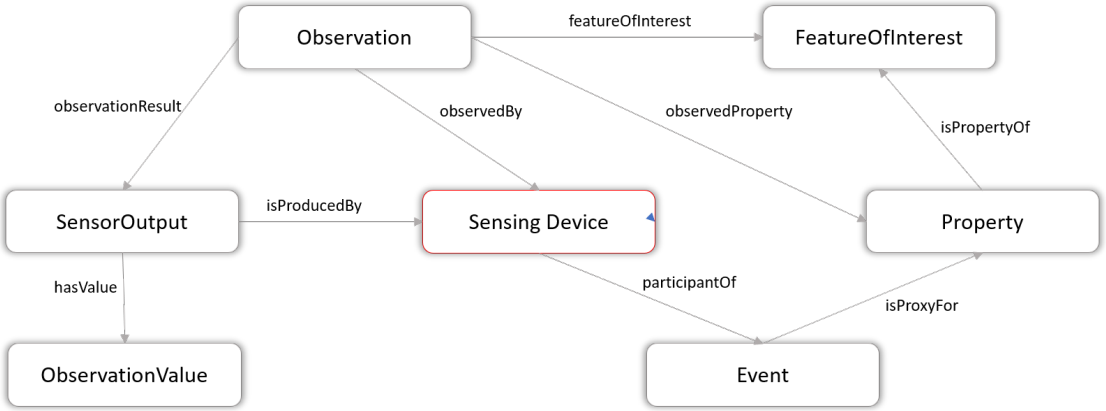

Figure 3.3: TechOnto Ontology - Sensor ontology main classes and properties.

to components, sensors and processes descriptions to support discovery and traceability. For example, failure analysis of a component can be captured together with its sensor measurements and system-level impact. The class *function* can also be realized by failure processes that have temporal bindings, state, and causality.

Sensors and Sensing Ontology

The sensor-specific domain ontology (see figure. 3.3) defines the capabilities of sensors and sensor networks, and covers sensing principles and capabilities, such as the following ones:

- Define how a sensor performs in a particular context
- Characterize the quality of sensed data
- Tasking of sensors

To represent these capabilities, use the Semantic Sensor Network (SSN) ontology (for more information, see http://www.w3.org/2005/Incubator/ssn/ssnx/ssn). The SSN ontology covers the subdomains that are sensor-specific, such as the sensing principles and capabilities, and can be used to define how a sensor performs in a particular context to help characterize the quality of sensed.

Quantity Kinds and Units of Measure Ontology

Quantity kinds (for example, temperature, pressure, and velocity) and units of measure (for example, meter, kilogram, and degree Celsius) reference ontologies provide a standard representation of measurable quantities, units of measure, and actual

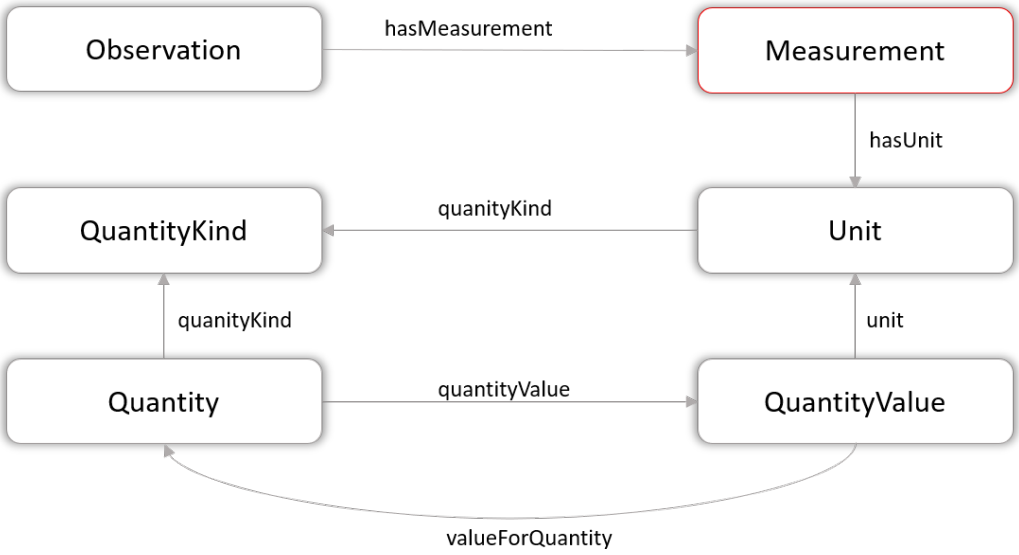

Figure 3.4: TechOnto Ontology - QUDT ontology main classes and properties.

values of quantities. These ontologies are needed to provide a unified model of measurable quantities, units for measuring different kinds of quantities, the numerical values of quantities in different units of measure, and the data structures and data types that are used to store and manipulate these objects in software. The ontology includes instance data populating the model with standard quantities, units, and quantity values. A few reference ontologies for quantity kinds and units of measure exist, each with a different purpose, level of completion, and comprehensiveness. The scope of this paper does not include a comparative study of these ontologies. Rather, this paper focus on the NASA Quantity - Unit - Dimension - Type (QUDT) Ontology (for more information, see http://www.qudt.org). This ontology is by far the most comprehensive and complete regarding quantity kinds and units.

The concepts that are modelled in the quantity kinds, quantity values, and units of measure ontologies are shown in Figure 3.4 and described as follows:

- Quantity Kind is any observable property that can be measured and quantified numerically. Examples include physical properties Line Length, Mass, Time, and Force. Other properties can include Currency, Interest Rate, and Price to Earning Ratio.

- Quantity is an observable property of an object, event, or system that can be measured and quantified numerically. Examples include the mass of a hydrogen atom, the temperature at a certain site, or the duration of a specific meeting. The attributes include (1) Kind identifies the observable property that is quantified, and 2)Magnitude expresses its relative size compared to other quantities of same kind.

- Unit of Measure is a particular quantity of a given kind that is chosen as

a scale for measuring other quantities of the same kind. Examples include Meters, Kilograms, and Volts.

- Quantity Value is the numerical value of a quantity's magnitude with respect to a chosen unit of measure for the corresponding quantity kind. Examples include 5 kilograms or 3 meters.

The QUDT ontology defines the base classes properties and restrictions that are used for modeling physical quantities, units of measure, and their dimensions in various measurement systems. The QUDT ontology is a schema ontology and uses the name space and the prefix qudt for all internally defined resources that are described at http://www.linkedmodel.org/catalog/qudt/1.1/index.html

3.3.2 Analytical Ontology Model

Event detection and diagnostic ontology where event is a focus and starting point to integrate any type of event data including analytics. In principle, Event collects information of an order and/or diagnostic actions up to the Asset level versus the time. An *Event* is the result of the interactions which occur in a certain point of the time between an *actor* and any asset related information (i.e. *EventInformation*). Actor can be field service technician, the repair workshop technician or the control room operator. The *EventInformation* can be anything from a single component which is sent to be repaired or a gas turbine package which is disassembled for doing Level-C inspection. As it is obvious from this definition, an Event can take long for several days but in the current design, only one date is associated to each Event. To be able to verify the Events, each Event must have a reference document. The document can be a report which is written at the end of one preventive or diagnostic Event by an analyst, technician or a component repair report which is written by the repair workshop technician. These reports are the main source of information for analytical task. This ontology also defines special type of analytical events such as diagnostic event which present results from an analytical task executed over a set of data and results are recorded as relevant events. This motivates the reuse and combination of different diagnostic task and promotes interoperability of its execution. Details on the specification and formalism of analytical ontology are described in the following chapter.

3.3.3 Ontology Summary

Many industries today realize and demand improvements in their overall working life-cycle processes in order to meet their commercial as well as safety and security constraints. This ultimately demands for integration across the board i.e. existing information systems, applications and adopted standards. However, there exist a

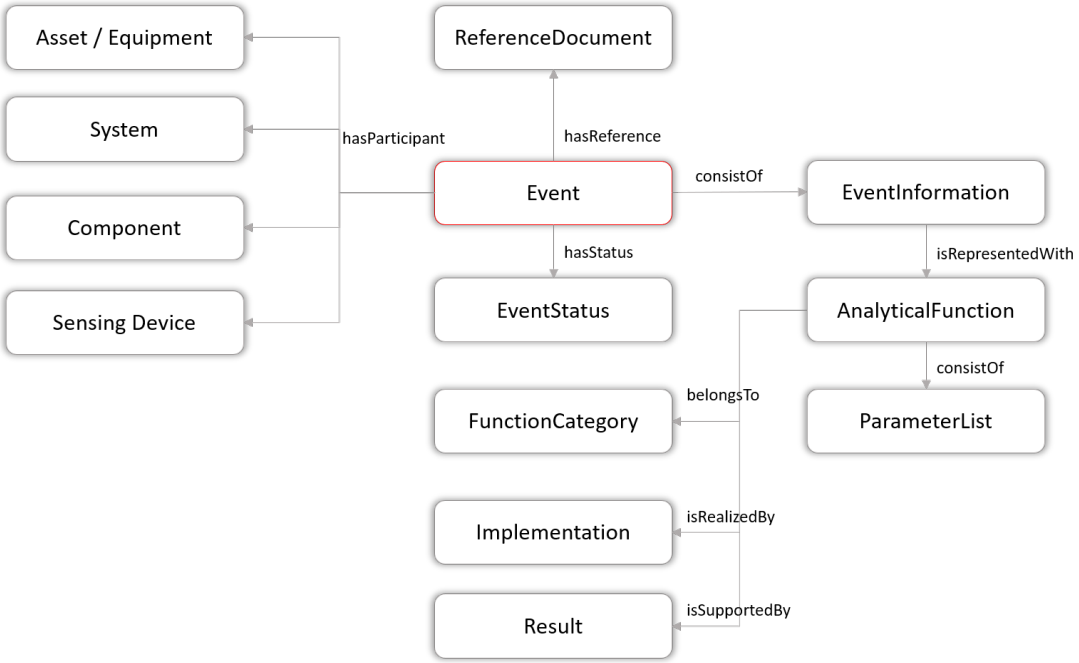

Figure 3.5: TechOnto Ontology - Event and Analytical ontology main classes and properties.

number of challenges which are directly related to a high cost of integrating incompatible proprietary representations of information. Nevertheless, with the advancement in technology, shared ontologies provide an easier approach to cater such challenges of integration and information access. For example, adopting domain ontologies that can define and create abstract information base for concepts such as physical objects, activities, meteorological and topological relations can be beneficial in integrating heterogeneous information. Furthermore, this abstraction layer can be extended to more specific classes and relations depending on the granularity of the information exchange.

Many ontologies exist, but not all of them are suitable for reuse. A critical knowledge engineering task is to select the representative reference ontologies that are appropriate to individual domain and scope. This thesis presents the domain together with reference ontologies that are selected (adopted) for the cross-industry domain of managing observations and measurements, topologies, events and analytics. Our proposed ontology contain four modules that also cater the two existing reference ontologies. Asset-specific domain model describes the physical and virtual concepts of an industrial equipment or component together with its component hierarchies, design models, configuration and registry informations. Observations and sensor measurement related data is conceptualized using SSN reference ontology whereas quality standards are expressed via qudt ontologies. An important module is Event and Prediction ontology that describes various event-driven information and actions that occur on an equipment. Special type of event are called analytical event that captures all the meta-data generated during an execution of an analytical task or

workflow. Practical applications and evaluations of the model against state-of-the-art are discussed in Chapter 6.

3.4 Ontology Model Manager (SOMM)

We have developed the Semantic Ontology Model Manager (SOMM) tool to support domain experts little or no background on semantic technologies in building ontologies and inserting data based on their information models. The interface of SOMM is restricted to support only the kinds of standard OWL 2 QL axioms and constraints discussed in Section 3.2. SOMM is built on top of the Web-Protege platform [91] by extending its front-end with new visual components and its back-end to access RDFox [92] for query answering and constraint validation, HermiT [88] for ontology classification, and LogMap [93] to support ontology alignment and merging. Our choice of WebProtege was based on requirements and CQs of the experts for the platform underpinning SOMM, namely that it (i) can be used as a Web application; (ii) is under active development; (iii) is open-source and modular; (iv) includes built-in functionality for ontology versioning and collaborative development; (v) provides a form-based and end-user oriented interface; and (vi) enables the automatic generation of forms to insert instance data. Although we considered other alternatives such as Protege- desktop [87], NeON toolkit [94], OBO-Edit [95], and TopBraid Composer [96], we found that only WebProtege satisfied all the aforementioned requirements. In the remainder of this section, we describe the main features of SOMM.

3.4.1 Form-based insertion of axioms

We have implemented a new form-based editor to attach properties to a class via existential, universal, cardinality and value restrictions. This visual component aims at supporting engineers with little background on semantic technologies in the creation and interpretation of the most common ontological axioms to capture the semantics of the models in Siemens. Figure 3.6 shows a screenshot of the SOMM class editor where the class SteamTurbine has four properties attached; for example the first row represents both an universal (default attachment) and existential restriction since attribute hasState is required (axioms (3.15) and (3.16)) while the second row is translated into an universal restriction and two cardinality restrictions

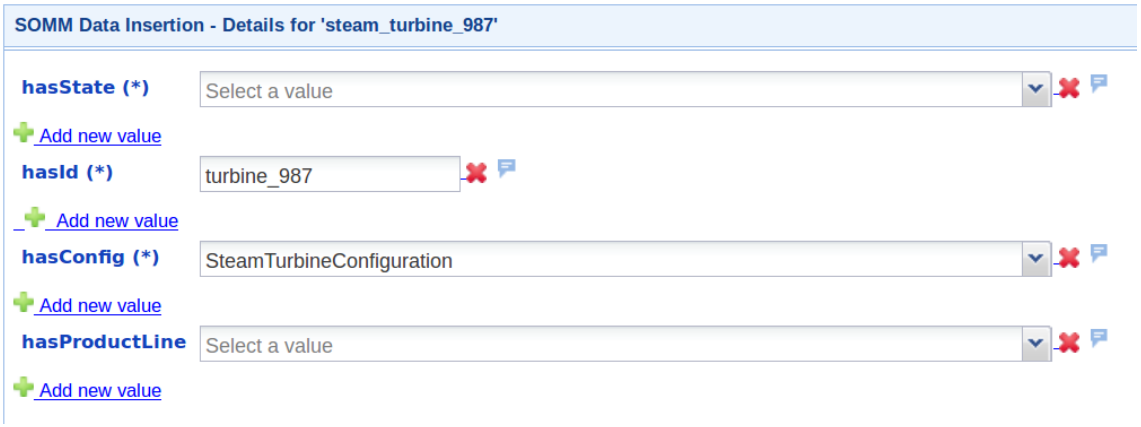

Figure 3.6: Data insertion in SOMM.

(axioms (3.17)-(3.19)).

$$\text{SubClassOf}(\text{SteamTurbine} \quad \text{ObjectSomeValuesFrom}(\text{hasState} \quad \text{State})) \tag{3.15}$$

$$\text{SubClassOf}(\text{SteamTurbine} \quad \text{ObjectAllValuesFrom}(\text{hasState} \quad \text{State})) \tag{3.16}$$

$$\text{SubClassOf}(\text{SteamTurbine} \quad \text{ObjectMinCardinality}(1 \quad \text{hasConfig} \quad \text{SteamTurbineConfig})) \tag{3.17}$$

$$\text{SubClassOf}(\text{SteamTurbine} \quad \text{ObjectMaxCardinality}(3 \quad \text{hasConfig} \quad \text{SteamTurbineConfig})) \tag{3.18}$$

$$\text{SubClassOf}(\text{SteamTurbine} \quad \text{ObjectAllValuesFrom}(\text{hasConfig} \quad \text{SteamTurbineConfig})) \tag{3.19}$$

3.4.2 Automatically generated data forms

SOMM exploits the Web-Protégé capabilities to generate knowledge acquisition forms to guide engineers during the data insertion process. The forms are automatically generated for each class driven by the axioms in the ontology, that is, SOMM generates an entry field for each of the properties attached to a class. SOMM not only considers directly attached properties but also inherited properties and *bottom-up* propagated properties. For example, the class Turbine does not have directly attached properties in one of our ontologies, however SOMM will suggest the attached properties of its subclasses (e.g. the ones for SteamTurbine). Figure 3.6 shows an example of the property fields for an instance of the class SteamTurbine, which has four attached properties according to Figure 3.6. Note that SOMM differentiates between required (marked with (*)) and optional fields.

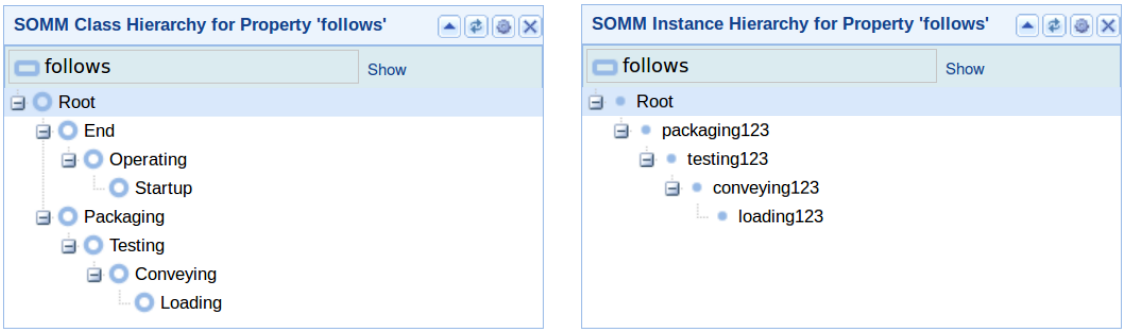

Figure 3.7: Tree-like navigation of the ontology classes and individuals in SOMM.

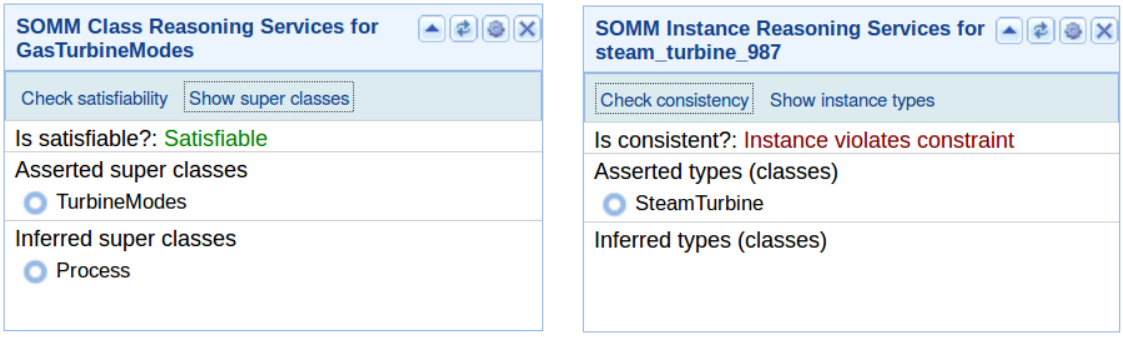

Figure 3.8: Reasoning services for ontology classes and individuals in SOMM.

3.4.3 Extended tree-like navigation of classes and individuals

SOMM also allows a tree-like navigation of the ontology classes and individuals according to a selected property. This visual components is a generalization of the well-known partonomy hierarchies for which we do not necessarily require a part whole relationship. SOMM exploits the attachment of properties to classes, inverse roles and role assertion axioms to built the navigation trees. Figure 3.7 shows the classes and individuals tree for the property follows, which defines dependencies among processes.

3.4.4 Ontology Alignment

SOMM integrates the ontology alignment system LogMap [93] to support model alignment and merging. SOMM allows to select and merge two available Web-Protégé projects or to import and merge an ontology into the active Web-Protégé project. Currently LogMap runs in an automatic mode, but we plan to extend SOMM interface to support user-interaction in the alignment process, as LogMap includes built-in interactive matching capabilities [97].

3.4.5 Reasoning services

Web-Protégé does not currently includes reasoning services in its default distribution. SOMM relies on the OWL 2 reasoner HermiT [88] to support standard reasoning services such as class consistency and ontology classification. The datalog reasoner RDFOx [92] is used to perform the constraint validation task. As discussed in Section 3.2.2, for data reasoning purposes, ontology axioms are split into two sets: inference axioms and integrity constraint axioms, and translated into datalog rules. SOMM implements the approach presented at the end of section 3.2.2 to check for integrity constraints violations. Figure 3.8 shows the SOMM intefaces to invoke the supported reasoning services. For example, Figure 3.8 shows that the class GasTurbineModes is satisfiable and Process is an inferred superclass after performing reasoning, while Figure 3.8 states that the individual steam_turbine_987 violates one of the integrity constraints. If we recall the example in Figure 3.6, steam_turbine_987 is missing the attribute hasState which is required for all steam turbines (see Figure 3.6). According to the translation into rules provided in row α_2 in Table 3.2 steam_turbine_987 is *not* in has_1_hasState_State and hence deriving $Violation(\text{steam\_turbine\_987}, \alpha_2)$, where $\alpha_2 = \text{SteamTurbine SubClassOf : hasState some State}$.

4 Ontology Language for Semantically driven Analytical Tasks

In this chapter we explore the use of ontology language to semantically define data analysis tasks on Industrial data. This results in proposing a new approach for capturing semantics of analytical tasks, algorithms, inputs and outputs and evaluate the effectiveness of the proposed approach on a number of analytical tasks on real industrial use-cases. We conclude that semantically defined methods that considers analytical concepts produce, in most cases, a higher interoperability, scalability and performance than those state-of-the-art.

The material in this chapter has been published in [98, 99, 100].

4.1 Introduction

Traditional approaches to data analytics on industrial data have three main limitations as discussed in previous chapters. Firstly, the number of queries to extract data for analysis is finite. This means that a domain expert is often dependent on the IT expert to formulate such queries for him. This limits the data exploration capabilities of the user and may restrict data integration capabilities from heterogeneous data infrastructures such as when new databases, new columns, new meta-data emerge. Secondly and more importantly, data analysis implementations and components are highly data dependent in the sense that specific characteristic of individual data points such as sensors and pieces of equipment are explicitly encoded in the code. As a result for a typical diagnostic task an engineer has to configure and run dozens to hundreds of such models with some or little modification in the attributes such as sensor tags, component codes, sensor and threshold values, equipment configuration or design attributes etc. For example, a typical gas turbine has about 2000 sensors and a diagnostic task to detect whether purging is over can be captured with over 30 analytical models (e.g. a predictive model). Many of these models may differ only on specific sensor identifiers or the number of data patterns and speed signals to aggregate. Adapting these models to another equipment will also require the basic understanding of the type and configuration of the equipment type and consequently may require a change of data source and

corresponding identifiers. Third and the most interesting by-product of the data dependency in analytical models is the challenge of authoring, reuse and maintenance of analytical models and their results. Most often than not, analytical workflows are static, use-case driven and have limited re-usability. For example, models that compute performance degradation key performance indicators (KPI) cannot be easily reused to capture reliability KPIs. Details on the analytical workflow are discussed in Chapter 5.

In this chapter, we present our results of extracting and using the semantically defined analytical language aiming mainly at addressing the challenges faced by current state-of-the-art and consequently improving their performances. This chapter addresses the following research question:

[RQ2]Can an analytical-aware ontology language for analytical tasks enhance data analysis performance?

To this end, we propose to extend the traditional data driven approach to analytics with an OBDA layer and a new semantic language to what we call textitSemantically defined Analytical Language (SAL). Our proposed language enjoys the following features:

- Signals orientation: The language should treat signals as first class citizens and allow for their manipulation: to filter, aggregate, combine, and compare signals;
- Expressiveness: The language should capture most of the features of the rule models as well as analytical operation.
- Usability: The language should be simple and concise enough so that the engineers can significantly save time in specifying analytical tasks;
- Efficiency: The language should allow for efficient execution of analytical tasks and must be FO rewritable.

Our language allows to write complex analytical 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 analysis features and, on the other hand, it has good computational properties. In particular, *SAL* allows for rewriting [23] of analytical task written over OWL 2 QL ontologies into multiple data-dependent rule-sets with the help of ontologies and OBDA mappings. We implemented *SAL* and a prototypical Semantically defined Analytics system. We evaluated usability of our solution with engineers at Siemens by checking how fast they are in formulating analytical tasks in *SAL*. We also evaluated the efficiency of our solution in processing analytical tasks over turbine signals in a controlled environment. (See chapter 7 for use-case evaluations.)

In rest of this chapter we introduce our language and define its basic components, then we specify the details of capturing analytical expressions and their semantics. Lastly, we define the formal properties of the language.

4.2 Building blocks of Proposed Language

We start by introducing basic notation for our proposed ontology language. In particular, we introduce notions for (i) (sensor) signals and (ii) *Knowledge Bases* (KBs). The former we use to capture the sensor data-points over time and the latter we use to capture background knowledge of equipment and signals as well as concrete characteristics of the equipment that undergoes analysis. Both signals and KBs are building blocks of our semantic language *SAL* (defined in the next section).

4.2.1 Sensor Signals

In our setting, a *signal* is the foundation for our language. A signal s is a pair (o_s, f_s) of a *signal id* o_s and a *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 whose reading, such as temperature, is obtained from a single sensor (e.g., in a train) 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, i.e., $\perp \notin f_s(I)$. 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 $\mathcal{S} = \{s_1, \ldots, s_n\}$.

4.2.2 Knowledge Bases and Queries

A Knowledge Base $\mathcal{K}$ is a pair of an *ontology* $\mathcal{O}$ and a *data set* $\mathcal{D}$. An *ontology* describes background knowledge of an application domain in a formal language. We refer the reader to [23] for detailed definitions of ontologies. In our setting, we consider ontologies (as described in chapter 3) that describe general characteristics of 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 DoorSensor is a kind of PressureSensor:

$$\mathsf{SubClassOf}(\mathsf{DoorSensor}\ \ \mathsf{PressureSensor}). \tag{4.1}$$

Data sets of KBs consist of *data assertions* enumerating concrete sensors, equipments, and their components. The following assertions says that sensors SKNF_X01,SKNF_X02, SKNF_X03 and SKNF_X04 are all door sensors:

$$\begin{aligned}&\mathsf{ClassAssertion}(\mathsf{DoorSensor}\ \ \mathsf{SKNF\_X01}),\\ &\mathsf{ClassAssertion}(\mathsf{DoorSensor}\ \ \mathsf{SKNF\_X02}),\\ &\mathsf{ClassAssertion}(\mathsf{DoorSensor}\ \ \mathsf{SKNF\_X03}),\\ &\mathsf{ClassAssertion}(\mathsf{DoorSensor}\ \ \mathsf{SKNF\_X04}).\end{aligned} \tag{4.2}$$

In order to enjoy favourable semantic and computational characteristics of OBDA, we consider well-studied ontology language OWL 2 QL that allows to express sub-class (resp. sub-property) axioms between classes and projections of properties (resp. corollary between properties).

A formal basis for OWL 2 QL is *DL-Lite*$_R$ [23]. Here, we briefly introduce the main construct of *DL-Lite*$_R$ and the main reasoning tasks, query answering. For more details on the language we refer to [23].

In *DL-Lite*$_R$ concepts and roles are of the following form:

$$\begin{aligned} B &::= A \mid \exists R, & \qquad C &::= B \mid \neg B,\\ R &::= P \mid P^-, & \qquad E &::= R \mid \neg R \end{aligned}$$

where A denotes an atomic concept, P an atomic role, and P^- the inverse of P. Further, B denotes a basic concept (i.e., an atomic concept A or an unqualified existential quantification on a basic role $\exists R$) and R a basic role (i.e., an atomic role P or its inverse). Finally, C denotes a general concept (i.e., a basic concept or its negation) and E a general role (i.e., a basic role or its negation).

A *DL-Lite*$_R$ Tbox (or ontology) is a finite set of inclusion statements of the form $B \sqsubseteq C$ or $R \sqsubseteq E$.

A *DL-Lite*$_R$ Abox consists of a finite set of membership assertions on atomic concepts and roles of the form $A(a)$ and $P(a, b)$.

In general, Tbox is known as a *terminological component* that describes set of concepts and their properties, whereas Abox is an *assertion component* that describes a fact associated with concept within a knowledge base.

A *DL-Lite*$_R$ KB $\mathcal{K} = (\mathcal{O}, \mathcal{A})$ is a pair of a TBox $\mathcal{O}$ and an ABox $\mathcal{A}$. This means that Tbox and Abox together makes up a knowledge base.

The formal interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ of *DL-Lite*$_R$ is the standard First order logic interpretation where Δ is the domain and $\cdot^{\mathcal{I}}$ is the interpretation function.

DL-Lite$_R$ has favourable computational properties of answering unions of conjunctive queries (CQs) under the *certain answers semantics* [23]. It is based on the concept of certain answers, that is the answers that hold over all interpretations. Under this semantics the answer set of a non Boolean CQ $q(x)$ over a KB $\mathcal{K}$ is defined as follows:

$$ans(q, \mathcal{K}) = \{t \in \mathcal{C} \mid \mathcal{K} \models q(t)\},$$

where $\mathcal{C}$ is the set of the constants appearing in the KB, and $q(t)$ is the closed formula obtained by replacing in the query definition the free variables in x by the constants in t.

For example, the following union of CQs returns all main car sensors:

$$\begin{aligned} &\mathsf{MainCarDoors}(x) \leftarrow \mathsf{doorSensor}(x) \wedge \mathsf{locatedIn}(\mathsf{x}, \mathsf{y}) \wedge \\ &\quad (\mathsf{PlatformAccessArea}(y) \vee \mathsf{CabinAccessArea}(y)). \end{aligned} \tag{4.3}$$

To be precise, the above contains disjunction in the body thus it can be represented as a union of two CQs.

4.3 Semantically defined Analytical Language *SAL*

In this section, we introduce formally the syntax and semantics of our semantically defined analytical language *SAL*. To do so, we first introduce analytical expressions that allow one to manipulate basic signals using mathematical functions and queries over KBs. Then in following chapter we introduce a notion of workflow that allow one to compose and combine expressions, and to send desired alert messages. Finally, we provide semantics of our language that formally defines how *SAL* should be executed.

4.3.1 Analytical Expressions

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

$$\begin{array}{rcll} C & = & Q & \mid \{s_1, \ldots, s_m\} \mid \\ & & \alpha \circ C & \mid C_1 : value(\odot, \alpha) \mid \\ & & \mathsf{agg}\ C_1 & \mid C_1 : duration(\odot, t) \mid \\ & & C_1 :\ align\ C_2 & \mid C_1 : trend(direction) \mid \\ & & C_1 : forecast(\alpha). & \end{array}$$

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

Expressions $C = Q$ and $C = \{s_1, \ldots, s_m\}$ we call *basic analytical expressions* and other we call *complex analytical expressions*.

The formal meaning of analytical expressions is defined in Figure 4.1. In order to make the mathematics right, we assume that $c \circ \bot = \bot \circ c = \bot$ and $c \odot \bot = \bot \odot c = false$ for $c \in \mathbb{R}$, and analogously we assume for aggregate functions. If the value of a analytical function at a time point is not defined with these rules, then we define it as $\bot$.

Example 1. *The data-driven analytical rules that can be used to determine that car doors function well, can be expressed with two concepts in SAL as follows:*

$$\mathsf{DoorsLocked} = sum\ \mathsf{MainCarDoors} : value(=, LockedValue), \tag{4.4}$$

$$\mathsf{PressureUp} = \mathsf{CabinPressure} : trend(\text{'up'}) : duration(>, 33sec) \tag{4.5}$$

Here, MainCarDoors *is the CQ defined in Equation* (4.3)*. For brevity we do not introduce a new concept for each expression but we just join them with symbol ":". The constant LockedValue is a parameter of for analysing door of a train, and they are instantiated from the train configuration when the expressions are evaluated.* △

Now we are going to define the semantics of the analytical expressions.

4.3.2 Semantics of *SAL*

We now define how to determine whether these analytical expressions are FO-rewritable and is well-suited in OBDA setting. 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 signal ids (basic or ones defined by expressions). Thus, each object o is also has an assigned function f_s that represents the signal value of that object. Observe that o can also be an id of a train 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 $f_\bot$ 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 $\mathcal{I}_{\mathcal{S}}$—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, $s^{\mathcal{I}_{\mathcal{S}}} = s$.

$C =$	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 $f_{s'} = \alpha \circ f_s$.
$C_1 : value(\odot, \alpha)$	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 $f_{s'}(t) = \perp$.
$C_1 : duration(\odot, t')$	one signal s' for each signal s in C_1 with $f_{s'}(t) = f_s(t)$ if exists an interval I s.t.: f_s is defined I, $t \in I$ and $size(I) \odot t'$; otherwise $f_{s'}(t) = \perp$.
$\{s_1, \ldots, s_m\}$	all enumerated signal $\{s_1, \ldots, s_m\}$.
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.
$C_1 : align\ C_2$	a signal s_1 from C_1 if: exists a signal s_2 from C_2 that is *aligned* with s_1, i.e., for each 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.
$C_1 : trend(direction)$	one signal s' for each signal s in C_1 with $f_{s'}(t) = f_s(t)$ if exists an interval I around t s.t.: f_s is defined I, and f_s is an increasing or decreasing function on I for $direction$=*up* (=*down* resp.)
$C_1 : forecast(\alpha)$	one signal s' for signal s in C_1 with $f_{s'}(t') = f_s(t)$ that is, s' is obtained in forecast interval I' s.t. f_s is defined in I, and f_s is a regression function on I for a given number of observations.

Figure 4.1: Meaning of analytical 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".

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 analytical expressions and we do it recursively following the definitions in Figure 4.1. 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.

4.4 Formal Properties of *SAL*

In this part, we study the formal properties for our *SAL* language.

First, we assume from now that data signal functions are given on the input as a step functions over intervals (which is the case in the running example).

Second, we would like to ensure that our language in *SAL* indeed allow to be rewritten using OBDA techniques. This condition requires more technical explanation and we discuss it in more detail in the next paragraph.

Third, we want to understand what is the upper bound of the complexity of our problem. In particular, we measure the complexity of the problem in the size of two main components: the size of workflow/program and data. We expect that size of the data largely dominates the size of workflow, and while the data can be huge (several GBs) that size of program can be significantly big (several thousands of rules) thus both measures are relevant.

First Order (FO) rewritability The formal condition that determines if a language is suitable for OBDA is called First Order (FO) rewritability. In this part we define it formally. An OBDA setting is a triple $(\mathcal{O}, \mathcal{S}, \mathcal{M})$ where $\mathcal{O}$ in the intensional level of an ontology, $\mathcal{S}$ is a relational schema representing the schema of sources and $\mathcal{M}$ is a set of mapping assertions that describe how to populate ontology with the database. Typically, mappings are are select-project-join SQL queries over sources that describe how ontology is populated. We say that query answering over a setting $(\mathcal{O}, \mathcal{S}, \mathcal{M})$ is *FO-rewritable* if for each query q over $\mathcal{O}$ there exists a FO query (i.e., an linear algebra query) q' over $\mathcal{S}$ such that for any database D over S we have that evaluating q over $\mathcal{O}$ (populated via $\mathcal{M}$ for D) gives the same result as evaluating q' over D only. In other words, we are reducing answering queries over ontologies into answering queries over sources. It is expected that query q' is more complex than q since it has to take into account $\mathcal{M}$ and $\mathcal{T}$.

It is known (e.g., see [101]) that if query answering in some formal language is

FO-rewritable, then the data complexity of the problem is in AC^0 computational complexity [102]. Data complexity of a problem is the computational complexity of the problem when all parameters apart from the data are fixed. Class AC^0 is "weaker" than polynomial time and even linear time, and it is represents a class of problems that are highly parallelizable. Intuitively, a problem is in AC^0 if it can be decided in constant time when the number of processors corresponds to the size of data. A standard way to show that a problem is in AC^0 is to reduce the problem to another problem for which it is already known to have AC^0 in data complexity. For example, data complexity of checking whether a tuple is the answer of a non-recursive Datalog query over a database is in AC^0 in the size of the database.

Hence, to address both problems from above, we encode our *SAL* into fact-entailment problem over an extended version of recently introduced non-recursive metric Datalog [103], Datalog$_{\text{nr}}$*MTL* . The reason for doing this is twofold. First, Datalog$_{\text{nr}}$*MTL* (inspired by a well-studied Metric Temporal Logic [104]) provides a natural way to model rules that reason over time intervals. Second, Datalog$_{\text{nr}}$*MTL* is a suitable language for OBDA setting, that is, it has been show how to rewrite queries over the rules in Datalog$_{\text{nr}}$*MTL* into standard SQL over the sources [103].

Still, Datalog$_{\text{nr}}$*MTL* cannot be immediately related to our language since it does not support aggregates and some other logic constructs that we need for our encoding (in particular, functional symbols, negation and aggregates [102]). So first, we extend DatalogMTL with functional symbols, aggregation, etc. under reasonable restrictions, without increasing the complexity. Then to show that our problem is FO-rewritable we do the semantic workflow encoding described in Chapter 5.

4.4.1 Extended Datalog*MTL*

In this part we introduce our extension of Datalog*MTL*. At this moment we only briefly introduce the main constructs of the language.

An atom A in extended Datalog*MTL* is either a comparison (e.g., $\tau \leq \tau'$) or defined by the grammar

$$\begin{aligned} A \;::=\; & P(\tau_1, \ldots, \tau_m) \;\mid\; \top \;\mid\; \boxplus_\varrho A \;\mid\; \boxminus_\varrho A \;\mid \\ & \diamondplus_\varrho A \;\mid\; \diamondminus_\varrho A \;\mid\; A \,\mathcal{U}_\varrho\, A' \;\mid\; A \,\mathcal{S}_\varrho\, A' \;\mid \\ & \neg A \;\mid\; \tau = \mathsf{agg}[\![\tau_i \mid P(\tau_1, \ldots, \tau_m)]\!] \end{aligned}$$

Here, P is a predicate, ϱ is an interval in reals, τ is a term (possibly with functional symbols), $\mathsf{agg} \in \{\mathsf{min}, \mathsf{max}, \mathsf{avg}, \mathsf{sum}, \mathsf{count}\}$ and brackets $[\![\cdot]\!]$ denote multiset (values can repeat).

A *datalogMTL program*, Σ, is a finite set of *rules* of the form

$$A^+ \leftarrow A_1 \wedge \cdots \wedge A_k \qquad \text{or} \qquad \bot \leftarrow A_1 \wedge \cdots \wedge A_k,$$

where A^+ is an atom that *does not* contain any 'non-deterministic' operators $\diamondplus_\varrho$, $\diamondminus_\varrho$, $\mathcal{U}_\varrho$, $\mathcal{S}_\varrho$, negated atoms, or aggregate operators.

For our purposes it is sufficient to have non-recursive programs. Informally, that are programs where dependency (direct or indirect) between predicates is acyclic. In fact, it is not trivial to understand how one would even define recursion in case of aggregates and negation. Therefore, we only consider extended DatalogMTL program that are non-recursive.

In DatalogMTL, temporal operators are defined over intervals and they take the form $\boxplus_\varrho$, $\diamondplus_\varrho$ and $\mathcal{U}_\varrho$, which refer to the future, and $\boxminus_\varrho$, $\diamondminus_\varrho$ and $\mathcal{S}_\varrho$, which refer to the past where ϱ is an interval. For example, $\boxplus_\varrho A$ is true at t iff an atom A is true in all points of an interval ϱ in the future from t, while $\diamondminus_\varrho A$ is true at t iff there exists a point in the past not longer than ϱ from t where A is true. For the complete semantics of the temporal operators and rules.

A (temporal) *data instance* is a finite set of *facts* of the form $P(c)@\iota$, where $P(c)$ is a ground atom and ι an interval. The fact $P(c)@\iota$ states that $P(c)$ holds throughout the interval ι. Moreover, we simply write $P(c)@t$ for $P(c)@[t,t]$.

Finally, every satisfiable DatalogMTL Σ program with database $\mathcal{D}$ has the *canonical* (or *minimal*) *model of* $\models$ *and* $\mathcal{D}$, $\mathfrak{M}_{\Pi,\mathcal{D}}$. As usual, the most important property of canonical model is that if a fact holds in canonical model then it holds in any other model.

4.4.2 An Example Encoding into Extended Datalog*MTL*

We start with an example of the encoding for analytical expressions. In the next chapter we present the complete encoding including workflows.

Example 2 (Example of Encoding)**.** *The query in analytical expression in Examples 1 and 4 can be encoded in a modular way, starting from simpler to more complex expressions.*

We start with the encoding rules for Example 1.

First, we show how to capture the expression "sum MainCarDoors*" in* (4.4)*. For that we use the following rule:*

$$\begin{aligned}&\mathsf{SumMainCarDoors}(c_{ar}), \mathsf{value}(c_{ar}, v_1) \leftarrow \\ &\quad sum[\![v \mid \mathsf{MainCarDoors}(x), \mathsf{value}(x, v)]\!] = v_1\end{aligned}$$

Intuitively, this introduces a new constant c_{ar} representing the "aggregated main car door sensor" and assign the average value of all main car door *sensors to it.*

Then, we encode the second part of (4.4), $value(=, LockedValue)$, *using* SumMainCarDoors *with the rule:*

$$\begin{aligned}\mathsf{DoorsLocked}(x) \leftarrow\ & \mathsf{SumMainCarDoors}(x),\\ & \mathsf{value}(x, v), v = \mathsf{LockedValue}\end{aligned}$$

To encode expression (4.5) *from Example 1 we need to use temporal operators. In particular, to encode* CabinPressure : $trend(\text{'up'})$ *we need to copy all intervals of a signal in* CabinPressure *on which the signal is trending up. For that we need universal quantification ("*$\forall$*"). This is expressible in Datalog by two rules connected with a negation. First we compute intervals on which a signal is not trending up with the rule:*

$$\begin{aligned}notTrendUp_{\mathsf{CP}}(x) \leftarrow\ & \mathsf{CabinPressure}(x), value(x, v_1),\\ & \diamondsuit_{(0,\delta]}(value(x, v_2), v_1 > v_2)\end{aligned}$$

Intuitively, formula $(value(x, v_1), \diamondsuit_{(0,\delta]}(value(x, v_2), v_1 > v_2))$ *evaluates to true for some value* v_1 *at a time point* t *if there exists an interval of a size at most* δ *containing* t *in which signal* x *has another value* v_2 *that is smaller than* v_1*. Here, a parameter* δ *is a "small" real number and it is typically selected based on the size of signal sampling.*

Then we compute the trending-up intervals by eliminating non-trending-up time points:

$$\begin{aligned}\mathsf{CabinPressureAux}(f_{\mathsf{cp}}(x)) \leftarrow\ & \mathsf{CabinPressure}(x),\\ & \neg notTrendUp_{\mathsf{CP}}(x)\end{aligned}$$

Here, functional symbol f_{cp} *is used to create a new signal identifier for each* x*. The values of the new signals are the same as originals and they are just copied for each time-point that is "trending up":*

$$value(f_{\mathsf{cp}}(x), v) \leftarrow \mathsf{CabinPressureAux}(f_{\mathsf{cp}}(x)), value(x, v)$$

To encode the construct duration we also need temporal operators. In particular, we encode construct : $duration(>, 33sec)$ *with the rule:*

$$\mathsf{PressureUp}(f_{\mathsf{pu}}(x)) \leftarrow \diamondsuit_{[0,33s]} \boxplus_{[0,33s]} \mathsf{CabinPressureAux}(x)$$

Intuitively, the temporal operator $\boxminus_{[0,33s]}$ *selects "an event that lasts for the last 33s", and the temporal operator* $\diamondsuit_{[0,33s]}$*, selects "an event happens within the last 33s". The nested these two* $\diamondsuit_{[0,33s]}\boxminus_{[0,33s]}$ *selects the whole duration of all the events lasting at least 33s.*

Similarly as above, the value is transferred with the rule:

$$value(f_{\mathsf{pu}}(x), v) \leftarrow \mathsf{PressureUp}(f_{\mathsf{pu}}(x)), value(x, v)$$

Finally, to encode message firing (5.1) *from Example 4 we introduce two propositions* p_{dl} *and* p_{pu} *for concepts* DoorsLocked *and* PressureUp*, respectively. In particular,* p_{dl} *is true if there exists a signal in* DoorsLocked *that has at least one value. And similarly for* p_{pu}*. This is encoded with the rules:*

$$\begin{aligned} p_{\mathsf{dl}} &\leftarrow \diamondminus_{[0,\infty)}\mathsf{DoorsLocked}, \\ p_{\mathsf{dl}} &\leftarrow \diamondplus_{[0,\infty)}\mathsf{DoorsLocked}, \\ p_{\mathsf{up}} &\leftarrow \diamondminus_{[0,\infty)}\mathsf{PressureUp}, \\ p_{\mathsf{up}} &\leftarrow \diamondplus_{[0,\infty)}\mathsf{PressureUp} \end{aligned}$$

Here, $\diamondminus_{[0,\infty)}$ *and* $\diamondplus_{[0,\infty)}$ *are used to check if* DoorsLocked *has at least one signal with value in the past or in the future, respectively.*

Then we encode with the firing a message:

$$message(\text{“}All\ car\ doors\ OK\text{”}) \leftarrow p_{\mathsf{dl}}, p_{\mathsf{up}}$$

△

Analytical expressions C	Encoding of C
Q	$\tau_C(x) \leftarrow Q(x), \text{value}(x, v).$
$\{s_1, \ldots, s_m\}$	$\tau_C(s_i) \leftarrow \text{value}(s_i, v)$, for each s_i
$\alpha \circ C_1$, where $\circ \in \{+, -, \times, /\}$	$\tau_C(f_C(x)), \text{value}(f_C(x), v) \leftarrow \tau_{C_1}(x),$ $\text{value}(x, v), v = \alpha \circ v'.$
$C_1 : \mathit{value}(\odot, \alpha)$, where $\odot \in \{<, >, \leq, \geq\}$	$\tau_C(f_C(x)), \text{value}(f_C(x), v) \leftarrow \tau_{C_1}(x),$ $\text{value}(x, v), v \odot \alpha.$
$C_1 : \mathit{duration}(\geq, t)$	$\tau_C(f_C(x)) \leftarrow \diamondsuit_{[0,t]} \boxplus_{[0,t]} \tau_{C_1}(x).$ $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v).$
$C_1 : \mathit{duration}(<, t)$	$\tau_C(f_C(x)) \leftarrow \tau_{C_1}(x),\ \neg(\diamondsuit_{[0,t]}(\boxplus_{[0,t]}\tau_{C_1}(x)).$ $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v).$
$\mathsf{agg}\ C_1$, where $\mathsf{agg} \in \{\mathsf{min}, \mathsf{max}, \mathsf{avg}, \mathsf{sum}, \mathsf{count}\}$	$v = \mathsf{agg}[\![v_1 \mid \text{value}(x, v_1), \tau_{C_1}(x)]\!],$ where c is a fresh constant, $\mathsf{agg}[\![\cdot]\!]$ is an aggregation operator over bags
$C_1 : \mathit{after}[t]\ C_2$	$\tau_C(f_C(x_1)) \leftarrow (\tau_{C_1}(x_1))\mathcal{U}_{[0,\infty)}$ $\big((\neg\tau_{C_1}(x_1) \wedge \neg\tau_{C_2}(x_2))\,\mathcal{U}_{[0,t]}\ \tau_{C_2}(x_2)\big).$ $\text{value}(f_C(x_1), v) \leftarrow \tau_C(f_C(x_1)), \text{value}(x_1, v)$
$C_1 : \mathit{before}[t]\ C_2$	$\tau_C(f_C(x_1)) \leftarrow (\tau_{C_1}(x_1))\mathcal{S}_{[0,\infty)}$ $\big((\neg\tau_{C_1}(x_1) \wedge \neg\tau_{C_2}(x_2))\,\mathcal{S}_{[0,t]}\ \tau_{C_2}(x_2)\big).$ $\text{value}(f_C(x_1), v) \leftarrow \tau_C(f_C(x_1)), \text{value}(x_1, v)$
$C_1 : \mathit{within}\ C_2$	$\tau_C(f_C(x_1)) \leftarrow \Big((\tau_{C_1}(x_1) \wedge \tau_{C_2}(x_2))$ $\mathcal{S}_{[0,\infty)}(\neg\tau_{C_1}(x_1))\Big)\ \mathcal{U}_{[0,\infty)}\ (\neg\tau_{C_1}(x_1)).$ $\text{value}(f_C(x_1), v) \leftarrow \tau_C(f_C(x_1)), \text{value}(x_1, v).$
$C_1 : \mathit{trend}(\mathsf{up})$	$\tau_C(f_C(x)) \leftarrow \tau_{C_1}(x), \neg\ \text{notTrendUp}_{C_1}(x)$ $\text{notTrendUp}_{C_1}(x) \leftarrow \tau_{C_1}(x), \text{value}(x, v_1),$ $\diamondsuit_{(0,\delta]}(\text{value}(x, v_2), v_1 > v_2)$ where δ is a "small enough" positive real number $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v).$
$C_1 : \mathit{trend}(\mathsf{down})$	$\tau_C(f_C(x)) \leftarrow \tau_{C_1}(x), \neg\ \text{notTrendDown}_{C_1}(x)$ $\text{notTrendDown}_{C_1}(x) \leftarrow \text{value}(x, v_1),$ $\diamondsuit_{(0,\delta]}(\text{value}(x, v_2), v_1 < v_2)$ where δ is a "small enough" positive real number $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v).$

Boolean combinations D	Encoding of D
$D = C$	$p_D \leftarrow \diamondplus_{[0,\infty)}\tau_C(x).$ $p_D \leftarrow \diamondsuit_{[0,\infty)}\tau_C(x).$
$D = D_1$ and D_2	$p_D \leftarrow p_{D_1}, p_{D_2}.$
$D =$ not D_1	$p_D \leftarrow \neg p_{D_1}.$
$\mathit{message}(m) = D$	$\mathit{message}(m) \leftarrow p_D.$

Figure 4.2: The encoding *SAL* language into extended *datalogMTL*. For each analytical expression in the left column, the corresponding *datalogMTL*rules are provided in the right column.

5 Ontology Language for Semantically driven Analytical Workflow Generation

In this chapter we present our results on using semantically defined workflows for data analysis on Industrial data. We present a new approach that incorporates conceptual semantics of analytical workflows, inputs and outputs and evaluate the effectiveness of the proposed approach on a number of diagnostic tasks on real industrial use-cases. Our conclusion is that semantic-based methods that considers analytical concepts can achieve a higher interoperability and performance than most of the state-of-the-art systems used for authoring and executing workflows.

The material in this chapter has been published in [98, 105, 106, 99, 107, 100].

5.1 Introduction

Analytical workflows are heavily used in large and data intensive companies. An important application of such workflows is equipment analytics when equipment KPIs and reports are computed by aggregating equipments operational, master, and analytical data. In most of the cases, this data satisfies data variety dimensions and this dependence poses significant challenges in authoring, reuse, and maintenance of analytical workflows by engineers and data scientists. In this chapter we will address these problems by relying on semantic technologies: we use ontologies to give a high level representation of equipments operational and master data and offer a high level language to express an analytical workflows over ontologies.

An analytical workflow typically consists of the following steps:

1 data access when users obtain permissions to enterprise data on different levels,

2 data analysis and discovery, when users extract and analyse data by interacting with the existing templates for dashboards and extract relevant knowledge from data,

3 collaboration and sharing when users find extra insights from the data and knowledge when shared with colleagues.

Modern Business Intelligence systems and analytical platforms allow to combine these steps in analytical workflows and to iterate over them. Step 2 in such workflows is where self-service is crucial. Indeed, an analytical platform should be easy to use so that business users from all skill levels can easily reuse a dashboard or modify and add components. In data intensive companies such as Siemens such self-service is often hampered by the fact that re-use and modification of dashboards and their components require deep knowledge of schemata and formats of underlying data. Due to the challenging data dimensions, such knowledge is only affordable to IT specialists.

In this chapter, we present our developed semantically defined workflows that incorporate semantics into the traditional workflows. This chapter addresses the following research question:

[RQ3]Can the semantic-driven analytical workflows boost data analysis performance?

To this end, we propose to extend the traditional data driven approach to analytics with an OBDA layer and a new semantic language to what we call *SAL*, described in previous Chapter 4. Our language allows to write complex analytical workflows in an abstract fashion and to exploit both ontological vocabulary and queries over ontologies to identify relevant data sets and analytical models implemented in various technology platforms. We designed the language in such a way that, on the one hand, it captures the main data analysis features and, on the other hand, it has good computational properties. In particular, *SAL* allows for rewriting [23] of analytical workflow written over OWL 2 QL ontologies into multiple data-dependent rule-sets with the help of ontologies and OBDA mappings. We implemented *SAL* and a prototypical Semantically defined analytical system. We evaluated usability of our solution with engineers at Siemens by checking how fast they can formulate and combine workflows using our *SAL*. We also evaluated the efficiency of our solution in processing diagnostic workflows over turbine signals in a controlled environment. (See chapter 7 for details.)

In rest of this chapter we introduce our language , then we specify the details of capturing analytical workflows and their semantics. Lastly, we define the formal properties of the workflow language.

5.2 Workflow Generation using SAL

We now show how to use analytical expressions to compose workflows and to create alert messages.

In the following we will consider *well formed* sets of analytical expressions, that is, sets where each concepts 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_2.

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

Example 3. *The running example program* $\Pi = (\mathcal{S}, \mathcal{K}, \mathcal{H})$ *has the following components: signals* $\mathcal{S}$ *for sensors* $\{\mathsf{SKNF\_X01}, \mathsf{SKNF\_X02}, \mathsf{SKNF\_X03}, \mathsf{SKNF\_X04}\}$, *KB* $\mathcal{K}$ *that consists of axioms from Equations* (4.1) *and* (4.2), *and* $\mathcal{H}$ *that consists of expressions from Equations* (4.4) *and* (4.5). △

5.2.1 Message Rules

On top of workflows Π *SAL* allows to define *message rules* that report the current status of a system.

Formally, they are defined as Boolean combinations of analytical expressions:

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

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

$$message(m) = D.$$

Example 4. *Using Equations* (4.4)–(4.5) *we define the following message:*

$$message(\text{“}All\ car\ doors\ OK\text{”}) = \mathsf{DoorsLocked}\ and\ \mathsf{PressureUp}. \tag{5.1}$$

The message intuitively indicates that the doors are functioning and locked. △

Now we are going to define the semantics of the analytical workflows.

5.2.2 Semantics of Workflow and Firing a Message Rule

We now define how to determine whether a workflow Π fires a message rule r.

Let Π be a workflow 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, where $\mathcal{I}_{FOL}$ interprets

objects and their relationships (like in OWL 2) and $\mathcal{I}_{\mathcal{S}}$—signals. it holds $C^{\mathcal{I}_{FOL}} \neq \emptyset$, that is, the concept that fires r is not empty. Our workflows and rules enjoy the *canonical* model property, that is, each workflow has a unique (Hilbert) interpretation [17] which is minimal and can be constructed starting from basic signals and ontology by following analytical expressions. Thus, one can verify $C^{\mathcal{I}_{FOL}} \neq \emptyset$ only on the canonical model. This implies that one can evaluate *SAL* workflows and expressions in a bottom-up fashion. We now illustrate this approach on our running example.

Example 5. *Consider our running workflow* 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* MainCarDoors *that collects all sensor ids for a particular train. Then, we evaluate the expressions in* $\mathcal{M}$ *following the dependency graph of definitions. We start by evaluation the expression from Equation* (4.4), *again in a bottom-up fashion. Concept* $\mathsf{MainCarDoors}^{\mathcal{I}_{\Pi}}$ *contains sensor ids:* SKNF_X01, SKNF_X02, SKNF_X03 *and* SKNF_X04. *At the same time, those sensors have analytical functions assigned from* $\mathcal{S}^{\mathcal{I}_{\Pi}}$. *Let us call them* f_1, f_2, f_3 *and* f_4. *Expression* sum MainCarDoors *computes a new signal, say* s_5, *by taking sum of* f_1, f_2, f_3 *and* f_4 *at each time point. After this, it eliminates all values of* s_5 *that are* $\neq$ *LockedValue. Similarly, we compute signal transformations for the expression from Equation* (4.5). *Finally, we use those two expressions to evaluate the message rule from Equation* (5.1). *If there exists at least one signal in evaluated expressions corresponding to Equations* (4.4) *and* (4.5), *then the message is fired.* △

5.3 Formal Properties of Semantically driven Analytical Workflows

In this part, we study the formal properties for workflows generated using our *SAL* language.

First, we assume from now that signal functions are given on the input as a step functions over intervals (which is the case in the running example).

Second, we would like to ensure that our workflows in *SAL* indeed allow to be rewritten using OBDA techniques. This condition requires more technical explanation and we discuss it in more detail in the next paragraph.

Third, we want to understand what is the upper bound of the complexity of our problem. In particular, we measure the complexity of the problem in the size of two main components: the size of workflow/program and data. We expect that size of the data largely dominates the size of workflow, and while the data can be huge (several GBs) that size of program can be significantly big (several thousands of rules) thus both measures are relevant.

First Order (FO) rewritability The formal condition that determines if the workflow language is suitable for OBDA is called First Order (FO) rewritability. In this part we define it formally. We encode the specification of firing a message in *SAL* workflow into fact-entailment problem over an extended version of recently introduced non-recursive metric Datalog [103], Datalog$_{\text{nr}}$*MTL* . The reason for doing this is twofold. First, Datalog$_{\text{nr}}$*MTL* (inspired by a well-studied Metric Temporal Logic [104]) provides a natural way to model rules that reason over time intervals. Second, Datalog$_{\text{nr}}$*MTL* is a suitable language for OBDA setting, that is, it has been show how to rewrite queries over the rules in Datalog$_{\text{nr}}$*MTL* into standard SQL over the sources [103].

Still, Datalog$_{\text{nr}}$*MTL* cannot be immediately related to our language since it does not support aggregates and some other logic constructs that we need for our encoding (in particular, functional symbols, negation and aggregates [102]). So first, we extend DatalogMTL with functional symbols, aggregation, etc. under reasonable restrictions, without increasing the complexity. Then to show that our problem is FO-rewritable we do the following encoding. Given a workflow Π and a message rule r we create an extended non-recursive Datalog$_{\text{nr}}$*MTL* $\Sigma_{\Pi,r}$ and a proposition m_r only such that: Π fires r iff Σ_Π "entails" m_r. A corollary of this gives us *(i)* that our language is suitable for OBDA setting (follows from the encoding); *(ii)* ways to reformulate our workflows and rules into SQL queries (extending the principles in [103]).

5.3.1 Extended Datalog*MTL*

In this part we introduce our extension of Datalog*MTL*. At this moment we only briefly introduce the main constructs of the workflow.

A *datalogMTL program*, Σ, is a finite set of *rules* of the form

$$A^+ \leftarrow A_1 \wedge \cdots \wedge A_k \qquad \text{or} \qquad \bot \leftarrow A_1 \wedge \cdots \wedge A_k,$$

where A^+ is an atom that *does not* contain any 'non-deterministic' operators $\diamondplus_\varrho$, $\diamondminus_\varrho$, $\mathcal{U}_\varrho$, $\mathcal{S}_\varrho$, negated atoms, or aggregate operators.

For our purposes it is sufficient to have non-recursive programs. Informally, that are programs where dependency (direct or indirect) between predicates is acyclic. In fact, it is not trivial to understand how one would even define recursion in case of aggregates and negation. Therefore, we only consider extended Datalog*MTL* program that are non-recursive.

Finally, every satisfiable Datalog*MTL* Σ program with database $\mathcal{D}$ has the *canonical* (or *minimal*) *model of* Π *and* $\mathcal{D}$, $\mathfrak{M}_{\Pi,\mathcal{D}}$. As usual, the most important property of canonical model is that if a fact holds in canonical model then it holds in any other model.

5.3.2 Encoding into Extended Datalog*MTL*

In this part we provide full specification of the encoding.

Let $(\mathcal{S}, \mathcal{K}, \mathcal{H})$ be an *SAL* workflow. We define a corresponding extended Datalog MTL program $(\mathcal{D}_{\mathcal{S},\mathcal{K}}, \Pi_{\mathcal{H}})$ where temporal facts $\mathcal{D}_{\mathcal{S},\mathcal{K}}$ encodes $\mathcal{S}$ and $\mathcal{K}$ and program $\Pi_{\mathcal{H}}$ encodes expressions $\mathcal{H}$ in the following way.

For each basic signal $s = (o_s, f_s)$ in $\mathcal{S}$:

- if $f_s(t) = v$ we add $value(s, v)@t$ to $\mathcal{D}_{\mathcal{S},\mathcal{K}}$, and
- if o_s is an answer of Q over KB $\mathcal{K}$ then we add $Q(o_s)@(-\infty, +\infty)$ to $\mathcal{D}_{\mathcal{S},\mathcal{K}}$.

We observe that encoding of signals as a finite database instance is possible due the fact that we assume signals to be step functions.

Workflow $\Pi_{\mathcal{H}}$ is constructed from $\mathcal{H}$ following the encodings in Table 4.2. The encoding is obtained by using a unary predicate τ_C for each analytical expression C and binary predicate value which we describe in the next paragraphs. It is important to note that these predicates are interpreted not like FO-predicates but using point based semantics (e.g., $\tau_C(o)$ is true or false for a constant o *at a given time point* t). For detailed semantics of such rules see [103].

More formally, for a signal $s = (o_s, f_s)$, the fact $\tau_C(o_s)$ is true at a time point t iff *(i)* $o \in C^{\mathcal{I}}$ and *(ii)* $f_s(t)$ is a real number. Condition *(ii)* simplifies the encoding since we do not need to define when a signal does not have a real value at a point; otherwise we have to have the rules that encode the absence of a real value. Further, we use functional symbols, e.g., f_C, to generate fresh signal identifier. E.g., for a signal s, $f_C(o_s)$ represents a new signal id obtained from s for the expression C.

To store the value of a signal at a time point we use the predicate value. That is, value(o_s, v) is true at point t iff $f_s(t) = v$.

The encoding rules for *trend*(up) and *trend*(down) are based on intervals. For them we introduce a parameter δ, a "small" real number, that we use to select an interval around a time point. In theory, such parameter should converge to 0 to indeed check the trend of a real function (in fact, one needs the first derivative), however, in practice we expect that one can select such δ a priori (e.g., the length of signal sampling since signals are step functions) that is sufficiently small to check the trend of a function for a particular time point.

5.3.3 Formal Properties of the Encoding

In this part we state the formal properties of the encoding and the most import consequence: FO rewritability.

We do this in two steps. First we introduce two lemmas that characterize the encodings of auxiliary predicates *value*, τ_C's and propositional p_D's. Then we use them to show the main encoding theorem.

The following lemma establishes correspondence between a program and auxiliary predicates *value* and τ_C's.

Lemma 1. *Let $\Pi = (\mathcal{S}, \mathcal{K}, \mathcal{H})$ be an SAL workflow and $\Sigma_\Pi = (\mathcal{D}_{\mathcal{S},\mathcal{K}}, \Pi_\mathcal{H})$ be an extended DatalogMTL program as defined above. Further, let $\mathcal{I}$ be the canonical interpretation for Π and let $\mathfrak{M}$ be the canonical interpretation for Σ_Π. Then, for a analytical expression C and a time point t the following is equivalent:*

- $s^\mathcal{I} \in C^\mathcal{I}$ *and* $f_s(t) = v$;
- $\mathfrak{M}, t \models \tau_C(o_s)$ *and* $\mathfrak{M}, t \models \mathit{value}(o_s, v)$.

Proof. The proof (in the both directions of "iff") is based on induction on the number of rules that are required to generate expression C starting from basic analytical expressions. We show direction "$\Leftarrow$". The opposite one can be shown analogously.

Induction Base: In this case, C is defined either with $C = Q$ or $C = \{s_1, \ldots, s_m\}$.

Let us assume $C = Q$, and $s^\mathcal{I} \in C^\mathcal{I}$ and $f_s(t) = v$. Since s is a basic signal, $\mathcal{D}_{\mathcal{S},\mathcal{K}}$ must contain the fact $\mathrm{value}(o_s, v)@t$. Moreover $\mathcal{K} \models Q(o_s)$ hence according to the rule $\tau_C(x) \leftarrow Q(x), \mathrm{value}(x, v)$ we have that $\mathfrak{M}, t \models \tau_C(o_s)$. Since, $\mathrm{value}(x, v)@t \in \mathcal{D}_{\mathcal{S},\mathcal{K}}$ we also have that $\mathfrak{M}, t \models \mathrm{value}(o_s, v)$.

Assume now that $C = \{s_i, \ldots, s_m\}$ and $s = s_i$ for some i. Then it must be $s^\mathcal{I} \in C^\mathcal{I}$. Next, let us assume that $f_s(t) = v$. Since s is a basic concept, we have that $\mathrm{value}(o_s, v)@t$ is in $\mathcal{D}_{\mathcal{S},\mathcal{K}}$, and thus $\mathfrak{M}, t \models \mathrm{value}(o_s, v)$. Further, following the encoding rule for C, $\tau_C(x) \leftarrow \mathrm{value}(x, v)$, we have that $\mathfrak{M}, t \models \tau_C(o_s)$.

Induction Step: Consider now that C is an expression that is created by other expressions in at most $n+1$. For example, let us assume that $C \leftarrow C_1 : \mathit{duration}(\geq, t')$. Induction step for the other rules can be shown analogously.

We assume that $s^\mathcal{I} \in C^\mathcal{I}$ and $f_s(t) = v$ for some t and v. Since C is created from C_1 then it must exists s_1 such that $o_s = f_C(o_{s_1})$ and $f_{s_1}(t) = v$ for some interval I that contains t and is longer t'. Since, C_1 is created in at most n steps by induction hypothesis we have that $\mathfrak{M}, t \models \tau_{C_1}(o_{s_1})$ and $\mathfrak{M}, t \models \mathrm{value}(o_{s_1}, v)$. Now we analyze the encoding rule $\tau_C(f_C(x)) \leftarrow \diamondsuit_{[0,t']} \boxplus_{[0,t']} \tau_{C_1}(x)$. Intuitively, the body

of the rule evaluates to true for some x if there exists a time point in the "past" of t (expressed with condition $\diamondminus_{[0,t']}$) contained in an interval of size t' (expressed with condition $\diamondminus_{[0,t']}$) such that on that interval $\tau_{C_1}(x)$ is true, i.e, $\mathfrak{M}, t'' \models \tau_{C_1}(x)$ for all $t'' \in I$. Since I is such interval for which $\tau_{C_1}(o_{s_1})$ is true, we have that encoding rule fires and makes $\tau_C(f_S(o_s))$ true at point t, i.e., $\mathfrak{M}, t \models \tau_C(f_S(o_{s_1}))$. Furthermore, from the rule $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v)$ and the fact that $\mathfrak{M}, t \models \tau_C(f_S(o_{s_1})), \text{value}(s_1, v)$ it holds that $\mathfrak{M}, t \models \text{value}(f_C(o_{s_1}), v)$. This concludes the proof. □

The following lemma defines correspondence between a Boolean combinations of analytical expressions and their encoding rules.

Lemma 2. *Let $\Pi = (\mathcal{S}, \mathcal{K}, \mathcal{H})$ be an SAL workflow and $\Sigma_\Pi = (\mathcal{D}_{\mathcal{S},\mathcal{K}}, \Pi_\mathcal{H})$ be an extended DatalogMTL program as defined above. Further, let $\mathcal{I}$ be the canonical interpretation for Π and let $\mathfrak{M}$ be the canonical interpretation for Σ_Π. Then, for a Boolean combination of analytical expression D we have that the following is equivalent:*

- *$D^\mathcal{I}$ is true;*
- *$\mathfrak{M}, t \models p_D$ for all time point t.*

Proof. The proof is based on induction on the size of the Boolean combination that constitutes D.

Induction Base: We assume $D = C$ for complex expression C and assume $D^\mathcal{I}$ is true. Then there must exists a signal s such $o_s \in C^\mathcal{I}$ which has at least one value v at some time point t. From Lemma 1 we have that $\mathfrak{M}, t \models \tau_C(o_s)$. Thus, from the encoding rule $p_D \leftarrow \tau_C(x)$ we have that $\mathfrak{M}, t \models p_D$.

Induction Step: We prove induction step for the case $D = D_1$ and D_2. Similarly, it can be show in case $D = \neg D_1$.

Assume that D is true in $\mathcal{I}$, then also D_1 and D_2 are true. Since D_1 and D_2 are constructed in less steps then D by induction hypothesis we have that $\mathfrak{M}, t \models p_{D_1}$ and $\mathfrak{M}, t \models p_{D_2}$. Hence, $\mathfrak{M}, t \models p_D$. □

For an extended DatalogMTL program Σ, ground atom A and a time point t we define that $\Sigma \models A@t$ for the canonical model $\mathfrak{M}$ of Σ it holds $\mathfrak{M}, t \models A$ Then, directly from Lemmas 1 and 2 we have the following theorem.

Theorem 1 (Encoding Theorem)**.** *Let Π be a processing workflow and r a message rule. Let Σ_Π the extended DatalogMTL that encodes Π as described above and let the grounded propositional m_r be the head of DatalogMTL rule encoding r. Then the following holds:*

$$\Pi \textit{ fires } r \quad \textit{iff} \quad \Sigma_\Pi \models m_r@t, \textit{ for any time point } t$$

5.3.4 Consequences of the Encoding Theorem

In this part we analyze the direct consequences of the encoding theorem.

First, we observe is extended DatalogMTL program Σ_Π is a non-recursive one. It is not hard to show ideas of Theorem 5 in [103] that new extended DatalogMTL will preserve computational properties required for OBDA setting. Formally, that means that *data complexity* [23] for the fact-entailment problem is in AC^0 in data complexity.

The second observation is that we can extend rewriting techniques developed for DatalogMTL in [103] that allow us to rewrite our rules into standard SQL. More involving part of rewriting lies on rewriting algebra of intervals, and for more details we refer [103]. Rewriting that includes functional symbols, negation, aggregation, and built-in arithmetic can be done can be done straightforwardly.

Let Σ be an extended DatalogMTL program, $\mathcal{D}$ a set of facts and A an grounded atom. As usual, $\Sigma, \mathcal{D} \models A@t$ for some time point t holds if for the canonical model $\mathfrak{M}$ of $\Sigma \cup \mathcal{D}$ it holds $\mathfrak{M}, t \models A$. Decision problem *success* is the problem of checking whether $\Sigma, \mathcal{D} \models A@t$. We refer to program (resp. data) complexity if all parameters are fixed except the program (resp. set of facts).

Lemma 3. *Success problem for extended DatalogMTL programs is PSPACE-complete in combined and program complexity and in AC^0 in data complexity.*

Proof Idea. Hardness follows from Theorem 5 in [103]. To show membership it is sufficient to observe that each derivation in an extended DatalogMTL (as in regular Datalog program) program is of length polynomial in the size of the program. Thus it is in *PSPACE*. □

From Lemma 3 and Theorem 1 we have the following.

Theorem 2. *The problem of checking whether a message rule is fired is PSPACE-complete (it is complete already in size of analytical expressions and workflow), and it is in AC^0 in the size of signal data and ontological data.*

5.4 Analysis of Workflows Generation using SAL

There are certain challenges with management of analytical workflows. Development of a diagnostic or analytical workflow is typically a collaborative and open ended process by a group of diagnostic engineers. Thus, the engineers may introduce models that either repeat what other models already express or contradict them, i.e., by stating that purging is *over* while the other rule model says that is it *in progress*.

The former problem of redundancy in diagnostic analytics affects the performance of diagnostics and the latter of inconsistency among models makes diagnostic results counter-intuitive and unreliable. Moreover, the complex the model 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 workflow analysis support that includes detection of redundancy and inconsistency in analytical workflows, as well as computation of provenance for diagnostic results.

In order to address the above mentioned challenges, we propose how to execute semantic workflows, verify redundancy and inconsistency in workflows, and to compute provenance that explains the reasons for analytical results.

Algorithm 1: Firing a message

Input: program $\Pi = (\mathcal{D}, \Sigma)$, and a message $msg(m) \leftarrow D$
Output: true if $\Pi \models m$, false otherwise

Step 1 For each concept C in $\mathcal{O}$ do *classification* [23], that is, compute all sub concepts $sub(C)$ of C implied by $\mathcal{O}$, i.e., $C' \in sub(C)$ iff "$\mathcal{O} \models C' \sqsubseteq C$".

Step 2 For each signal expression C in Σ, compute $CICAN$ by

- ***2.1*** Replacing each C in Σ with all sub classes $sub(C)$ in all possible ways. Let $\Sigma_{\mathcal{O}}$ be the new set of rules.
- ***2.2*** Then evaluate each of the expressions by computing $ICAN$ of $\Sigma_{\mathcal{O}}$ in a bottom-up fashion starting from $\mathcal{D}$.

Step 3 Return true if $DICAN \neq \emptyset$; false otherwise.

5.4.1 Redundancy of workflows

One of the critical problems of the workflow analysis is redundancy. To analyse this problem, the simplest test is to understand whether one message from a certain workflow 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 analytical layer of a program. Formally, given messages m_1 and m_2 we say that m_1 is *implied* by m_2 over the analytical 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 [108],

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 $\mathsf{filterValue}(>, 20) \Rightarrow \mathsf{filterValue}(>, 10)$ holds). Moreover, in order to avoid exponential generation of new interval in signal rules, we assume that signal functions are step functions over uniform size intervals and that signal expression are following this interval granularity when defining new signals.

Under these assumptions, Algorithm 2 allows us to verify whether there is a redundancy between two message rules.

Algorithm 2: Checking redundancy

Input: Workflow layer Σ, messages $msg(m_1) \leftarrow D_1$, $msg(m_2) \leftarrow D_2$
Output: true if $\Sigma \models m_1 \Rightarrow m_2$; otherwise false

Step 1: Unfold D_1 (reps. D_2) following signal expression in Σ into D_1' (reps. D_2') such that it contains only basic concepts from the ontology.

Step 2: Unfold D_1' (reps. D_2') further into D_1'' (reps. D_2'') following classification as defined in Steps 1 and 2.1 of Alg. 1.

Step 3: Turn D_1'' (reps. D_2') in propositional Boolean formula ϕ_1'' (reps. ϕ_2') by dropping signal operations, treating each concept as a propositional, and treating *and* and *not* as logical operators.

Step 4: If $\phi_1'' \wedge \neg\phi_2'$ is un-satisfiable return true; otherwise false.

5.4.2 Consistency of workflows

Another important task in workflow analysis is checking if two messages from the workflow are behaving consistently with their meaning.

In particular, we check whether for a given analytical 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 workflows 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 $\Sigma \models \text{consist}(m_1, m_2)$ 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 $m_2^{\neg} \leftarrow not\ D$, then $\Sigma \models \text{consist}(m_1, m_2)$ iff $\Sigma \models m_1 \Rightarrow m_2^{\neg}$. Hence, one can adapt Algorithm 2 for checking consistency.

5.4.3 Provenance of workflow

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-workflow of Π that fire the message. Notice that there may exist several such minimal sub-workflows. One of these can be computed by iteratively removing all superfluous axioms, until only relevant ones remain [109].

5.4.4 Computational Complexity

Now we analyze the computational complexity of the following tasks: *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 5.1. In the following we provide intuitions for each of the tasks.

Following Algorithm 1 we prove 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 and each basic concepts has at most polynomial many sub-concepts 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 [23] by unfolding Boolean expressions, signal expressions and ontology as in Algorithm 2, and then checking firing as query evaluation.

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

Table 5.1: Computational complexity of our reasoning task. The complexity means that our problem is in that class; -c means that the problem is complete for that class.

Regarding *redundancy* and *consistency*, the membership in coNP follows from the algorithm, and the hardness from coNP- hardness of un-satisfiability 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 [110].

6 Semantically-defined Analytics System

In this chapter, we first present the architecture of our Semantically-defined Analytics system, describe its deployment in practice.

The material in this chapter has been published in [98, 105, 107].

6.1 System Architecture

The main functionality of our Analytics-aware Semantic Diagnostics system for industrial use-cases is to formulate *SAL* analytical workflows using the analytical functions, to deploy them in various components of an industrial use-cases, to execute the workflows in these components, and to visualise the results of execution. We now give details of our system by following its architecture in Figure 6.1. There are four essential layers in the architecture, two of which, *application* and *OBDA*, reside in the centralised element of the architecture, and two, *analytics execution* and *data*, reside in individual components of the industrial system. Our system is mostly implemented in Java. We now discuss the system layer by layer.

Application Layer

On the *application layer*, the system offers two user-oriented modules. The first module allows engineers to author, store, and load diagnostic workflows by formulating sets of analytical workflows in *SAL* and data retrieving queries. Such formulation is guided by the domain ontology (see Chapter 3) stored in the system. In Figure 6.2 (top-left) one can observe a screenshot of the semantic language 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 semantic workflows), and to track deployment of workflows in equipment. In Figure 6.2 (top-right) one can see visualisation of signals from two components of one turbine. Analytical workflows formulated in the application layer are converted into XML-based specifications and sent to the OBDA layer, which returns back the messages and materialised semantic signals, that is, signals over the ontological terms. In

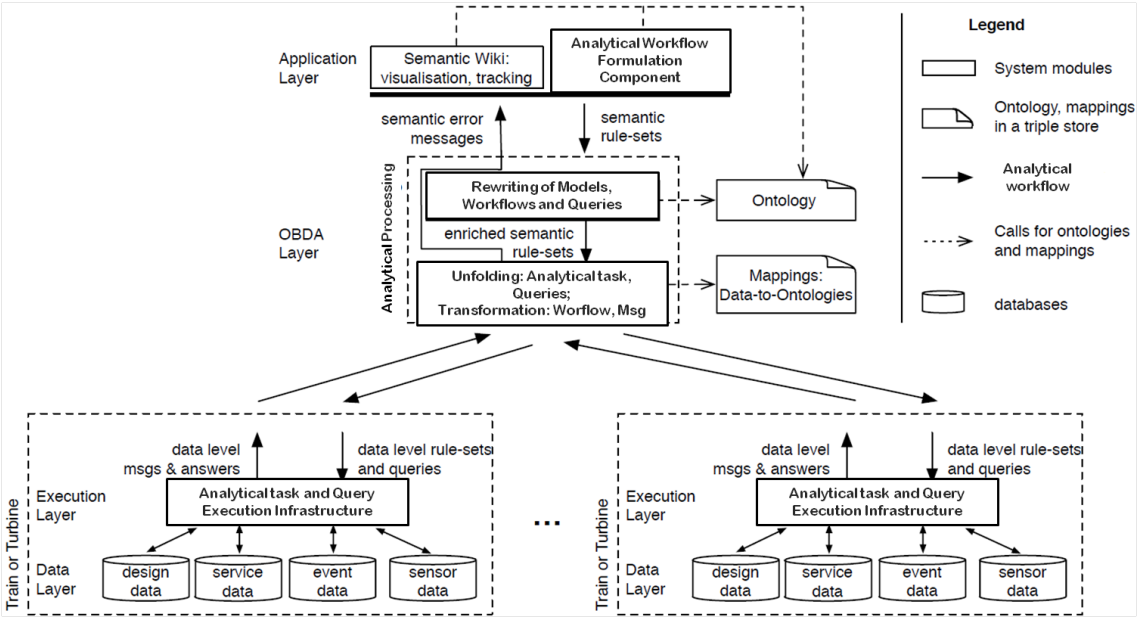

Figure 6.1: Architecture of Analytics-aware Semantic Diagnostics system for industrial use-cases.

Figure 6.2 (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.

Note that during the course of the thesis, we have developed an extension to the existing Siemens diagnostic rule-based editor and a dedicated wiki-based visualisation monitor for semantic data signals. Also note that we use the latter for visualising query answers and messages formatted according to our proposed domain-ontology and stored as RDF.

SAL driven OBDA Layer

The *OBDA layer* takes care of transforming semantic workflows written in *SAL* into XML-specification with appropriate SQL and program scripts. This transformation has two steps: rewriting of workflows and queries with the help of ontologies (at this step both workflows 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 which we were developing earlier within an FP7 European project called Optique [81]. The OBDA layer also transforms signals, query answers, and messages from the data to semantic representation.

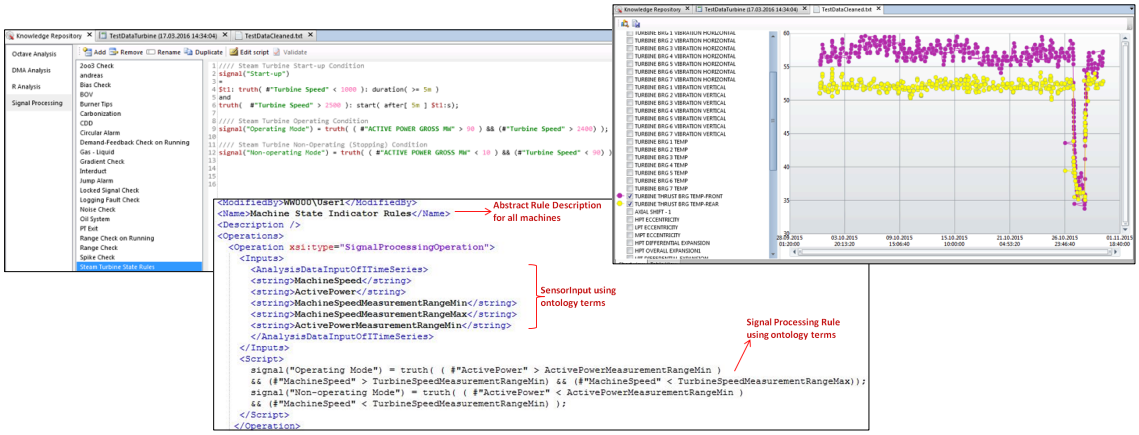

Figure 6.2: SPR editor (top-left), Wiki-based visualisation monitor for semantic signals (top-right), and a fragment of an XML-based specification of an workflow (bottom).

Workflow Execution Layer

The *workflow execution layer* takes care of planning and executing data-driven analytical functions and queries received from the OBDA layer; and it is deployed in each piece of equipment that participates in the industrial equipment. If the received functions are in for example the drools rule language then the executor instantiates them with concrete sensors extracted with queries and passes them to the Drools Fusion, the analytical rule engine currently used at Siemens. If the received functions are in SQL then it plans the execution order and executes them together with the other queries. Likewise, we support R scripts and KNIME analytical framework to execute respective analytics.

Data Layer

Finally, in the *data layer* there is all relevant data, e.g., equipment (train or turbine) design specifications, historical information about services that were performed over the equipment, previously detected events, and the raw sensor signals.

6.2 Deployment in Industrial Environment

For evaluation purpose, we have deployed our Analytics-aware Semantic Analytical system at Siemens power generation to analyze gas turbines equipment respectively. We integrated the system with four types of data sources namely Teradata, MS SQL, SAP HANA and IBM Maximo. More details are to follow in the case study section. For analytical processing we connected our system to the Siemens deployment of

Drools Fusion, R analytics, KNIME and Python platform. An important aspect of the deployment was the development of a domain-specific, analytical ontology and mappings which are adopted for each use-case. Details of the deployment application (see Fig.6.3) for power generation known as OpereX application is discussed below:

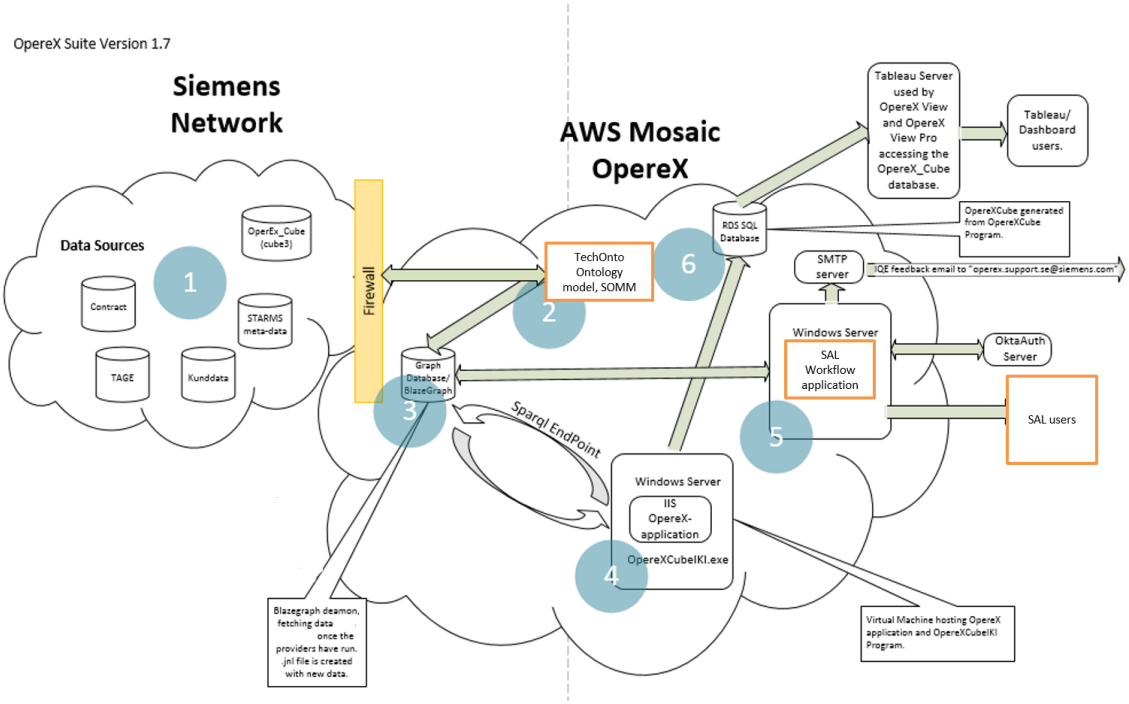

Figure 6.3: OpereX deployment suite for Semantically defined analytics for power generation business.

The following building blocks are part of Siemens power generation OpereX application:

1 There are four types of data sources namely Teradata, MS SQL, SAP HANA and IBM Maximo configured for contract related data, operational profile data, turbine design data, sensor data and maintenance related data.

2 The second component is our proposed solution where store the model/ontology and managing how the data should be fetched and stored in the graph database by using R2RML mapping language. The steps are:

- Data sources i.e connections to other databases from where you want to fetch the data.
- Mappings, how to fetch the data from the data sources and map them into the graph database according to the ontology.
- Providers, runs the different mappings, fetches the data and store the OBDA data in the graph database. We already have a model/ontology finished, we use that to import data from the data sources and manipulate

the columns and rows from the different tables to store them in our Blazegraph which is a graph database.

3 Blazegraph is a graph database that uses triples to store all the OBDA data from the data sources. On the Blazegraph instance a script (Provider loader deamon) is run to fetch data from SOMM once a provider has been run. The script takes that data and push it to the Blazegraph graph database.

4 This is a Windows console application (.exe) that takes the data from Blazegraph and creates a new SQL cube with all the data and stores it inside a SQL-database in Mosaic.

5 The SAL workflow is a web service that visualize the data from Blazegraph as well as retrieving the data. This way SAL users doesn't need to know any database languages like CQL or SPARQL to get data. The user is able to execute analytical task and to generate semantic workflows using SAL language constructs. The implementation details are hidden and the user does not need to know how the algorithms are executed. This web service is using Siemens authentication application, and SMTP for email.

Scheduling of processes and applications The purpose and meaning of the OpereX suite is to provide data from different data sources and run analytics on the data for different users. The users can access the data via Tableau dashboards and via the SAL Editor, see Fig.6.2. Since the OpereX Suite is dependent of a continuous data flow for getting the latest data we need to schedule the different scripts, providers and analytical programs according to the OpereX suite overview above.

7 Case Studies and Evaluations

In this chapter, we present details on three use-cases selected to evaluate our work in an industrial setting. Firstly, we describe different analytical tasks, processes and current challenges to execute and manage data analytics in three industrial use-cases. Secondly, we present a number of evaluations conducted to determine the quality and performance metrics of our proposed ontology model, *SAL* language and semantic system. We also conducted evaluations to measure the effort to formulate and process analytical workflows as well as produce run-time analysis for different tasks and workflows for each use-case.

The material in this chapter is published in [100, 66, 111, 112, 113, 114, 115, 116, 117]

7.1 Case Description 1: Turbine Diagnostics

Siemens produces a variety of rotating appliances, including gas and steam turbines, generators, and compressors. These appliances are complex machines and typically used in different critical processes including power generation where each hour of downtime may cost thousands of Euros. Thus, these appliances should be under constant monitoring that requires an in-depth knowledge of their components and setup (see Fig.7.1). Siemens provides such monitoring via service centres and operates over fifty such centres worldwide, where each centre is responsible for several thousand appliances. Typical monitoring tasks of a service centre include: reactive and preventive diagnostics of turbines which is about data analysis applied after a malfunction or an abnormal behaviour such as vibration, temperature or pressure increase, unexpected events, or even unexpected shut-downs of a unit is detected; predictive analysis of turbine conditions which is about data analysis of data streams received from these appliances. We now discuss these monitoring tasks in detail and present requirements to enhance them.

Reactive and Preventive Diagnostics: is usually applied after a malfunction of a unit has occurred, e.g., the abnormal shut-down of a turbine. Complementing the preventive diagnostic task which is performed before a malfunction of a unit, when its abnormal behaviour is detected, e.g., high vibration or temperature increase. Diagnostic tasks are triggered either when a customer sends a service ticket claiming assistance or an automated diagnostic system creates such a ticket. Fig. 7.2 depicts

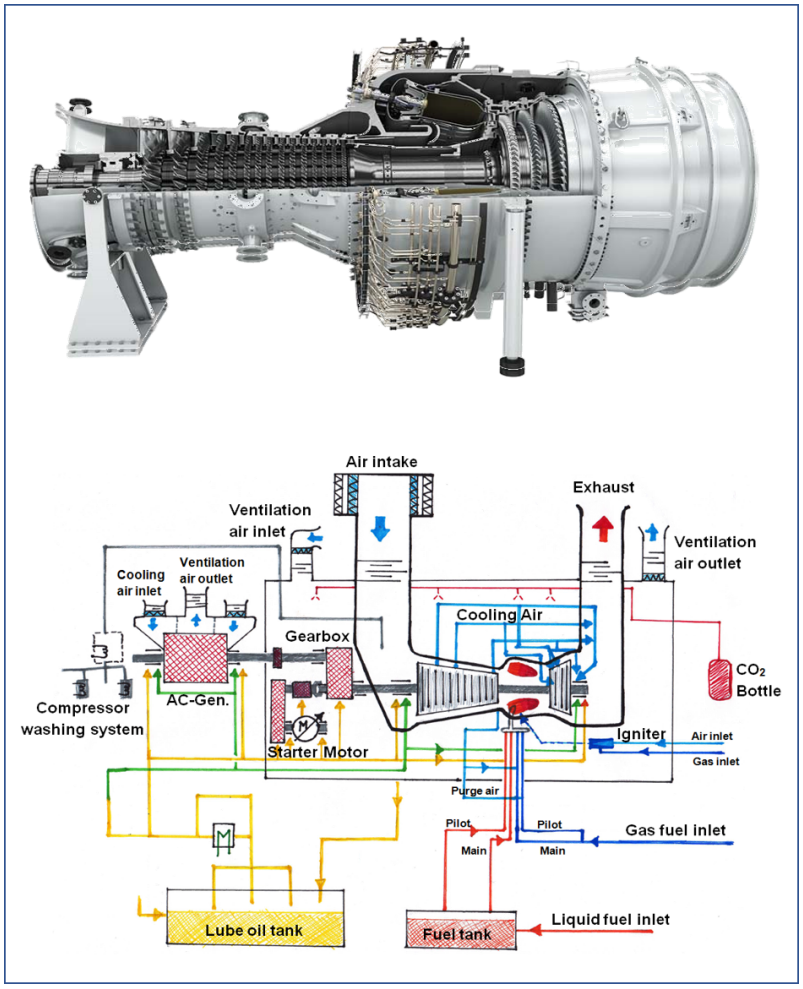

Figure 7.1: Siemens SGT-800 Turbine model and its structural components [118].

a general process triggered when a service ticket arrives. We now discuss each step of the process in detail.

Arrival of a service ticket. A service ticket typically contains information on when a problem occurred and its frequency. In some cases the ticket isolates the location of the problem in the appliance and its cause, but often it has no or few details.

An example of a reactive monitoring request from a customer is:

Example 6. *Figure out why the turbine failed to start during the last five hours, with the goal of checking that there will be no fault of the turbine.* △

A typical preventive monitoring request could be:

Example 7. *Will there be a failure of the turbine after the observed temperature increase?* △

Data acquisition: Service engineers gather relevant data by querying databases that are updated every hour, or on demand, and contain sensor and event data.

In order to support data gathering, Siemens equips service centres with more than 4,000 predefined queries and query patterns of different complexity. Engineers use the queries by setting parameters such as time periods, names of events or sensors, sensor types, etc.

Based on the service ticket of Example 6, the engineer formulates the following information need and has to find appropriate queries to cover it:

Example 8. *Return the most frequent start failure and warning messages of the gas turbine T01 during the last week. Moreover, find analogous cases of failures for turbines of the same type as T01 in the last three months.* △

Query result visualisation: Sensor data is visualised with the use of standard diagrams, and event messages are presented as a list, i.e., as an Excel spreadsheet, with timestamps and additional attributes.

Data preprocessing: The queried data is preprocessed using generic procedures such as sensor check (i.e., whether sensor data quality is appropriate), threshold and trend analysis. Independent from the concrete ticket, these preprocessing steps are done manually, e.g., over the visualised Excel spreadsheets, or using specialised analytic tools.

Data analysis: The engineer uses sophisticated diagnostic models and tools for complex analysis, e.g., Principal Component Analysis or other statistical methods, to detect and isolate the given problem based on the preprocessed data. Typically, analytical tasks are executed individually for each ticket. The gathering and analysis steps are often carried out iteratively, i.e., the results from one iteration are used to pose additional queries.

Report preparation: This process terminates when an explanation for the problem in the service ticket is established. In this case the engineer provides the customer with a report aggregating the result of the analysis and describing possible further actions.

Predictive Analysis In predictive analysis, in contrast to the diagnostic process described above, appliances are continuously monitored, i.e., without prior service tickets, using online processing of the incoming sensor data. The other process steps of predictive analysis are similar to the ones described in the previous section, but have to be applied online to streaming data with minimal user intervention. The purpose here is to analyse the current condition of an appliance by combining operating information, system data, specifications of concrete product lines, and temporal phases of operating regimes. This information allows to predict whether some parts of an appliance should be repaired soon, assess risks related to the use of these parts, and adjust maintenance intervals for each part by automatically integrating this information into service scheduling, thus, minimizing maintenance cost.

For predictive analysis of turbines, the diagnostic engineer may want to be automatically notified when a turbine shows repetitive start failures combined with increased vibration values during its operating time.

This can be formulated as follows:

Example 9. *Notify me if a turbine that had more than three start failures in the last two weeks additionally shows abnormal vibration values in operative phases.* △

Challenges in practice The main bottleneck for diagnostics is the data gathering part, which takes up to 75% of the overall diagnostic time. The main reason is that finding the right data for analytics is very hard due to limitations of predefined queries, complexity of data, complexity of query formulation, and limitation to explicitly stated information. In Fig. 7.2 we schematically depict the complex

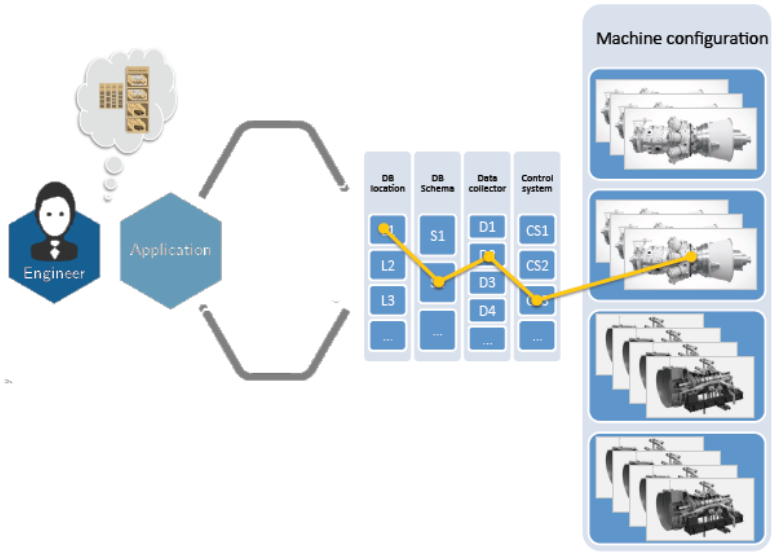

Figure 7.2: Current state-of-the-art for turbine diagnostics.

process of data access that requires to determine the right DB location, then the right schemata, and the corresponding data collectors and controllers deployed in turbines. Moreover, often diagnostic tasks involve up to dozens of turbines and thus this process should be done for each of them.

One example of Siemens turbine model (see Fig. 7.1 for example) has about 2,000 sensors constantly producing measurements. This data can be roughly grouped into three categories: (i) sensor and event data from appliances; (ii) analytical data obtained as results of monitoring tasks conducted by service centres for the last several years; and (iii) miscellaneous data, typically stored in XML, containing technical description of appliances, types of configurations for appliances, indicates in which databases information from sensors is stored, history of weather forecasts, etc. All in all the data is stored in several thousand databases having a variety of different schemata. The size of the data is in the order of hundreds of Terabytes,

e.g., there are about 15 GB of data associated to a single turbine, and they currently grow with the average rate of 30 GB per day. At the moment there is no unified access point to the Siemens data and it is required.

Existing predefined analytical tasks and workflows in the catalogue, about 4,000 workflows, are often not sufficient to cover information needs as they are often either too general, thus yielding an overload of irrelevant information, or too specific, thus not providing enough relevant analysis use-case. For gathering relevant data, service engineers often have to use several queries and workflows and manually combine their results. When this is not sufficient, existing workflows have to be modified or new workflows should be created. To this end the engineer contacts an IT expert and this leads to a complex and time-consuming interaction that takes up to weeks. The reason why it takes so long is miscommunication, high workload of IT personnel, complexity of query formulation, and long query execution times. In average up to 35 queries require modification every month, and up to 10% of queries are changed throughout a year. Moreover, several new workflows are developed monthly. Therefore, flexible modification and definition of workflows is one of the strong requirements for the improvement of the diagnostic process.

Predictive analysis requires the use of both static information from the past and streaming information on the current status of appliances. Access to historical data allows to detect, for instance, seasonal patterns. Continuous monitoring of the streaming data provides prognosis for key performance indicators and countermeasures before a system shut-down occurs. Currently, service engineers do not have direct access to streaming data. However, engineers often need to access event and sensor data from several appliances, and stream processing for each related turbine. One of the requirements for the predictive analysis is the possibility to integrate sensor and event data from several turbines and diagnostic centres and provide the use of analytical queries on such data sets.

In a nutshell, the current challenges consist of solutions structured per life-cycle stage:

- Combining different dimensions of data is time-consuming and requires highly specialised experts. This limits the level to which Siemens can leverage the vast value of service and operational data.
- Limited access to data between individual parts of the product life-cycle.
- Siemens R&D experts spend at least 75% of their time on data gathering and preparation.
- Lead time for R&D projects for component improvement is often 2 years or more per component per turbine type.

7.2 Case Description 2: Train Diagnostics

Trains (as in Fig.7.3) are predictable mechanical systems, which are controlled and driven by software that has been carefully designed to gather, store and transmit data deemed relevant to support train operation. Train software and maintenance processes define data structures. Our goal is to use the structure in Mobility data to improve data services at Mobility. Existing data-service systems are in place to 1.) receive raw data from the train, 2.) extract and transform the data into relational database tables, 3.) support statistical/machine learning analyses of the data and visualization of the results.

Figure 7.3: Siemens Train - Vectron and its structural components [119].

Challenges in practice There are two significant challenges in the current state-of-the-art solution. As a first, basic step, they solution lack full documentation of Mobility meta-data. For example, knowing what type data exist? How it can be analyzed? How do analysis change with software upgrades or configurations?

Consequently, the first challenge is to build interactive, browsable documentation, which can be browsed by all data service stakeholders to establish what data exists where, how it is linked and how it is analysed in existing data services. This is the meta-data management component of the data and analytical concepts.

The aim is to integrate heterogeneous data sets and find a logical relation between them. For example, mapping between relational data in the data warehouse and the data structure defined by train software. In other words, currently the data is available to data service interfaces only in a relatively flat, high-level relational structure. Detailed information about the data structures defined by the train software is available only through configuration experts by manual interactions.

For Example: Sensor data from a device is not as useful in isolation as it is with a certain context. A context could be data from related entities or data about a sensor environment. Context is required to answer questions like:

Example 10. *How is diagnostic code xyz from data source A related to code abc in data source B? Do they share triggering sensors?* △

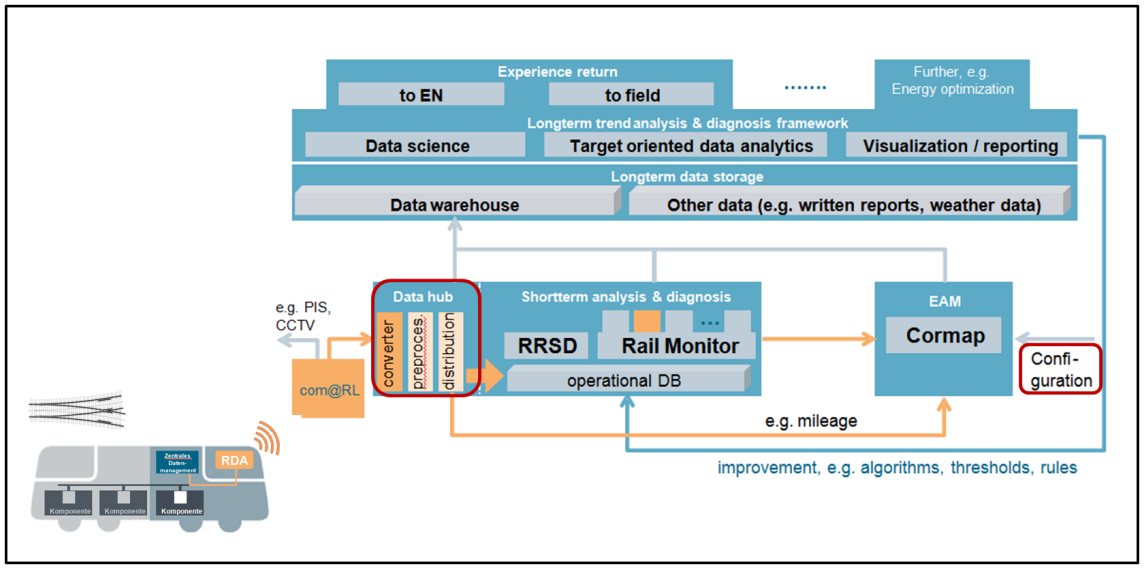

Figure 7.4: Downstream, shore-side IT Systems flatten the train-software defined data structures into relational databases, which provide only superficial links between data points.

While it may sometimes be possible to bootstrap a context from the data itself, e.g. through clustering methods, it is preferable to define the context explicitly through a logical model whenever possible. This is especially desirable when the data is generated mechanistically by machines that are explicitly configured to send data in a certain structure. To make data service output trustworthy and transparent, the data structure configured into the hardware machines must be carried forward to support the algorithmic output. Consequently, our goal in this case study is to bring configuration information out of the data structures defined by train software and up into a formal, logical data model that can be used to enhance analytical engine reliability and improve confidence in algorithmic output.

The second and most important challenge is to delve deeper into the knowledge models that are implicit in the train software, in engineering data and in maintenance management systems. Benefits of such knowledge models for Mobility include

- Improved transparency in the data value chain,
- Greater trust in analytical algorithm output
- Ability to synchronise train software modifications with algorithms
- Continuous improvement: a better understanding of how to develop train software to support data services in the future
- A basis for collaboration between train software engineers, data analysts and maintenance managers.

The state-of-the-art solution for analytics adopted at Mobility today try to recover the underlying knowledge models by a given set of diagnostic data in a flat, relational database system. This approach is limited by the fundamental fact that correlation does not imply causation. In the end there is no way to prove that a hypothesized relationship is true. The estimates are guesses that live in a confidence interval that must be quantified in order to estimate the dependability of data-service / analytical output down the line. To the extent that an algorithm also operates within a certain range of confidence, uncertainties about data-relationships begin to multiply with uncertainties inherent in analytical output. This will be a problem wherever we would like to rely on analytical algorithms to guide critical decisions where the error margin must be as small as possible (for example because an incorrect outcome may have safety-critical, commercial or reputational repercussions).

In our view, the only way to truly understand the causal relationships that lead to the data to be analysed, is to embed structural information from the train software directly into the data analytical systems. To do this, we must translate available data structures into formal domain models which can then be linked to the data.

Our proposal(see Figure. 7.5) is to lift structure from the train software configuration into an ontology and a triple-store. This structure glues existing relationally organized data tables and other data sources more tightly via a deeper underlying structure. Our proposed solution of consuming data and analytics through the ontology is is called Analytical-aware semantic system which is the topic of the thesis. The outcome of our solution for mobility is:

- Data and analytical machinery is linked without having to be moved,
- Data can be accessed and analysed via a trusted, structural model, which captures the mechanistically programmed relationships.

Details of the evaluations are present in the following sections.

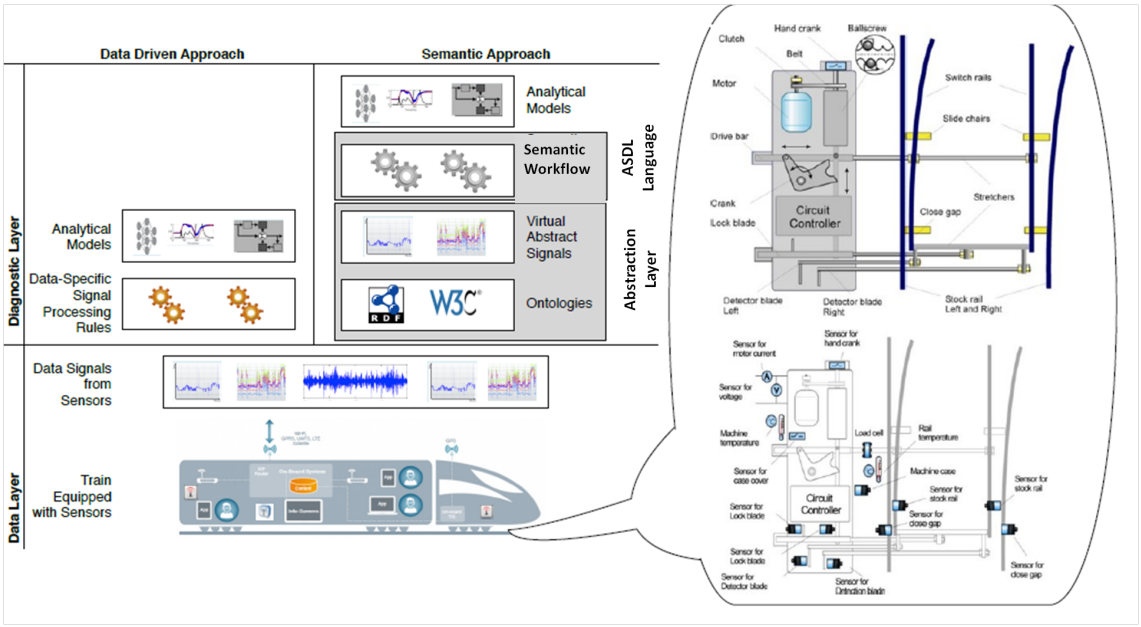

Figure 7.5: Industrial Analytical-aware Semantic Solution.

7.3 Case Description 3: Smart-grid Analytics

Smart Grid modernizes the traditional power grids with a two-way flow of electricity and informational resources. These help in creating a distributed energy networks as well as provide automated control over these networks. The real advantage of adopting such advance management of networks enables real-time monitoring and diagnostic services that ultimately helps in achieving the right balance of demand and supply at all levels of grid components.

Challenges in practice Today, the IT applications mainly manages the transmission and distribution of power. These are further sub-divided into discrete sub-applications (e.g. load calculation). These sub-applications employ data-intensive analysis and are treated individually. They capture a certain aspect of a grid (e.g. energy efficacy) by utilizing corresponding models (e.g. linear regression) and multidisciplinary techniques such as machine learning, deep learning, etc. Likewise, it requires integrated skill set from diverse fields, including, mathematics, statistics, and machine learning, and domain knowledge to craft an individual analytical task and to manage it. This means that for a different aspect of a grid, an analyst can devise a variety of analytical tasks that would entail different data, modelling techniques, and algorithms. An important class of such tasks that are commonly used in practice allows

- filters, aggregates, combine, and compare signals coming from sensors installed in a grid component and
- fire notification messages when a certain pattern in signals is detected.

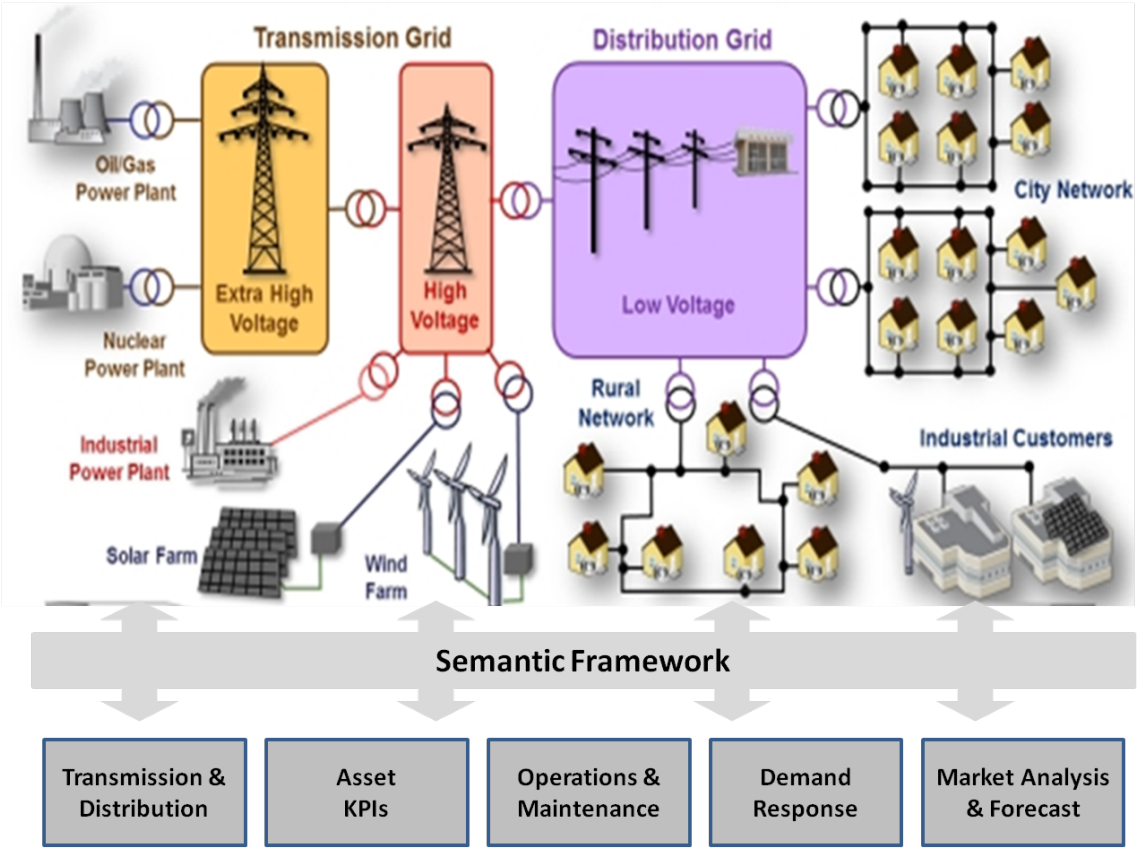

Figure 7.6: Smart-grid Snapshot [120].

The main challenge for automating analytics in most modern industrial grid system are highly data dependent in the sense that specific characteristic of individual sensors and pieces of equipment are explicitly encoded in the application program. As the result for a typical task engineers have to write dozens of programs that involve hundreds of sensor ids, component codes, sensor and threshold values as well as grid configuration and design data.

For example, a typical base load power station has about 3000 sensors and a typical task to determine its base load power output on demand requires around 160 signal processing rules, most of which are similar in structure but different device specific data values.

Thus, there is a need in industry for a higher level semantic language that allows to express what the analytical task should do rather than how it should do it for specific grid component. Such language should be high level, data independent, while powerful enough to express in a concise way most of typical analytical tasks.

Our proposed solution address the above mentioned challenges and ease the interaction of grid components. In particular we rely on ontologies to define a novel analytics-aware language and on reasoning over ontologies to foster execution and

maintenance of analytical tasks. In a nutshell, we achieve the data access and interoperatability of signal processing rules by providing:

- a semantic language that treats aggregations and other analytical functions 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-driven grid programs that combine functions with grid knowledge captured using ontologies and allow to express complex tasks in an abstract fashion by exploiting both ontological vocabulary and queries over ontologies to identify relevant information (such as sensor ids, subsystems and set point values) about the grid components and devices.

Our proposed solution provides all these services mentioned above and have been evaluated on publicly available data set from Government of Texas. Details of the evaluations are presented in the following sections.

7.4 Evaluations

In this section, we present five different set of evaluations of our approach over three industrial sectors. The first set of evaluation is ontology model related evaluation of our proposed domain ontology. The goal of the evaluation is to verify the applicability and reusability of our ontology models across different industrial use-cases.

The second and third set of evaluation analyzes the efficiency of our proposed ontology language and sytem compared to the state-of-the-art solutions and establish convincing results for our approach.

The fourth set of evaluation is conducted to analyze the effectiveness of our approach in terms of reduction in effort. We produce results for time it takes to make data validation and query answering services accessibility to the domain experts using our approach versus the state-of-the-art.

The fifth set of evaluations is runtime analysis of our approach versus the state-of-the-art solutions to show the performance improvements in formulating analytical task and workflows using our *SAL*.

7.4.1 Evaluation of Ontology Models

We assess the quality of our proposed solution by checking how good our ontology covers the case study data sets and then how good our mappings cover the terms in the data.

Confidence and Coverage One of the most important quality check for any ontology is the confidence and coverage that reflects how well the ontology represents the domain it models and its data is mapped to.

Method: To evaluate the coverage of the data by the ontology, we developed an alignment method that comprises of two steps. Firstly we find a syntactic match of the ontology terms with the domain data set. Secondly we perform a structural comparison of neighbourhoods around these terms that have a syntactic match. The alignment results in a set of pairs of matched terms together with a score showing how well they match. For this purpose we used and extended popular approach of an ontology alignment system LogMap. It was extended to perform both syntactic and structural matching of ontologies together with the required alignment of ontologies and data. The main challenge was to define the notion of a structural neighbourhood of a data term in a set of conjunctive queries.

However, to meet our experiment requirement, we introduced the following notion: given a set of data $\mathcal{D}$ and a term t, its neighbourhood in $\mathcal{D}$ is the set of all terms $t0$ occurring in some $\mathcal{Q}$ belongs to $\mathcal{D}$ that contains t, together with the surrounding sequence of terms that is common in all such $\mathcal{Q}$. We performed the coverage assessment separately for the state-of-the-art ISO 15926 and our developed TechOnto ontology from each of the three datasets i.e. turbine, train and smart-grid. Finally, together with three domain experts we performed a manual assessment of each matching for classes of each domain-specific data set. The manual assessment contributed to the class matching that are correct from the domain expert point of view. We termed such cases as true positives. The incorrect class matching are termed as false positives. The class which are ambiguous and where domain experts did not reach a conclusion are termed as semi-positives.

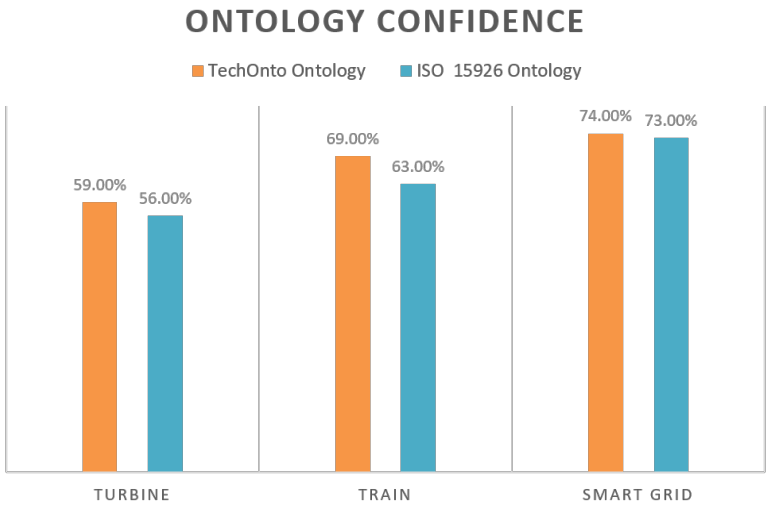

Figure 7.7: Confidence evaluation of the Ontology w.r.t three data sets represented.

Results: The results of the matching are in Figure 7.7, it describes the confidence of the data terms by ontologies: the three show the coverage of classes by, respectively

ISO and TechOnto ontologies. The results are comparable of both the ontologies because of high generalization of the ontology terms together with the support of synonyms. The results of our manual assessments are also in Figure 7.8. For example, manual assessment of coverage of turbine classes by the ISO ontology gave 15% of true positives (they are correct for domain experts), then, 24% of semi-true positives, and 20% are false positives (the matches are wrong for domain experts). In the case of our ontology, ones that were matched to the data terms are 31% of true positive for turbine data sets. In case of trains, TechOnto gave 23% of true positives as compared to 18% of correct matching for ISO ontology. Also the false positives are of higher number for the state-of-the-art ontology. The same scenario resulted for smart-grid use-case. The reason behind a good fit of TechOnto ontology is use of high level of generalization especially in case of system and component hierarchies and analytical concepts. Most of the smart grid and train data set reflected such relations. However, the coverage can further be improved by enhancing the model for geo-spatial and information entity related concepts such as documents, plans. Nevertheless, it is important to note that the matching results are highly depended on the type of selected data sets and domain expert competence.

Accuracy Accuracy is yet another important evaluation criteria to assess the quality of the ontology model. It determines if the asserted knowledge (i.e. the mapped data) in the ontology agrees with the expert's knowledge about the domain. A higher accuracy typically results from correct definitions and descriptions in the ontology data.

Method: Typical the concept of error rates such as word or concept-error rates are used to determine the accuracy of an ontology model. In our work, we also determine error rates for each of our use-case driven task evaluation. Following are the definitions for error rate in our scenario:

- Superfluous concepts e.g. is-a and semantic relations are considered as insertion errors.
- Missing concepts, is-a and semantic relations are treated as deletion errors and
- Off-target or ambiguous concepts are known as substitution errors.

Given appropriate tasks which are basically defined as semantic queries posed by the domain expert and maximally independent query translating algorithms operating on the ontology in solving these tasks. Table 7.9 presents an overview of error rates against which we determine our accuracy index.

According to our definition, we devised an approach stated as follows:

- One or more ontologies can be evaluated against a given user query and their response in terms of performance can be determined.

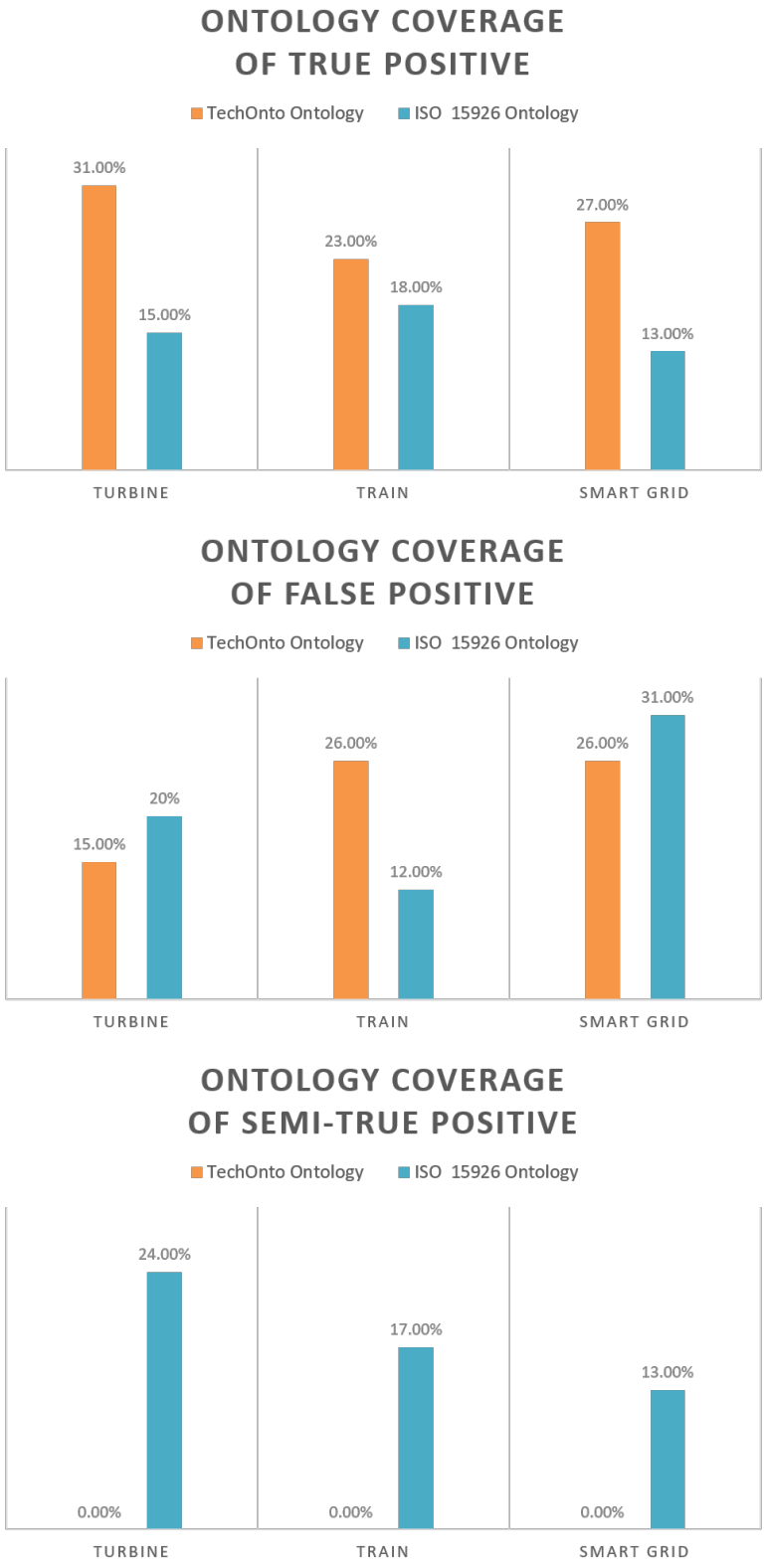

Figure 7.8: Coverage evaluation of the Ontology for true positives, false positives and semi-true positives.

level	insertion	deletion	substitution
1	irreverent concepts	omitted concepts	ambiguous concepts
2	isa too coarse	isa too fine	isa too polygamous
3	irreverent relations	missing relations	indirect relations

Figure 7.9: Overview of the error rates.

- Then based on the query results, one can count the insertion, deletion and substitution errors,
- Based on error rates , one can improve the ontology, and
- Later re-evaluate the query results with the improved ontology which ultimately should improve the performance.

We evaluated our proposed ontology model using the error rate definitions as described above and in Table 7.9. If correct relation was found against the corresponding concept in the ontology, we mark it as accurate match. For counting the inaccuracies, we counted the semantic relation error rates as described in Table 7.9.

The accuracy is defined as the number of correctly classified instances and is computed as the total number of instances minus the total number of inaccurate matches, where inaccurate matches belong to deletion, insertion and substitution error types.

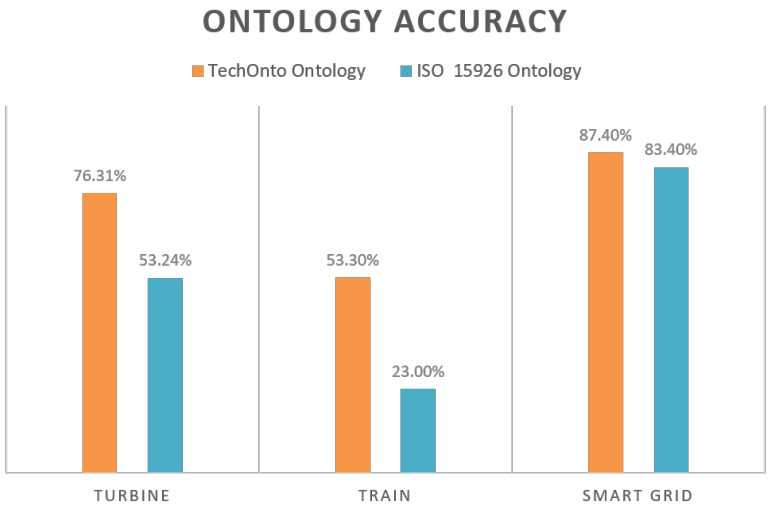

Figure 7.10: Overview of ontology model accuracy of TechOnto and ISO 15926.

Results: As compared to the ISO 15926 standard we obtained the accuracies as shown in Table 7.10. The 53.24% for turbine case indicate clear cases where a pertinent (at least for some queries) relation was not modelled in the ontology. Whereas for train and smart grid are accuracies are better for TechOnto ontology. One can easily improve the model by populating the improved model and re-evaluating.

However, this corresponds more to an engineering perspective where bootstrapping approaches are popular. However, it can be semantically weak model and less expressive. A more significant and interesting outcome of such ontology evaluation concerns a methodology to make such a schema more general and scalable across different use-cases. It is our aim to demonstrate that such evaluations are hard to determine the quality of the model in general. Domain experts can easily drive here to build an accurate model by annotating domain descriptions and their view of the world. In a bigger picture, one or more domain ontologies can be aligned and / or reuse fragments from other models to improve interoperability and reusability. Thus, we conclude that our proposed TechOnto ontology has better accuracy than ISO standard ontology as it accomplishes the domain-specific queries better and can be aligned across multiple domain models.

Precision and Recall Precision is defined as total number of correctly found over whole knowledge defined in ontology, whereas Recall is total correctly found over all knowledge that should be found. We consider for all our three use-case sets, their data description Gw and semantic expand set Gs in order to evaluate our proposed ontology and get the precision and recall index.

Method: We use the state-of-the-art ISO 15926 ontology to conduct the semantic search on ontologies and determine its effects on different ontologies. We used the information sources in case-study section to ensure knowledge consistency. Here, natural language processing algorithm has been employed to retrieve triples for the use-case data sources. Then we merge the synonyms in tuples based on their linguistic similarity of vocabularies and finally we get the connected ISO ontology. We created a sample test set of 150 data instances from each case study data sources and defined them as searchable objects. According to statistical analysis, more than 95% of the users usually enters $1\tilde{4}$ keywords to search an object. $1\tilde{3}$ keywords in each test set was sampled again to be imported in both our proposed TechOnto ontology as well as ISO ontology. Then we build subsumption hierarchies within $2\tilde{3}$ jump and associated direct properties with keywords in ontologies. This way, we were able to develop two semantic expanded sets of our TechnOnto ontology and ISO 15926 ontology. Then we collect all vocabularies in one data description as Gw and calculate the precision and recall.

Results: We calculated the average values of 150 data instances, and determine their precision-recall indexes as presented in Table 7.11. Based on Table 7.11, we got the following conclusion. We proved that if the same ontology is used then the precision remains the same and this is because of increase in keywords. The precision of TechnOnto ontology is 11% higher than that of ISO 15926 ontology in turbine use-case. Similar results are found in case of recall, that increases with increase in number of keywords. However, in the case of recall there is a slow decrease in growth. Nevertheless, we found that both ontologies have constant recall in cases where we use same number of keywords. Results are different in case of precision. Due to the use of linguistic similarity, some irrelevant concepts were inherited. For example:

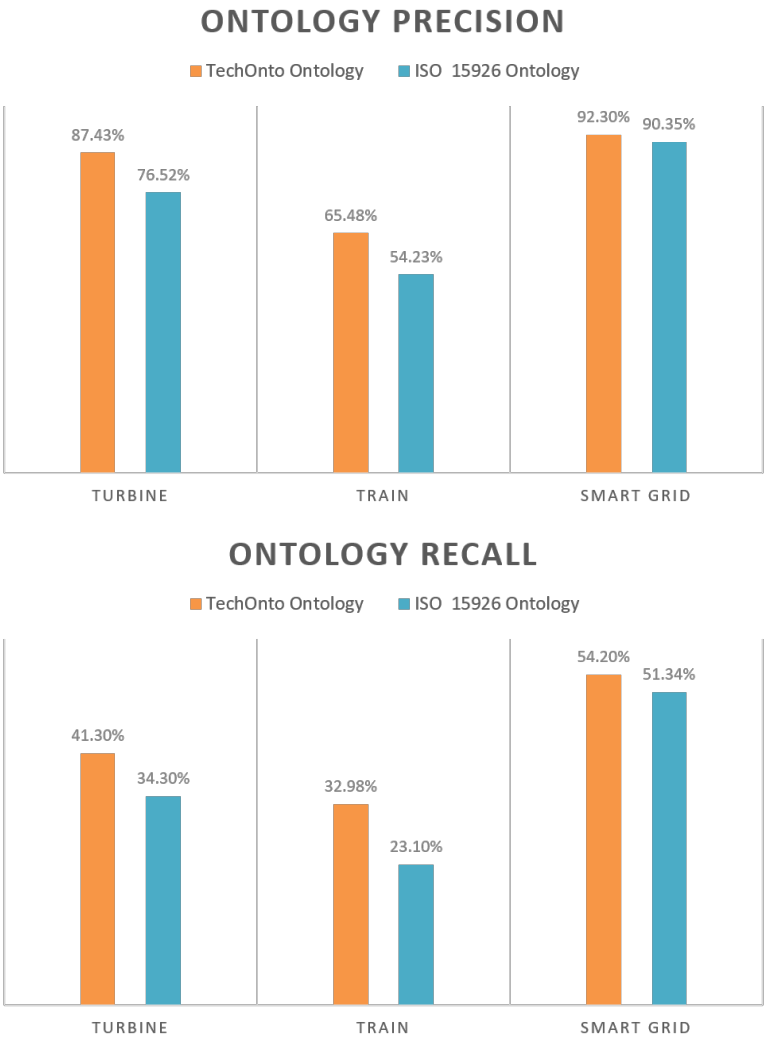

Figure 7.11: Overview of precision and recall of keywords from TechOnto and ISO 15926.

device-has-name-is-sensor and location-has-name-is-china. The correct inheritance should be device-has-name, name-is-sensor, location-has-name, and name-is-china. Now, because of linguistics based merging, these concepts *name* then the *sensor* inherited *device* and *location*. This is the sole reason in decline of precision index. Whereas, because of increase in parent-child concepts, we get an increasing recall.

7.4.2 Evaluation of Ontology Languages

In this section, we evaluate features of the state-of-the-art ontology languages against our proposed language that are relevant w.r.t. data integration, analytical processing and domain-specific semantics. Table 7.12 summarizes the results of our evaluation. Details of each language is discussed as follows:

In order to assess our proposed language capabilities and functionalities, we have

	CQL	CQEL	C-SPARQL	STARQL	DatalogMTL	SAL
Data Model	RDF streams	RDF streams	RDF streams	RDF streams	RDF streams	RDF streams
Union, Join, Optional, Filter	Yes	Yes	Yes	Yes	Yes	Yes
IF Expression	No	No	Yes	Yes	Yes	Yes
Aggregate	Yes	Yes	Yes	Yes	No	Yes
Analytical functions	No	No	Yes	Yes	No	Yes
Property Paths	No	No	Yes	No	No	No
Time Windows	Yes	Yes	Yes	Yes	Yes	Yes
Historic data	No	No	No	Yes	Yes	Yes
Execution	RDF stream processor	RDF stream processor	DSMS based evaluation with triple store	external query processing	external query processing	external query processing
Query Optimization	Adaptive query processing operators	Adaptive query processing operators	Static plan optimization	Static algebra optimizations, host evaluator specific	Static algebra optimizations	Static algebra optimizations
Stored Data	Stored linked data	Stored linked data	Internal triple store	Data source dependent	Data source dependent	Data source dependent
Reasoning	No	No	RDF entailment	Yes (DL-Lite)	Yes (DL-Lite)	Yes (DL-Lite)
Maturity	High	High	Low	Low	Medium	Medium
Event Pattern Expressiveness	High	High	Medium	Medium	Low	High
Conceptual Coherence	Low	Low	High	Medium	High	High
Dynamic Rules	Low	Low	High	High	High	High
Heterogeneous knowledge sources	Medium	Medium	Low	Low	Low	High

Figure 7.12: Comparision of state-of-the-art and our proposed language *SAL*.

extended and conducted our comparative analysis for all relevant semantic languages (that we are aware of). The results are shown in Table 7.12. To our conclusion, generally all these languages are supporting basic functionalities like union, join, optional and filter, some of them (including STARQL) have already incorporated SPARQL 1.1 expressiveness with IF clauses, aggregations, arithmetic expressions and more. In addition to this, all of them have limited support of temporal windows which is an important feature while analyzing signal data streams.

The table presents specific streaming capabilities as well as operators of each semantic language. Here, we can distinguish between two distinct groups of query languages that perform differently in the management of time and of temporal concepts and their operators. One of the first group of languages allows access to timestamps by functions on each triple or objects within windows. Such languages include C-SPARQL, and STARQL. However, STARQL is a non-reified version with a semantics of temporal concepts and states whereas C-SPARQL uses an in-between approach offering temporal functions on objects for retrieving their timestamps. C-SPARQL could generally easily lead to inconsistencies, in cases where an object occurs several times inside a window in different temporal states.

On the other hand, there exist group of languages that are developed to cater temporal sequences and their specific sequencing operators. Such languages include datalogMTL, STARQL and our language *SAL* that can easily support applications of complex event processing (CEP).

With recent adoption of Semantic Technologies, these languages offers several new operators with functionalities that are not supported by any other state-of-the-art language formalism. For example, they support functionalities of querying historic static data as well as support comparision operators to analyze live data streams. datalogMTL is capable of synchronizing different kinds of input streams (by using

different kinds of window widths and slides) by using one or more pulse functions. This allows to have a a regular query output for possibly asynchronous input. Moreover, with an integration of optimized UDFs to ExaStream (such as an optimized version of the correlation function), STARQL and our language offers foundational feature set to implement analytics driven OBDA approach. Most of semantic languages mentioned above rely on native implementations of query processors. For example, CQELS reimplements functionalities, which do already exist in DSMS and therefore can be seen as standalone engine. Whereas, an internal DSMS is supported by C-SPARQL, but it has no feature for RDB to RDF mappings or query rewritings.

The only two ontology languages using an OBDA approach with mappings and a extensible backend are datalogMTL and STARQL. They both suffer the same disadvantage because they both rely on external DSMS. Query rewriting and translation of results can be expensive, while the expressiveness of the underlying systems restricts the input of the RDF streaming queries. Nevertheless, OBDA approaches can rely on various back-end optimisations to accelerate query processing.

CQL and CQEL are very popularly used in systems equipped for event processing. However, both are active open source languages that are well received in market and have reached a stable state. The language is also supported with detailed user guides, tutorials and comprehensive documentation. On the other hand, C-SPARQL is still in its initial phases. The language is not mature enough and lack of documentation makes it difficult to use. There are very rare occurrences in literature that talks about adoption of C-SPARQL, STARQL and datalogMTL languages in a real-world use-cases. However, in this thesis, we considered extension to datalogMTL as well as provide real world examples of its use in industrial sectors. Our solution language is deployed for Siemens businesses and is part of a product already, therefore we consider the solution mature enough.

We can easily conclude that CQL, CQEL and *SAL* is mature and acceptable enough to provide a rich set of operators for example, to construct event patterns. These languages support different types of temporal constraints along with sliding windows as a reasonable set of aggregation functions. In constract, C-SPARQL, STARQL and datalogMTL are less expressive but general event processing tasks can also be defined using these languages.

C-SPARQL, STARQL, datalogMTL and *SAL* allows the processing of static and streaming data as well as the integration of static domain knowledge by using only a single language construct. Its queris are able to combine event stream processing and SPARQL queries together using a single interface. In this sense, a their query language is self-contained and coherent. Users only require basic semantic query language skills to formulate and execute tasks. In comparision, the languages such as CQL and CQEL do not allow easy access to the domain knowledge bases. A supporting Java/Jena application must be programmed in order to integrate the domain knowledge. A typical example is where a CQL-based architecture is developed to

combine the query language (EQL) for stream processing and Jena/SPARQL code written in a Java adapter class to retrieve knowledge bases. In such cases, most of the state-of-the-art languages are not self contained and are highly dependent on programming logic of the adapter classes.

The main disadvantages of state-of-the-art languages is their inefficiency to change a rule at runtime. This is difficult to manage because a change can lead to a change in pattern / template as well. Use of SPARQL query makes such changes easier to manage. This is because SPARQL queries can be stored as strings in a separate file and can be reloaded at runtime.

Languages such as C-SPARQL, STARQL and datalogMTL are also restricted in a sense that they store the ontological background knowledge in RDF format. Whereas, CQL can be used and adapted for arbitrary adapters allowing the usage of different knowledge sources. However, the different data connectors have to be implemented and maintained by hand which is a costly operation.

Analytical reasoning is not supported by CQL language. It is incapable of deducing new knowledge automatically from a given knowledge base. C-SPARQL, STARQL, datalogMTL and *SAL* provides reasoning capabilities that are sufficient enough to cater many challenges queries.

Considering the given approaches from a conceptional point of view, our ontology language is better suited for inherent reasoning, operating on heterogeneous data sources and providing analytical operators and workflow management . For instance, SPARQL with analytical operators and RDF entailment can be achieved by using materialization or query rewriting.

7.4.3 Evaluation of Semantic Systems

In this section, we evaluate various adopted state-of-the-art systems of semantic web technologies used in industrial and software engineering research communities for a comparison with our implementation of the analytical-aware semantic system. All these analytical models and languages are open source and rely on an underlying meta model, e.g. Etalis comprises of an analytical engine in order to support and define user driven monitoring rules. In this set of evaluation, we consider comparing different semantic systems in their approach to define meta models and its overall system implementation.

***SAL*, TechOnto and SOMM:** As presented in this thesis, we propose to use a TechOnto ontology in combination with analytical-aware semantic language and SOMM as model editor. An application independent analytical ontology language is used for modeling of various analytical models and tasks. This analytical meta model is stored in our semantic system and can be adapted by the domain experts.

	SWRL + Protege editor	Java + Eclipse editor	Etalis + Prolog editor	SAL + TechOnto Ontology + SOMM
Application dependency	no restriction regarding application	no restriction regarding application	no restriction regarding application	no restriction regarding application
Adaptability	missing reification functionality, thus no adaptability	missing reification functionality, thus no adaptability	missing reification functionality, thus no adaptability	reification functionality is supported and can be adapted
Integration	relying on the same basis as technical ontology models, their knowledge can seamlessly be integrated	knowledge of the technical ontology models can be processed by dedicated APIs such as Jena	no connection between Prolog and SPARQL endpoint of SMW available	seamless integration since technical ontology relies on the same basis as other available reference models
Modeling patterns and libraries	libraries can be reused, but templates for modeling patterns do not exist	rule related modeling patterns and libraries could be established, but do not exist	no object-oriented programming approach, thus no templates or libraries	reusable libraries for rules and rule related modeling patterns are available
Consistency checks	monitoring rules without constraint definition, thus no consistency check possible	hard-coded checks, which are not automatically part of the model	neither object-based nor plant related consistency checks available	object-based and comprehensive domain-specific related consistency checks possible
Maintenance of analytical workflows	due to missing reification, no mechanism for semantically labeling or structuring of workflows	no advanced maintenance inferences considering the similarity of monitoring conditions	due to missing reification, no mechanism for semantically labeling or structuring of workflows	analytical ontology for semantically labeling and structuring of workflows

Figure 7.13: Comparision of state-of-the-art and our proposed analytical-aware semantic system.

The system also provides seamless integration is possible by linking from the ontology model into other reference models like sensor ontology, qudt ontologies etc. The system also provides guidance for the experts during the engineering of monitoring and diagnosis rules by reusable analytical libraries and rule related modeling patterns specified with the Semantic descriptions of a semantic workflows. The correct structure of the monitoring workflows is verified during the engineering phase. OWL Reasoner is used to execute and provide inference for the analytical ontology as well as to maintain quality in form of consistency checks. However, we can also verify the consistency of the ontology model against the instance data by defining e.g. SHACL rules, SWRL rules etc. One can also verify if the mappings are linked in a correct way i.e. the way ontology concepts are linked to the source data sets. Additionally, the domain experts can use the concepts of the TechOnto model to manually construct taxonomies of their assets, monitoring conditions or rules and to execute specified analytical and monitoring tasks by means of OWL reasoning.

SWRL + Protege: A popular rule language of the semantic web community is SWRL and it is important to draw a comparision of such semantic system with our proposed solution. SWRL is an application-independent rule language whereas the domain-specific constraints and the background knowledge of these constraints are seamlessly formulated and integrated in form of rules. However, an important drawback of using such rule system is its inability of reifying rules because there exist a signification gap in linking the terminological and its assertional resources. In addition to this, an important disadvantage of such system is lack of support of analytical functional libraries or templates where users can define patterns or run consistency checks. Due to the missing reification ability, there is also no mechanism to maintain semantic patterns, custom functions or either formulation of analytical workflows.

Java + Eclipse: Another plausible state-of-the-art system to define monitoring, analytical and diagnostic tasks to use them in an hard-wire programming language such as Java. We used Eclipse as generic editor and managing tool for defining Java based rules. Similarly to SWRL and others, such systems do not support adaptations to domain models to formulate analytical tasks or workflows in general.

However, it is an important feature to consider that Java allows dedicated APIs that can possibly process the domain-specific knowledge stored in any semantic system and integrate the knowledge in the monitoring and diagnosis tasks and workflows using API calls. However, it could be an absolute nightmare to maintain and manage such workflows. In addition to this, because of no predefined libraries for temporal operators or analytical-related modelling patterns repositories, such extensions have to be manages manually which is a high maintenance job. In regards to consistency checks, Java can handle pretty fairly but the algorithms need to be explicitly defined and run separately than a formulated analytical workflow. It can partially provide an ability to provide maintenance information about analytical workflows. However, similarity search of such monitoring rules or any advance reasoning cannot be supported by any automated means.

Etalis + Prolog: An important state-of-the-art that is widely used in semantic web community is Etalis configured with SWI-Prolog editor. Etalis is mainly a complex event processing rule engine that is application-independent. A major drawback of Etalis is its hard-coded reification functionality. As a result of which, mechanisms for semantically labelling or structuring analytical tasks and workflows can not be supported. Furthermore, domain-specific knowledge in form of RDF triples or OWL axioms cannot be integrated in the workflows. This is due to its interface restrictions that does not allow to establish connection ports between Prolog and the TripleStore endpoint. Etalis is not an object-oriented programming system, which means that specifying rule libraries, templates or consistency checks algorithms are harder or even impossible to implement and manage.

7.4.4 Evaluation of Effort

In this section we will verify whether our semantic approach to data analytics can offer a considerable reduction of effort in terms of time as compared to the state-of-the-art solutions. Here we will consider examples from our three use-cases and evaluate if the semantic approach was effective in reducing the effort in data access, data validation and query answering.

Effort based Analysis of Turbine use-case

Here we take an example from turbine use-case where unexpected damages discovered during on-site inspection often need to be reviewed by highly skilled experts. Reaching them and getting an evaluation is expensive and time-consuming. The challenge today is to utilise data from large installed base to collect all historical observations for each serialized component. For each, create a statistical model to judge the remaining life and risk of continued operation. Use this model to drive an on-line analytical system.

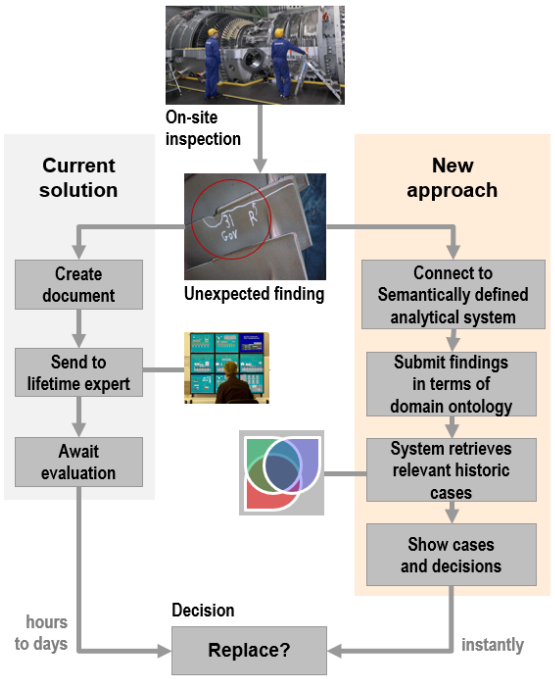

Figure 7.14: Onsite Inspection of Turbines

In the example see Fig.7.14 when an engineer discovers a 3mm crack at the base of the guide vane of the first compressor stage, he connects to the our semantic system and submits a problem description based on a standardised terminology using our domain-specific semantic model. The system responds by showing 5 cases of cracks in the same turbine type at the same location with a similar application context using reasoner and provide failure predictions of the current crack. The engineer reviews the analysis and historic decisions and decides that the cracked guide vane should be replaced.

Through the use of this semantic intelligence, the turbine value chain can be optimized by learning from past failures and suggest repair activities, propose upgrades for more power with the same reliability to reduce 20% of the service lead time and

Effort	*Existing Solution*	*Our Semantic Solution*
Query Formulation	40%	20%
Data Retrieval	30%	20%
Analytical Model Building	10%	20%
Analytical Model Execution	5%	10%
Analytics Deployment & Visualization	15%	30%

Table 7.1: Effort based analysis of existing solution versus our semantic approach for turbine use-case

improve the planning cost and by automatically predicting the KPIs of the top 10 most vulnerable power plants. This requires the integration of huge data streams retrieved from monitoring the turbine with unrelated data silos (such as operational machine data, maintenance logs, failure catalogs etc.), and experience-based information and statistics derived from sophisticated AI algorithms, as well as domain knowledge gained through past successful failure handling. However, the integration of huge amounts of heterogeneous data streams with the knowledge modeled in the knowledge bases and the results of the AI algorithms is not hard to achieve. The challenges consist of: R2.1: The modeling of the semantics models to integrate all the data. R2.2: The ability to extract the annotations and learn domain-specific rules automatically from event logs/data streams. R2.3: Integrating the analytical aspects in the knowledge-based techniques. R2.4: The decision to distribute certain tasks closer to the sensor streams or perform the analysis centralized in order to optimize cost-efficiency. Our use case is characterized by data from more than 2250 industrial gas turbines, more than 10TB of operations data that grows more than 2TB each year.

Effort based Analysis of Train use-case

The Mobility Data Warehouse is the default access point for Mobility data. More and more data is being integrated, including from complex sources. As the datawarehouse (DWH) schema grows, it is important to have independent management system of the DWH for quality control and sanity checks. A semantic solution to the DWH has proven as a potential candidate to link the DWH to external systems to boost analytics without the time consuming process of full data integration. In addition to this, there is often a gap between the design and development of data service technology (its development and testing) and its use in practice. A semantic approach to closing the gap would involve mapping out the workflows in maintenance operations that involve data into a semantic-based structure. Specific reports/algorithms/queries can then be attached to semantic solution. As a maintenance engineer *traverses* his segment of the workflow, he can see/consume/develop the data services that are directly relevant.

In the example, see Fig7.15 a maintenance engineer is able to generate troubleshooting reports instantly by using our semantic approach. He queries the system with his available maintenance related data which are mainly diagnostic codes available for the required machine using our domain-specific semantic model and automatically retrieves the relevant analytical workflow to execute which are relevant to the given diagnostic codes. The workflows here are now automatically embedded with the data for this machine and diagnostic code configuration data. With few clicks he can either adjust the workflows, create new ones or reuse the whole or segment of the available workflows at hand. After execution he can himself conclude the discuss and submit the troubleshooting report to his customer.

Through the use of this semantic intelligence, an ontology manages/defines/stores

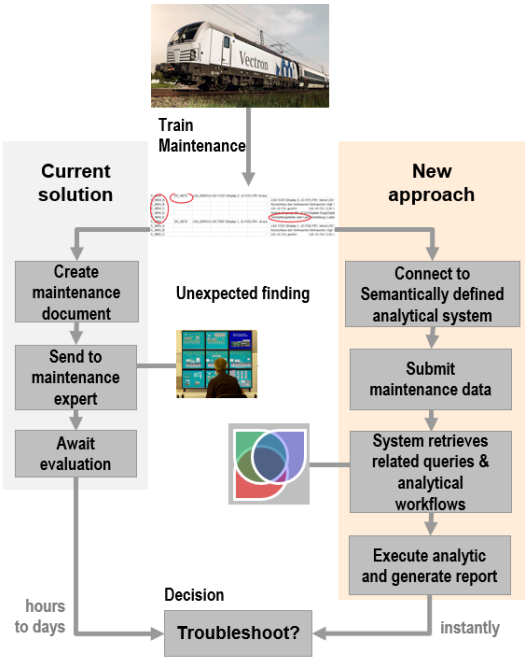

Figure 7.15: Maintenance of Trains

Effort	*Existing Solution*	*Our Semantic Solution*
Query Formulation	40%	20%
Data Retrieval	30%	20%
Analytical Model Building	20%	20%
Analytical Model Execution	5%	10%
Analytics Deployment & Visualization	5%	30%

Table 7.2: Effort based analysis of existing solution versus our semantic approach for train use-case

existing relationships defined in a DWH relational schema. It also includes relationships to data that is not yet in the DWH. Data can then be queried against the many systems via the ontology. The great advantage of this approach is that we can integrate new data sources on the fly without having to bring them into the DWH. The OBDA approach provides a *loose* semantic integration system integration which can be tightened by *hard* integration into the relational database model whenever a use-case requires high-performance transactional access to data. Another advantage is improving the reliability of analytics by including mechanistic links between data and analytical workflows. This approach assist colleagues at Mobility by building a feedback loop from configuration to the field. We aim to configure our semantic solution to make it more useful analysis to the maintenance engineers.

Effort based Analysis of Smart-grid use-case

North American market is moving towards Transactive Energy Market concept *"Transparent energy prices enable customers of all sizes to join traditional providers in producing, buying, and selling electricity using automated control to drive reliable and cost-efficient electricity system"*[121]. From the technical point of view we want data analytic platform that leverages the domain knowledge semantics as well. However, building an energy production cost modelling platform that takes into account various heterogeneous data sources (weather, plant, load, fuel price, generation, etc.) and produces forecasting of energy price and find optimal economic dispatch of energy generators (optimal output powers of generators to minimize total generation cost in the system) which is a non-trivial task.

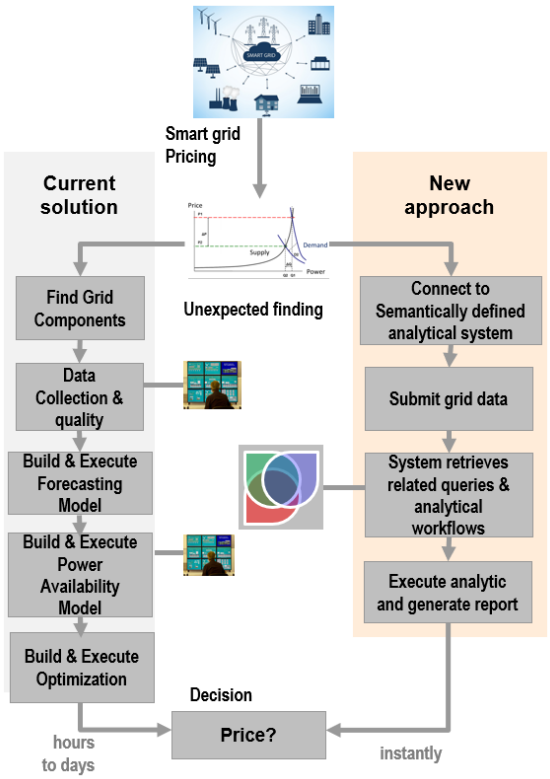

Figure 7.16: Price Forecasting in Smart-grid

In the example, see Fig7.16 a financial investor or analyst is able to generate reports instantly by using our semantic approach to analyze different scenarios and come with decision on their position in the market by optimizing their assets to get maximum turn over (Long term and Short term). He is able query the data that is collected from various heterogeneous sources: weather, power plants, fuel prices, operational, generation and load data, etc. This raw ingested data was cleaned and transformed into the format that is suitable for analytics.

The semantically generated analytical workflows for forecasting, power availability and cost optimization can be easily executed. The forecasting workflow contains

Effort	*Existing Solution*	*Our Semantic Solution*
Query Formulation	50%	20%
Data Retrieval	20%	20%
Analytical Model Building	20%	20%
Analytical Model Execution	5%	10%
Analytics Deployment & Visualization	5%	30%

Table 7.3: Effort based analysis of existing solution versus our semantic approach for smart grid use-case

further workflows to be executed. Particularly load forecast workflow that makes a prediction of energy load values; solar and wind forecast workflows that make predictions of solar and wind power generations. Asset management workflow that computes KPIs and availability of power plants and optimization workflow that optimizes the economic energy dispatch and price. In the final step a visualization workflow (Tibco Spotfire) was created to visualize data and analytics results.

Through the use semantically generated workflows, we concluded that the implementation costs and maintenance effort is reduced by half by using such a flexible semantic architecture. We were able to support for decision making in uncertainty and poorly maintained measurement environments and improve planning accuracy. The customers acknowledge the reduction of effort with respect to clarification and dispatch, also our approach helped in speeding up the data examination and decision making on a large amount of data at once.

7.4.5 Evaluation of Runtime-based Analysis

In this section, we present comprehensive runtime analysis of our approach over three case studies described in the previous section.

We present experiments to verify whether writing analytical task in *SAL* offers a considerable runtime saving comparing to formulating analytical functions in the any data dependent language or tool.

Runtime Analysis of SAL

To evaluate the runtime of our semantically defined analytical language, we conducted three case studies in different domains: *train diagnostics*, *turbine diagnostics* and *smart-grid diagnostics*.

#	Use-Case	Age	Occupation	Education	Sem. Web
TrP1	Train	34	R&D Engineer	MSc	yes
TrP2	Train	32	R&D Engineer	MSc	yes
TrP3	Train	47	Diagnostic Engineer	PhD	yes
TrP4	Train	45	Software Engineer	MSc	yes
TrP5	Train	34	Software Engineer	BSc	yes
TbP1	Turbine	43	Design Engineer	PhD	yes
TbP2	Turbine	46	Senior Diagnostic Engineer	PhD	yes
TbP3	Turbine	37	Diagnostic Engineer	MSc	yes
TbP4	Turbine	45	R&D Engineer	MSc	yes
TbP5	Turbine	34	Software Engineer	BSc	yes
TbP6	Turbine	33	Data Scientist	PhD	yes
GrP1	Smart Grid	34	Diagnostic Engineer	PhD	no
GrP2	Smart Grid	32	Diagnostic Engineer	PhD	no
GrP3	Smart Grid	41	R&D Engineer	PhD	yes
GrP4	Smart Grid	43	R&D Engineer	PhD	yes

Figure 7.17: Profile information of participants.

To this end we found 16 participants from Siemens, 5 for train diagnostics, 6 for turbines and 5 for smart-grid all of them are either engineers or software engineers. In Figure 7.17 we summarise relevant information about the participants. All of them are mid age, most have at least an MSc 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 5 participants never saw an editor for diagnostic rules, while the other 4 are quite familiar with rule editors.

During brainstorming sessions with Siemens analysts from energy and mobility departments as well as with the R&D personnel from Siemens Corporate Technology we selected 4 analytical tasks for trains, 5 for smart grid and 10 for turbines; they can be found in Figure 7.21. 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.

Before the study we gave the participants a short introduction with examples about diagnostic workflows and message rules in both Siemens and *SAL* languages. We also explained them the constructs of *SAL*, presented them our diagnostic ontology, and explained them the data. During the study participants were authoring analytical workflows for the tasks from Figure 7.21 using both existing Siemens rule language with CQL queries (as the baseline) and *SAL*; 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 with an Intel Core i5-4300U CPU at 2.60 GHz and 16 GB ofRAM running Windows 7 Enterprise (64 bits).

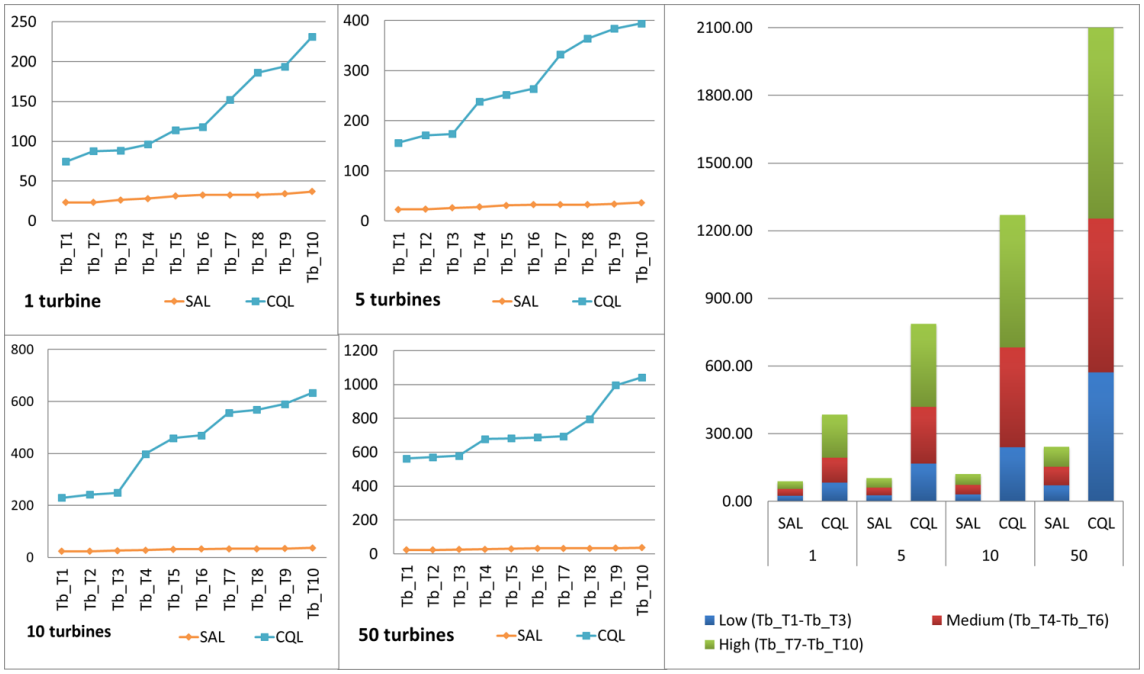

Figure 7.18: Results of the turbine user study. Left figures: the average time in seconds that the users took to express the tasks from Figure 7.21 for 1, 10, 50 turbine, respectively, using existing Siemens rule language (Baseline or B) and our semantic rule language *SAL* (Semantic or S). Right figures: the total time in seconds the user took to express these tasks grouped according to their complexity.

Figure 7.18 summarises the results of the user study. The four left figures present the average time that the five participants took to formulate the 5 tasks over respectively 1, 10, and 50 turbines, respectively. We now first discuss how the authoring time changes within each of the four figures, that is, when moving from simple to complex tasks

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 *SAL* is used. Moreover, in the semantic case the time only slightly increases when moving from simple (Tb_T1) to complex (Tb_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 these 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 train analytical tasks when moving from 1 to 50 trains. In the baseline case, moving to a higher number of trains requires to duplicate and modify the rules by first slightly modifying the

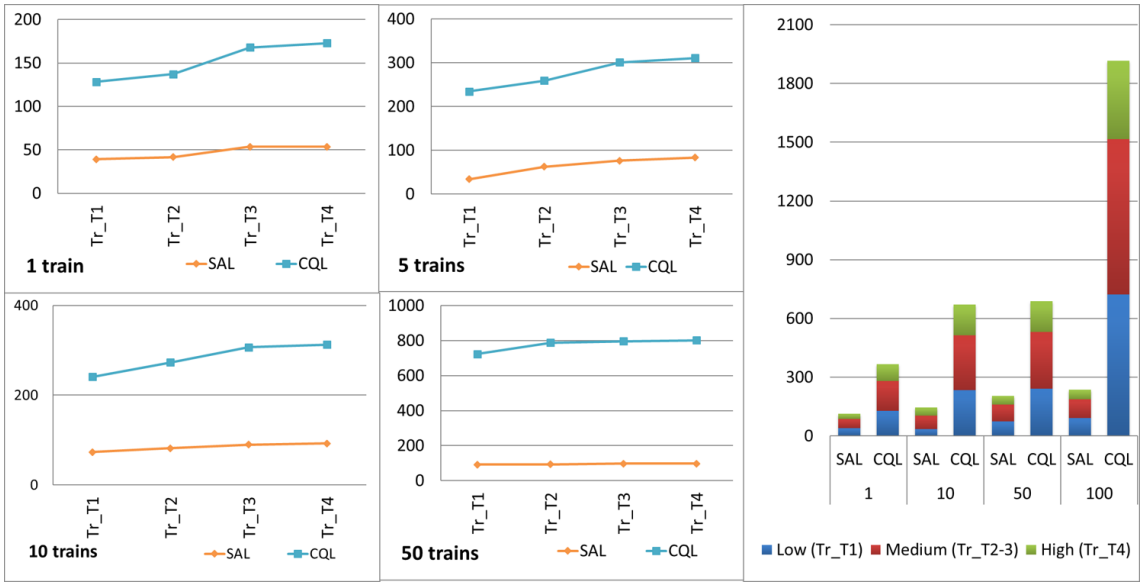

Figure 7.19: Results of the train user study. Left figures: the average time in seconds that the users took to express the tasks from Figure 7.21 for 1, 10, 50 train, respectively, using existing Siemens rule language (Baseline or B) and our semantic rule language *SAL* (Semantic or S). Right figures: the total time in seconds the user took to express these tasks grouped according to their complexity.

rule structure (to adapt the rules to train variations) and then replacing concrete sensors tags, threshold values, etc. In the semantic case, moving to a higher number of train 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 trains is only about twice higher than for 1 train. Indeed, to adapt the semantic diagnostic task Tr_T4 from 1 to 50 trains the participants in average spent 50 seconds, while formulating the original task for 1 train took them about 30 seconds.

Finally, let us consider how the total time for all 4 tasks changes when moving from 1 to 50 trains. This information is in Figure 7.19. One can see that in the baseline case the time goes from 500 to 2.100 seconds, while in the semantic case it goes from 90 to 290. Thus, for 4 tasks the semantic approach allows to save about 2.010 seconds and it is more than 4 times faster than the baseline approach.

Figure 7.20 summarises the results of the user study. The four left figures present the average time that the five participants took to formulate the 5 tasks over respectively 1, 10, and 50 grid components, respectively. We now first discuss how the authoring time changes within each of the four figures, that is, when moving from simple to complex tasks

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 *SAL* is used. Moreover, in the semantic case the time only slightly increases when moving from simple (Gr_T1) to complex (Gr_T5) tasks, while in the baseline case it increases significantly: from 2

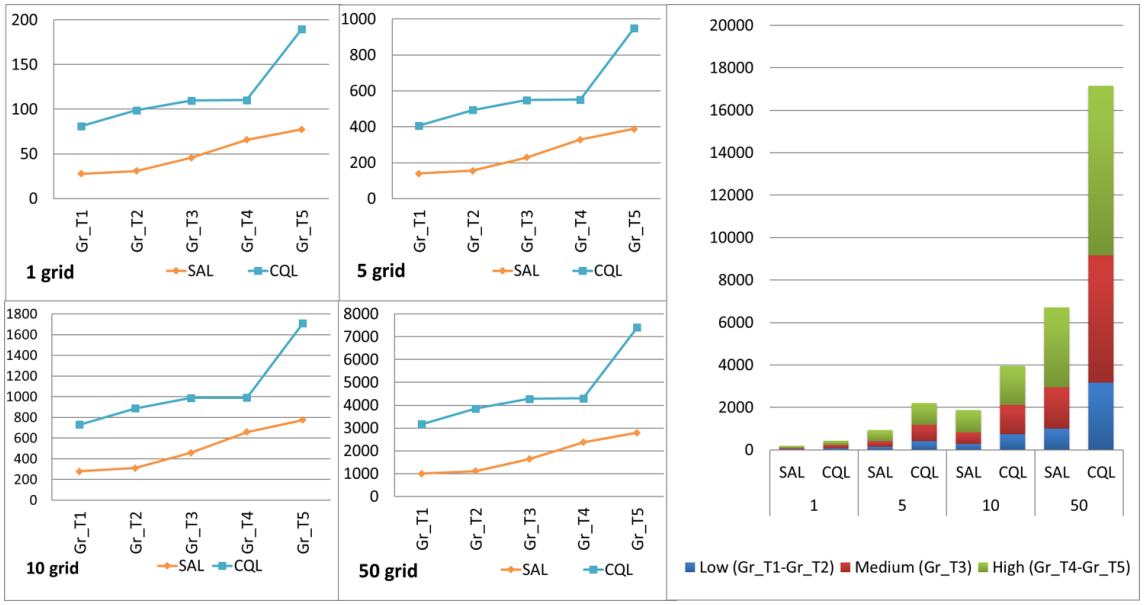

Figure 7.20: Results of the smart grid user study. Left figures: the average time in seconds that the users took to express the tasks from Figure 7.21 for 1, 10, 50 grid components, respectively, using existing Siemens rule language (Baseline or B) and our semantic rule language *SAL* (Semantic or S). Right figures: the total time in seconds the user took to express these tasks grouped according to their complexity.

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 these 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.

Task #	Complexity	sensors,event,code lines	Analytical task
Tr_T1	Low	(23,6,223)	Car doors ok
Tr_T2	Medium	(13,22,453)	Start-up normally
Tr_T3	Medium	(19,23,421)	Axle faults
Tr_T4	High	(21,64,631)	Brake release
Tb_T1	Low	(4,2,102)	Variable guided vanes analysis
Tb_T2	Low	(6,5,133)	Multiple start attempts
Tb_T3	Low	(6,3,149)	Lube oil system analysis
Tb_T4	Medium	(6,2,231)	Monitoring Trainstates
Tb_T5	Medium	(18,0,282)	Interduct thermocouple analysis
Tb_T6	Medium	(16,2,287)	Igniter failure detection
Tb_T7	High	(17,3,311)	Bearing carbonisation
Tb_T8	High	(19,2,335)	Combustion chamber dynamics
Tb_T9	High	(15,4,376)	Gearbox Unit Shutdown
Tb_T10	High	(12,8,401)	Surge detection
Gr_T1	Low	(14,1,400)	HVAC unit analyses
Gr_T2	Low	(25,2,500)	Power plant trip analysis
Gr_T3	Medium	(77,5,760)	Power station state monitoring
Gr_T4	High	(350,8,1387)	Economic dispatch curve
Gr_T5	High	(648,38,1654)	Synchronization reserve

Figure 7.21: Analytical tasks for Siemens trains, turbines and grid that were used in the case study, where complexity is defined using the number of sensor tags, event messages, and lines of code.

Runtime Analysis of Workflow Generation using SAL

This section present experiment set to evaluate the runtime performance of the semantic workflow. The goal is to analyze the efficiency of the CQL code generated by our OBDA component (see Chapter 6 for details on our OBDA component).

We consider 3 different scenarios which are related to train diagnostics, performance analytics of gas turbines, and smart grid component analysis, respectively. For each

scenario we prepared the diagnostic tasks, corresponding data, and execute the SQL queries translated from the rules 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) and 106 GB of RAM. The experiments of our system consisted of two steps: translation of semantic driven analytical workflows into SQL queries and then execution of generated queries.

We now describe the details of the three scenarios and their evaluations.

Diagnostic workflow of trains: In Figure 7.22 we present 4 diagnostic tasks of trains (T_1 is the running example). On the data side, we took measurements from 29 sensors as well as the relevant information about the trains where the sensors were installed. Then, we scaled the original data both in number of sensors and time dimensions. Our scaling respects the structure of the original data. The largest data for 232 sensors took 8GB on disk in a PostgreSQL database engine. For these diagnostic tasks the generated SQL codes are ranging from 113 to 839 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 most complex one contains 120 joins). The results of the query evaluation are presented in Figure 7.23.

Performance workflow of gas turbines: In Figure 7.24, we present 3 tasks for performance measurement of steam turbines. The data contains aggregated values from various sensors, which are deployed in many different components of steam turbines. The data scales from 1 to 10 GB in a PostgreSQL database. The largest table contains 1 month data with 5 minutes frequency for 2449 turbines. For these tasks, the corresponding SQL queries range from 116 to 407 lines of code and contain at least 20 joins. The query evaluation results for the performance measurement of steam turbines is presented in Figure 7.25.

Smart grid component analysis: In Figure 7.27 we present 3 component analysis tasks. Note that Gr_W1–Gr_W3 are independent from each other. This is a good example of modularity of *SAL*. 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 5.1GB on disk in a PostgreSQL database engine. The generated SQL code ranges from 109 to 568 lines depending on the analytical task and the code is of a relatively complex structure, e.g., for each analytical task the corresponding SQL contains at least 10 joins. The results of the query execution are presented in Figure 7.27.

The overall evaluation results are encouraging. We observe that query evaluation scales well in all three scenarios. Specifically, the running time grows almost linearly with respect to the data size. We also observe that for turbines the computation of performance measurements (Figure 7.25) took much longer time than the analytical tasks. This can be explained by the fact that turbine diagnostics rules involve more

Train Workflow Tr_W1**:** "Verify that all car doors ok in locomotive L1?": See Equation (5.1)

Train Workflow Tr_W2**:** "Does locomotive L1 start-up normally?":

$$\begin{aligned}
\mathsf{StartingTractionEffort} &= \mathsf{StatorVoltage} : \mathit{trend}(\text{'up'}) : \\
&\quad \mathit{duration}(>, 10s) : \mathit{after}[5s] \\
\mathsf{TractionRotorRPM} &: \mathit{value}(>, \mathit{RotorStartMinThreshold}) : \mathit{after}[20s] \\
\mathsf{TrainSpeed} &: \mathit{value}(>, \mathit{MinLineSpeed}). \\
\mathsf{TractionControlOK} &= \mathsf{MotorTemperature} : \\
&\quad \mathit{value}(<, \mathit{TempMaxThreshold})\ \mathit{and} \\
\mathsf{CoolingControlPressure} &: \mathit{value}(<, \mathit{PressureMaxThreshold})\ \mathit{and} \\
\mathsf{DifferentialCurrent} &: \mathit{value}(<, \mathit{CurrentMaxThreshold}). \\
\mathsf{message}(\textit{"Locomotive Normal Start-up"}) &= \mathsf{StartingTractionEffort}\ \mathit{and}\ \mathsf{TractionControlOK} \\
\mathit{and}\ \mathsf{NormalBrakeRelease}&.
\end{aligned}$$

Train Workflow Tr_W3**:** "Does locomotive L1 have critical axle faults.":

$$\begin{aligned}
\mathsf{HotBearings} &= \mathit{avg}\ \mathsf{AllBearingsTempSensor} : \\
&\quad \mathit{value}(>, \mathit{BearingTemperatureMaxThreshold}). \\
\mathsf{HotWheelRims} &= \mathsf{WheelRimsTemperature} : \mathit{trend}(\text{'up'}) : \mathit{duration}(>, 10s) \\
\mathsf{message}(\textit{"Critical Axle"}) &= (\mathsf{HotBearings}\ \mathit{or}\ \mathsf{HotWheelRims}).
\end{aligned}$$

Train Workflow Tr_W4**:** "Verify that the service braking is released normally in each car of locomotive L1?":

$$\begin{aligned}
\mathsf{CompressorRestart} &= \mathsf{CompressorRestartPressure} : \\
&\quad \mathit{value}(<, \mathit{BrakeSystemMaxPressure}). \\
\mathsf{BrakeReleaseOK} &= \mathsf{BrakeReleaseRate} : \\
&\quad \mathit{value}(<, \mathit{BrakeReleaseRateMaxThreshold})\ \mathit{and} \\
\mathsf{AllCarBrakePressureValve} &: \mathit{value}(=, \mathit{ClosedValue})\ \mathit{and} \\
\mathsf{AirBrakesMainResVolume} &: \mathit{value}(<, \mathit{AirBrakesMainResVolumeMinThreshold}). \\
\mathsf{NormalBrakeRelease} &= \mathsf{CarDoorsOK}\ \mathit{within}\ \mathsf{CompressorRestart} : \\
&\quad \mathit{after}[2s]\ \mathsf{BrakeReleaseOK}. \\
\mathsf{message}(\textit{"Normal Brake Release"}) &= \mathsf{NormalBrakeRelease}.
\end{aligned}$$

Figure 7.22: Analytical workflows for train diagnostics.

aggregation functions, and the generated SQL queries become more selective for the later steps like coalescing and temporal join appearing after than the aggregation and consequently run faster.

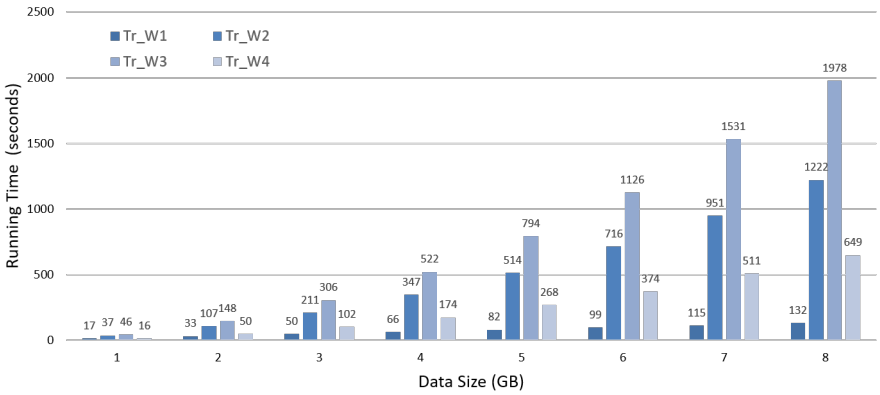

Figure 7.23: Workflow evaluation results for the train diagnostics.

Turbine Performance Analytical Workflow Tb_W1**:** "Steam Engine Failure Integrity":

$$\begin{aligned}\mathsf{FailureIntegrity} = \mathsf{TotalSteamFlow} : \mathit{trend}(\text{`}\mathit{down}\text{'}) :\\ \mathit{duration}(>, 10m) : \mathit{after}[5m]\\ \mathsf{TurbineSpeed} : \mathit{value}(>, 1000).\\ \mathsf{message}(\text{"}\mathit{Steam\ Engine\ Failure\ Integrity}\text{"}) = \mathsf{FailureIntegrity}.\end{aligned}$$

Turbine Performance Analytical Workflow Tb_W2**:** "Is turbine T1 in service?":

$$\begin{aligned}\mathsf{StartUp} = \mathsf{TurbineSpeed} : \mathit{value}(>, 2500) : \mathit{duration}(>, 10m) :\\ \mathit{after}[5m]\\ \mathsf{TurbineSpeed} : \mathit{value}(<, 1000) : \mathit{duration}(>, 10m).\\ \mathsf{OperatingMode} = \mathsf{ActivePowerGrossMW} : \mathit{value}(>, 90)\ \mathit{and}\\ \mathsf{TurbineSpeed} : \mathit{value}(>, 2400).\\ \mathsf{InService} = \mathsf{OperatingMode} : \mathit{duration}(>, 2h) : \mathit{after}[5m]\\ \mathsf{StartUp} : \mathit{duration}(>, 10m).\\ \mathsf{message}(\text{"}\mathit{Turbine\ In\ Service}\text{"}) = \mathsf{InService}.\end{aligned}$$

Turbine Performance Analytical Workflow Tb_W3**:** "Is turbine T1 in outage?":

$$\begin{aligned}\mathsf{NonOperatingMode} = \mathsf{ActivePowerGrossMW} : \mathit{value}(<, 10)\ \mathit{and}\\ \mathsf{TurbineSpeed} : \mathit{value}(<, 90).\\ \mathsf{InOutage1} = \mathsf{NonOperatingMode} : \mathit{duration}(>, 5m) : \mathit{after}[5m]\\ \mathsf{StartUp} : \mathit{duration}(>, 10m)\\ \mathsf{InOutage2} = \mathsf{NonOperatingMode} : \mathit{duration}(>, 5m) : \mathit{after}[5m]\\ \mathsf{OperatingMode} : \mathit{duration}(>, 2h)\\ \mathsf{message}(\text{"}\mathit{Turbine\ In\ Outage}\text{"}) = (\mathsf{InOutage1}\ \mathit{or}\ \mathsf{InOutage1}).\end{aligned}$$

Figure 7.24: Analytical workflow for performance analytics of turbines.

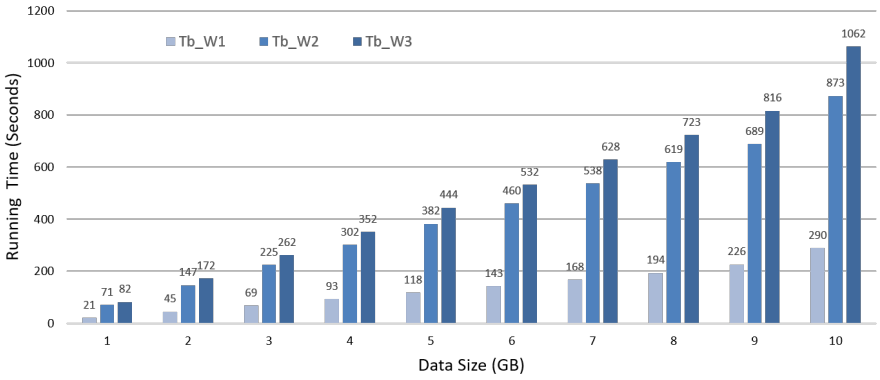

Figure 7.25: Workflow evaluation results for the performance analytics workflow of turbines.

Grid Analytical Workflow Gr_W1**:** "Is there a ramp change after 6 min in the grid Gr100?":

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

Grid Analytical Workflow Gr_W2**:** "Does the power station in grid Gr100 reach purging and ignition speed for 30 sec?":

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

Grid Analytical Workflow Gr_W3**:** "Does the turbine in grid Gr100 go from ignition to stand still within 1min and then stand still for 30 sec?":

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

Figure 7.26: Analytical workflows for smart grid component analysis.

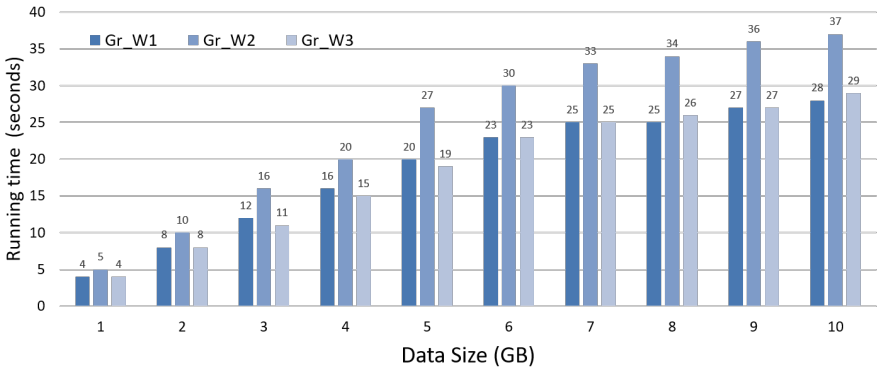

Figure 7.27: Workflow evaluation results for smart grid component analysis.

8 Conclusions and Future Work

In this chapter, we conclude our thesis research challenges and results. We specifically highlight our contributions in theory and in practice. Research limitations are presented to showcase the overall applicability and boundaries of our proposed solution. Lastly, we discuss few potential areas and extensions for future research.

Conclusion

The main focus of this dissertation was to improve data analysis performance for industrial analytical tasks and workflows. To achieve this goal we chose three data sets from different domains and investigated the role of domain-specific semantics on the performance of various state-of-the-art to data-access for executing and managing analytical task and workflows. To this end, we developed and worked on the following four research questions:

RQ1 Can domain-specific and analytical-aware ontology models for industrial equipment enhance data analysis performance?

RQ2 Can an analytical-aware ontology language of analytical tasks enhance data analysis performance?

RQ3 Can semantic-driven analytical workflows boost data analysis performance?

RQ4 Can a semantically defined analytical system boost data analysis performance?

Our main motivation behind all our research work is to prove that existing approaches to data analysis on industrial data are semantically weak. They lack representational semantics of interdisciplinary domain a when analyzing e.g. equipment related faults. This usually limits data analysis performance, because the results (e.g. fault identification) highly dependent on the contextual semantics and restricted by its nature of use, as described in Chapter 3. In Chapters 3, 4, 5 and 6 we present our proposed solution for extracting and incorporating conceptual semantics for data access and analytical tasks. We evaluated our solution for different diagnostic analysis tasks on three different industrial use-cases. To this end, in the second part (Chapter 7) we turned our attention towards studying the effectiveness

of our approach for multiple industrial datasets and initiated multiple analytical task and analytical workflows and compared the performance of our approaches against several state-of-the-art baselines.

Our main conclusion in this thesis is that the semantics of domain together with analytical functions and workflows should be considered when conducting any data analytical operation. Approaches that extract and use semantics for data analysis surpass those that merely rely on affect hard coded data models, or syntactic or rule structures that unambiguously reflect faults in a given equipment.

In the following sections, we summarise our main contributions to theory and practice, research limitations and future work.

Contributions

As mentioned in the introduction, the research presented in this dissertation is based on the scientific areas of technical systems engineering, ontology engineering, and semantic data integration and analytics as well as related areas. Consequently, it offers contributions to these different knowledge bases.

Research in the field of industrial technical systems is typically focused on the structuring, standardization and generalization of equipment and system data catalogs or on methods for their implementation within companies. Yet, despite the growing interest in this field, little work published in the ISO literature addresses the problem of data integration across different kinds of domain-specific systems. One of the difficulties in such integration arises from the lack of a uniform description method for any arbitrary measured physical or virtual component. Moreover, a description of the relations between such component also lacking. The main contribution of this dissertation to research on industrial technical systems therefore lies in providing an ontological formalization of all relevant elements, attributes, and properties. Our proposed TechOnto ontology provides a common language to enable data analysis across different distributed data sets and different domains and to foster interoperability among analytical tools. A more specific contribution related to the TechOnto ontology lies in the description of a technical architecture in which an ontology-based approach for data integration can be applied to achieve interoperability and reuse and to structure an inherently unstructured field. Because this dissertation is also influenced by the research areas of OE and semantic data integration, it also contributes to manage different reference ontology models and align and reuse their constructs to the TechOnto constructs. Our work also supports tooling to build and manage such ontology models and provide user-friendly interfaces to define domain-specific constraints. However, we only support data validation and query reasoning services in our implemented tool called SOMM model manager.

An important contribution to theory is the development of our semantic language

SAL for equipment diagnostics that is specifically tailored towards ontology mediated data integration scenarios such as industrial diagnostics. Our language has favourable computational properties and we have formally proved them by resorting to our extension of DatalogMTL. We provide preliminary grammar of semantic diagnostic tasks and analytical workflows, that includes analytical processing expressions, message rules with conjunction and negation. Moreover, we give proofs of our complexity results and conducted formal study of our *SAL* language. We also focused on theoretical aspects of redundancy, conflicts, and provenance for sets of semantic diagnostic workflows. In addition, we also present theoretical results on first order rewritability in presence of ontologies and present its comprehensive study via reduction of *SAL* to our non-trivial extension of DatalognrMTL.

The system architecture and the prototype application developed in this dissertation can guide the future development of tools to support industrial diagnostic applications, particularly web-based systems for the semantic modelling and data analytics across different domains. Through the theory-driven approach adopted in this research, it contributes to improving the already existing analytical tools within Siemens, which lack data integration and interoperability for analytical tasks and workflows. The developed TechOnto ontology can be used and is already deployed at Siemens Power generation business as an independent data format to achieve interoperability between different data sources and analytical tools. Moreover, it could function as a starting point for companies to develop interoperable analytics that would enable them to more easily perform performance comparisons within their own organizations and across organizational boundaries. With the linking of data sources to a diagnostic system that provides a standardized interface in the form of the TechOnto ontology, analytics-as- a-service could be offered in the future. For data scientist, the analytical-aware semantic system reduces the time end effort required to integrate different data sources into single workflow and to manage different analytical workflows using same or distributed data sets.

We showed practical benefits of our language and ontology-based rather than data-driven solutions on three industrial use-cases: trains, turbines and smart-grid, integrated in industrial diagnostic application. These benefits are the ease of formulation and favourable execution time of diagnostic workflows on industrial IoT with hundreds of pieces of complex industrial equipment and components in trains, turbines and smart-grids. We believe that our work opens new avenues for research in the areas of semantic access, semantic (industrial) IoT, and smart diagnostics, since it shows how such diagnostics can be abstracted from the data it should operate on and since it shows practical benefits of such approach. The main lesson we learned is the performance effectiveness of our semantic-driven analytical language in dealing with the complexity of the analytical tasks and workflows and the number of trains and sensors for analytics deployment. The evaluation shows that diagnostic engineers can save up to 66% of time by employing semantic 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 workflows. Another important lesson we learned

is that execution of semantic workflows is efficient and scales well to thousands of data points and tools which corresponds to real-world complex diagnostic tasks.

Research Limitations

This research has some limitations. This work assumes that the trend toward semantic-driven analytics will continues and that the formal description of analytical services will become increasingly important for the automation of performance analyses that are based upon it. The current increase in formalization activities is consistent with this assumption thus supports the relevance of this research.

The TechOnto domain ontology was developed based on various technical system data-catalog and on strategic and service-oriented industrial data collected over the last five years. Although these data provide a broad basis for the development of a domain-specific ontology, they cannot be considered to cover all aspects of every system in the market. Thus, the ontology must be made publicly available to provide a greater opportunity to identify current shortcomings for consideration in the next version of the ontology.

The developed semantic language *SAL* provides a limited number of analytical functions and operators that must be extended to cater more complex analytical tasks. Many machine learning tasks such as clustering, classification are strong candidates for language extension.

Additional limitations arise from the data formats used for integration. At present, the implementation presupposes connections to relational databases. This is because most of the underlying data used for integration within this dissertation are already stored in relational databases. This limitation will likely lead to higher effort in attaching non-relational databases to the system by third parties. Likewise, streaming data is also not supported in the current version of our proposed system, which will require additional resources to implement query optimization and translation mechanisms.

Moreover, the system is designed for the internal purposes of an industrial partner Siemens with special security clearances. As a result, no component of the semantic system, TechOnto ontology or language is publicly available. However, we are making efforts to publicize the semantic language as it bears no links to any domain in particular.

Future Work

Based on the results of the included publications and in conjunction with the conducted research and evaluations, the following section presents ideas for future research in the major areas addressed by this work.

First and the foremost, we would like to strengthen the evaluation by considering additional domain specifications and their related diagnostic tasks. One of the strong candidate is in additive manufacturing business where in a typical manufacturing plant, data is generated and stored whenever a piece of equipment consumes material or completes a task. This data is then accessed by plant operators using manufacturing execution systems (MES) software programs that monitor the operations in the plant and report anomalies. MESs are responsible for keeping track of the material inventory in different locations and tracing their consumption, thus ensuring that equipment and materials needed for each process are available at the relevant time. Most common conceptual models used in the manufacturing business is ISA-88/95 standard, where product, process and execution are main components used for analytical task. We think that comparing our domain models with the ISA standard and further authoring and managing diagnostic task for such use-case would generate more results and could lead to improvement in our proposals.

From the semantic language point of view, we can conduct further analysis by evaluating the recent trending semantic constraint language called SHACL. It would be interesting to see how both language would execute diagnostic tasks and how well they perform with respect to computational complexity and execution time.

Another important future research could also aim to exploit our proposed semantic analytical language to address data quality issues. In the world of semantics, approaches to data quality assessment is gaining much popularity and there is no sufficient work being done in the field. We think that by exploiting our language constructs and devising specific semantic workflows, we can accommodate various quality metrics such as identification of missing data, erroneous data, inconsistent data, outliers and other domain specific violations. This should ultimately lead to research in the areas of data quality assurance, data verification, enforcement and persistency techniques where analytical-aware semantics can be a potential solution.

Bibliography

[1] E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov, D. Bilidas, M. Giese, P. Haase, I. Horrocks, H. Kllapi, M. Koubarakis, Ö. Özçep, et al., Optique: Towards obda systems for industry, in: Extended Semantic Web Conference, Springer, 2013, pp. 125–140.

[2] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, Ontologies and databases: The dl-lite approach, in: Reasoning Web International Summer School, Springer, 2009, pp. 255–356.

[3] M. Pfaff, H. Krcmar, Natural language processing techniques for document classification in it benchmarking, in: Proceedings of the 17th International Conference on Enterprise Information Systems-Volume 1, SCITEPRESS-Science and Technology Publications, Lda, 2015, pp. 360–366.

[4] T. A. Runkler, Data Analytics - Models and Algorithms for Intelligent Data Analysis, Second Edition, Springer, 2016.

[5] A. Arasu, S. Babu, J. Widom, The cql continuous query language: semantic foundations and query execution, The VLDB Journal 15 (2) (2006) 121–142.

[6] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: Advances in neural information processing systems, 2013, pp. 2787–2795.

[7] D. Luckham, The power of events: An introduction to complex event processing in distributed enterprise systems, in: Rule Representation, Interchange and Reasoning on the Web: International Symposium, RuleML 2008, Orlando, FL, USA, October 30-31, 2008. Proceedings, Vol. 5321, Springer, 2008, p. 3.

[8] M. Nickel, K. Murphy, V. Tresp, E. Gabrilovich, A review of relational machine learning for knowledge graphs, Proceedings of the IEEE 104 (1) (2015) 11–33.

[9] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic web 8 (3) (2017) 489–508.

[10] P. Ristoski, H. Paulheim, Semantic web in data mining and knowledge discovery: A comprehensive survey, Web semantics: science, services and agents on the World Wide Web 36 (2016) 1–22.

[11] R. Socher, D. Chen, C. D. Manning, A. Ng, Reasoning with neural tensor networks for knowledge base completion, in: Advances in neural information processing systems, 2013, pp. 926–934.

[12] E. Kharlamov, N. Solomakhina, Ö. L. Özçep, D. Zheleznyakov, T. Hubauer, S. Lamparter, M. Roshchin, A. Soylu, S. Watson, How semantic technologies can enhance data access at siemens energy, in: ISWC, 2014, pp. 601–619.

[13] G. Vachtsevanos, F. L. Lewis, M. Roemer, A. Hess, B. Wu, Intelligent Fault Diagnosis and Prognosis for Engineering Systems, Wiley, 2006.

[14] W. Bolton, Instrumentation and Control Systems, Elsevier, 2004.

[15] E. S. Notes, Enter the data economy: Eu policies for a thriving data ecosystem, Issue 21 (2017) 1.

[16] I. Horrocks, M. Giese, E. Kharlamov, A. Waaler, Using semantic technology to tame the data variety challenge, IEEE Internet Computing 20 (6) (2016) 62–66.

[17] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, New York, NY, USA, 2003.

[18] C. Bizer, A. Seaborne, D2rq-treating non-rdf databases as virtual rdf graphs, in: Proceedings of the 3rd international semantic web conference (ISWC2004), Vol. 2004, Proceedings of ISWC2004, 2004.

[19] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, D. F. Savo, The mastro system for ontology-based data access, Semantic Web 2 (1) (2011) 43–53.

[20] F. Priyatna, R. Alonso-Calvo, S. Paraiso-Medina, G. Padron-Sanchez, Ó. Corcho, R2rml-based access and querying to relational clinical data with morph-rdb., in: SWAT4LS, 2015, pp. 142–151.

[21] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering sparql queries over relational databases, Semantic Web 8 (3) (2017) 471–487.

[22] K. Munir, M. S. Anjum, The use of ontologies for effective knowledge modelling

and information retrieval, Applied Computing and Informatics 14 (2) (2018) 116–126.

[23] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The dl-lite family, Journal of Automated reasoning 39 (3) (2007) 385–429.

[24] D. Calvanese, A. Mosca, J. Remesal, M. Rezk, G. Rull, A historical case of ontology-based data access, in: 2015 Digital Heritage, Vol. 2, IEEE, 2015, pp. 291–298.

[25] N. C. Cysneiros, A. C. Salgado, Including hierarchical navigation in a graph database query language with an obda approach, in: 2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW), IEEE, 2016, pp. 109–114.

[26] M. Pfaff, H. Krcmar, Semantic integration of semi-structured distributed data in the domain of it benchmarking, in: 16th International conference on enterprise information systems (ICEIS), 2014, pp. 320–324.

[27] E. Kharlamov, D. Hovland, E. Jiménez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao, D. Zheleznyakov, I. Horrocks, Ontology based access to exploration data at statoil, in: ISWC, 2015, pp. 93–112.

[28] D. Calvanese, P. Liuzzo, A. Mosca, J. Remesal, M. Rezk, G. Rull, Ontology-based data integration in epnet: Production and distribution of food during the roman empire, Eng. Appl. of AI 51 (2016) 212–229.

[29] B. Charron, Y. Hirate, D. Purcell, M. Rezk, Extracting semantic information for e-commerce, in: ISWC, 2016, pp. 273–290.

[30] R. B. Randall, Vibration-based condition monitoring: industrial, aerospace and automotive applications, Wiley, 2011.

[31] B. K. N. Rao, Handbook of condition monitoring, Elsevier, 1996.

[32] J. S. Mitchell, An introduction to machinery analysis and monitoring, Pennwell Books, 1993.

[33] European political strategy centre (epsc strategic notes. enter the data economy: Eu policies for a thriving data ecosystem. technical report 21, the european commission, 2017, `https://ec.europa.eu/epsc/sites/epsc/files/strategicnoteissue21.pdf`.

[34] B. Catania, T. Cerquitelli, S. Chiusano, G. Guerrini, M. Kämpf, A. Kemper,

B. Novikov, T. Palpanas, J. Pokorný, A. Vakali, New trends in databases and information systems.

[35] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, M. Grossniklaus, C-SPARQL: a continuous query language for RDF data streams, Int. J. Semantic Computing 4 (1) (2010) 3–25.

[36] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al., Swrl: A semantic web rule language combining owl and ruleml, W3C Member submission 21 (79) (2004) 1–31.

[37] S. Brandt, E. G. Kalaycı, R. Kontchakov, V. Ryzhikov, G. Xiao, M. Zakharyaschev, Ontology-based data access with a horn fragment of metric temporal logic, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[38] B. Raphael, Sir: A computer program for semantic information retrieval.

[39] M. Minsky, A framework for representing knowledge.

[40] R. J. Brachman, J. G. Schmolze, An overview of the kl-one knowledge representation system, in: Readings in artificial intelligence and databases, Elsevier, 1989, pp. 207–230.

[41] N. Guarino, Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy, Vol. 46, IOS press, 1998.

[42] T. R. Gruber, A translation approach to portable ontology specifications, Knowledge acquisition 5 (2) (1993) 199–220.

[43] R. Studer, V. R. Benjamins, D. Fensel, Knowledge engineering: principles and methods, Data & knowledge engineering 25 (1-2) (1998) 161–197.

[44] F. Baader, I. Horrocks, U. Sattler, Description logics, in: Handbook on ontologies, Springer, 2004, pp. 3–28.

[45] I. Horrocks, P. F. Patel-Schneider, Reducing owl entailment to description logic satisfiability, in: International semantic web conference, Springer, 2003, pp. 17–29.

[46] E. V. Kostylev, J. L. Reutter, Answering counting aggregate queries over ontologies of the dl-lite family, in: Twenty-Seventh AAAI Conference on Artificial Intelligence, 2013.

[47] J. Tappolet, A. Bernstein, Applied temporal rdf: Efficient temporal querying

of rdf data with sparql, in: European Semantic Web Conference, Springer, 2009, pp. 308–322.

[48] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, in: Journal on data semantics X, Springer, 2008, pp. 133–173.

[49] M. Rodriguez-Muro, D. Calvanese, Quest, an owl 2 ql reasoner for ontology-based data access, Citeseer, 2012.

[50] M. Rodriguez-Muro, D. Calvanese, High performance query answering over dl-lite ontologies, in: Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning, 2012.

[51] M. Rodrıguez-Muro, R. Kontchakov, M. Zakharyaschev, Query rewriting and optimisation with database dependencies in ontop, Proc. of DL 2013.

[52] A. Artale, R. Kontchakov, F. Wolter, M. Zakharyaschev, Temporal description logic for ontology-based data access, in: Twenty-Third International Joint Conference on Artificial Intelligence, 2013.

[53] A. Artale, R. Kontchakov, V. Ryzhikov, M. Zakharyaschev, The complexity of clausal fragments of ltl, in: International Conference on Logic for Programming Artificial Intelligence and Reasoning, Springer, 2013, pp. 35–52.

[54] R. Koymans, Specifying real-time properties with metric temporal logic, Real-time systems 2 (4) (1990) 255–299.

[55] S. Brandt, E. G. Kalaycı, V. Ryzhikov, G. Xiao, M. Zakharyaschev, A framework for temporal ontology-based data access: A proposal, in: M. Kirikova, K. Nørvåg, G. A. Papadopoulos, J. Gamper, R. Wrembel, J. Darmont, S. Rizzi (Eds.), New Trends in Databases and Information Systems - ADBIS 2017 Short Papers and Workshops, AMSD, BigNovelTI, DAS, SW4CH, DC, Nicosia, Cyprus, September 24-27, 2017, Proceedings, 2017, pp. 161–173.

[56] J.-P. Calbimonte, H. Y. Jeung, O. Corcho, K. Aberer, Enabling query technologies for the semantic sensor web, International Journal on Semantic Web and Information Systems 8 (EPFL-ARTICLE-183971) (2012) 43–63.

[57] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: International Semantic Web Conference, Springer, 2011, pp. 370–388.

[58] D. Anicic, P. Fodor, S. Rudolph, N. Stojanovic, EP-SPARQL: a unified language for event processing and stream reasoning, in: WWW, 2011, pp. 635–644.

[59] H. Beck, M. Dao-Tran, T. Eiter, M. Fink, LARS: A logic-based framework for analyzing reasoning over streams, in: AAAI, 2015, pp. 1431–1438.

[60] I. Grangel-González, L. Halilaj, G. Coskun, S. Auer, D. Collarana, M. Hoffmeister, Towards a semantic administrative shell for industry 4.0 components, in: ICSC, 2016, pp. 230–237.

[61] A. Stolz, B. Rodriguez-Castro, A. Radinger, M. Hepp, PCS2OWL: A generic approach for deriving web ontologies from product classification systems, in: ESWC, 2014, pp. 644–658.

[62] M. Hepp, J. de Bruijn, Gentax: A generic methodology for deriving OWL and RDF-S ontologies from hierarchical classifications, thesauri, and inconsistent taxonomies, in: ESWC, 2007, pp. 129–144.

[63] Classification and product description, `http://www.eclass.eu/`.

[64] C. Riedl, N. May, J. Finzen, S. Stathel, V. Kaufman, H. Krcmar, An idea ontology for innovation management, in: Semantic Services, Interoperability and Web Applications: Emerging Concepts, IGI Global, 2011, pp. 303–321.

[65] G. F. Ceschini, G. Mehdi, D. Naderi, M. Roshchin, A method and apparatus for performing a model-based failure analysis of a complex industrial system, uS Patent App. 15/579,972 (Jun. 21 2018).

[66] E. Kharlamov, T. Mailis, G. Mehdi, C. Neuenstadt, Ö. Özçep, M. Roshchin, N. Solomakhina, A. Soylu, C. Svingos, S. Brandt, et al., Semantic access to streaming and static data at siemens, Web Semantics: Science, Services and Agents on the World Wide Web 44 (2017) 54–74.

[67] E. Kharlamov, B. C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin, I. Horrocks, Capturing industrial information models with ontologies and constraints, in: ISWC, 2016, pp. 325–343.

[68] B. Cuenca Grau, E. Jimenez-Ruiz, E. Kharlamov, Y. Nenov, G. Mehdi, Somm: Industry oriented ontology management tool.

[69] G. Mehdi, S. Brandt, M. Roshchin, T. A. Runkler, Towards semantic reasoning in knowledge management systems, in: AI for Knowledge Management workshop at IJCAI, 2016.

[70] G. Mehdi, S. Brandt, M. Roshchin, T. A. Runkler, Semantic framework for industrial analytics and diagnostics, in: IJCAI, 2016, pp. 4016–4017.

[71] R. Stetter, Software Im Maschinenbau-Laestiges Anhangsel Oder Chance

Zur Marktfuehrerschaft?, VDMA, ITQ(http://www.software-kompetenz.de/en/).

[72] Siemens, Modeling New Perspectives: Digitalization - The Key to Increased Productivity, Efficiency and Flexibility (White Paper), DER SPIEGEL.

[73] H. Kagermann, W.-D. Lukas, Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution, VDI Nachrichten.

[74] Forschungsunion, Fokus: Das zukunftsprojekt industrie 4.0, handlungsempfehlungen zur umsetzung, Bericht der Promotorengruppe KOMMUNIKATION.

[75] S. Feldmann, S. J. Herzig, K. Kernschmidt, T. Wolfenstetter, D. Kammerl, A. Qamar, U. Lindemann, H. Krcmar, C. J. Paredis, B. Vogel-Heuser, Towards effective management of inconsistencies in model-based engineering of automated production systems, IFAC-PapersOnLine 48 (3) (2015) 916–923.

[76] V. Vyatkin, Software Engineering in Industrial Automation: State-of-the-Art Review, IEEE Transactions on Industrial Informatics 9 (3) (2013) 1234–1249.

[77] R. G. Qiu, M. Zhou, Mighty MESs; state-of-the-art and future manufacturing execution systems., IEEE Robot. Automat. Magazine 11 (1) (2004) 19–25.

[78] A. Gliozzo, O. Biran, S. Patwardhan, K. McKeown, Semantic technologies in ibm watson, in: Proceedings of the fourth workshop on teaching NLP and CL, 2013, pp. 85–92.

[79] G. Mehdi, E. Kharlamov, O. Savković, G. Xiao, E. G. Kalaycı, S. Brandt, I. Horrocks, M. Roshchin, T. Runkler, Semantic rule-based equipment diagnostics, in: International Semantic Web Conference, Springer, 2017, pp. 314–333.

[80] G. Mehdi, S. Brandt, M. Roshchin, T. Runkler, Towards semantic reasoning in knowledge management systems, in: IFIP International Workshop on Artificial Intelligence for Knowledge Management, Springer, 2016, pp. 132–146.

[81] E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov, D. Bilidas, M. Giese, P. Haase, I. Horrocks, H. Kllapi, M. Koubarakis, Ö. L. Özçep, M. Rodriguez-Muro, R. Rosati, M. Schmidt, R. Schlatte, A. Soylu, A. Waaler, Optique: Towards OBDA systems for industry, in: ESWC Satellite Events, 2013, pp. 125–140.

[82] R. Batres, M. West, D. Leal, D. Price, K. Masaki, Y. Shimada, T. Fuchino, Y. Naka, An upper ontology based on iso 15926, Computers & Chemical Engineering 31 (5-6) (2007) 519–534.

[83] B. Motik, I. Horrocks, U. Sattler, Bridging the gap between OWL and relational databases, J. Web Sem. 7 (2) (2009) 74–89.

[84] J. Tao, E. Sirin, J. Bao, D. L. McGuinness, Integrity constraints in OWL, in: AAAI, 2010.

[85] R. Mayntz, T. P. Hughes, The development of large technical systems, Campus Verlag, 1988.

[86] M. C. Suárez-Figueroa, A. Gómez-Pérez, M. Fernández-López, The neon methodology for ontology engineering, in: Ontology engineering in a networked world, Springer, 2012, pp. 9–34.

[87] T. Tudorache, N. F. Noy, S. W. Tu, M. A. Musen, Supporting Collaborative Ontology Development in Protégé, in: ISWC, 2008, pp. 17–32. doi:10.1007/978-3-540-88564-1_2.
URL http://dx.doi.org/10.1007/978-3-540-88564-1_2

[88] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Optimising ontology classification, in: ISWC, 2010, pp. 225–240.

[89] E. Kharlamov, B. C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin, I. Horrocks, Capturing industrial information models with ontologies and constraints, in: International Semantic Web Conference, Springer, 2016, pp. 325–343.

[90] B. Bishop, F. Ficsher, Iris integrated rule inference system, in: Workshop on Advancing Reasoning on the Web, 2008.

[91] T. Tudorache, C. Nyulas, N. F. Noy, M. A. Musen, WebProtégé: a Collaborative Ontology Editor and Knowledge Acquisition Tool for the Web, Semantic Web 4 (1) (2013) 89–99.

[92] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, J. Banerjee, RDFox: A highly-scalable RDF store, in: ISWC, 2015, pp. 3–20. doi:10.1007/978-3-319-25010-6_1.

[93] E. Jiménez-Ruiz, B. Cuenca Grau, LogMap: Logic-based and scalable ontology matching, in: ISWC, 2011, pp. 273–288.

[94] M. Erdmann, W. Waterfeld, Overview of the NeOn Toolkit, in: Ontology Engineering in a Networked World., 2012, pp. 281–301.

[95] J. Day-Richter, M. A. Harris, M. Haendel, S. Lewis, OBO-Edit - an ontology editor for biologists, Bioinformatics 23 (16) (2007) 2198–2200. doi:10.1093/

bioinformatics/btm112.
URL http://dx.doi.org/10.1093/bioinformatics/btm112

[96] Top Quadrant, Topbraid composer, http://www.topquadrant.com/.

[97] E. Jiménez-Ruiz, B. Cuenca Grau, Y. Zhou, I. Horrocks, Large-scale interactive ontology matching: Algorithms and implementation, in: ECAI, 2012, pp. 444–449.

[98] G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. G. Kalaycı, S. Brandt, I. Horrocks, M. Roshchin, T. A. Runkler, Semantic rule-based equipment diagnostics, in: ISWC, 2017, pp. 314–333.

[99] O. Savkovic, E. Kharlamov, G. Xiao, G. Mehdi, E. G. Kalayci, W. Nutt, M. Roshchin, I. Horrocks, Theoretical characterization of signal diagnostic processing language., in: Description Logics, 2018.

[100] E. Kharlamov, G. Mehdi, O. Savković, G. Xiao, E. G. Kalaycı, M. Roshchin, Semantically-enhanced rule-based diagnostics for industrial internet of things: The sdrl language and case study for siemens trains and turbines, Journal of Web Semantics 56 (2019) 11–29.

[101] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semantics 10 (2008) 133–173.

[102] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3).

[103] S. Brandt, E. G. Kalaycı, V. Ryzhikov, G. Xiao, M. Zakharyaschev, Querying log data with metric temporal logic, CoRR abs/1703.08982.

[104] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Syst. 2 (4).

[105] G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. G. Kalaycı, S. Brandt, I. Horrocks, M. Roshchin, T. Runkler, Semdia: Semantic rule-based equipment diagnostics tool, in: CIKM, 2017.

[106] E. Kharlamov, G. Mehdi, O. Savkovic, G. Xiao, S. Lamparter, I. Horrocks, A. Waaler, Towards simplification of analytical workflows with semantics at siemens, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 1951–1954.

[107] E. Kharlamov, O. Savkovic, G. Xiao, R. Penaloza, G. Mehdi, I. Horrocks, M. Roshchin, Semantic rules for machine diagnostics: Execution and management, in: CIKM, 2017.

[108] S. Cohen, W. Nutt, Y. Sagiv, Containment of aggregate queries, in: International Conference on Database Theory, Springer, 2003, pp. 111–125.

[109] A. Kalyanpur, B. Parsia, M. Horridge, E. Sirin, Finding all justifications of owl dl entailments, in: The Semantic Web, Springer, 2007, pp. 267–280.

[110] R. Penaloza, B. Sertkaya, Complexity of axiom pinpointing in the dl-lite family of description logics., in: ECAI, Vol. 215, 2010, pp. 29–34.

[111] G. Mehdi, T. Runkler, M. Roshchin, S. Suresh, N. Quang, Ontology-based integration of performance related data and models: An application to industrial turbine analytics, in: Industrial Informatics (INDIN), 2017 IEEE 15th International Conference on, IEEE, 2017, pp. 251–256.

[112] G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. G. Kalaycı, S. Brandt, I. Horrocks, M. Roshchin, T. Runkler, Semantic rules for siemens turbines, in: ISWC (Posters and Demos), 2017.

[113] G. Mehdi, D. Naderi, G. Ceschini, M. Roshchin, Model-based reasoning approach for automated failure analysis: An industrial gas turbine application, in: PHM, 2015.

[114] G. Mehdi, D. Naderi, G. Ceschini, M. Roshchin, Model-based approach to automated calculation of key performance indicators for industrial turbines, in: PHM, 2015.

[115] G. Mehdi, T. A. Runkler, S. Suresh, M. Roshchin, Q. Nguyen, Semantic-aware analytics for smart grids.

[116] O. Savković, E. Kharlamov, M. Ringsquandl, G. Xiao, G. Mehdi, E. G. Kalayc, W. Nutt, I. Horrocks, Semantic diagnostics of smart factories, in: Joint International Semantic Technology Conference, Springer, 2018, pp. 277–294.

[117] E. Kharlamov, O. Savković, M. Ringsquandl, G. Xiao, G. Mehdi, E. G. Kalayc, W. Nutt, M. Roshchin, I. Horrocks, T. Runkler, Diagnostics of trains with semantic diagnostics rules, in: International Conference on Inductive Logic Programming, Springer, 2018, pp. 54–71.

[118] Siemens power generation, `https://new.siemens.com/global/en/products/energy/power-generation/gas-turbines/sgt-800.html`.

[119] Siemens mobility, `https://new.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/energy-is-going-digital.html`.

[120] Siemens smart grid, `https://new.siemens.com/global/en/products/`

energy/energy-automation-and-smart-grid/smart-communications.html.

[121] A. Crapo, X. Wang, J. Lizzi, R. Larson, The semantically enabled smart grid, The Road to an Interoperable Grid (Grid-Interop).