SOMM: Industry Oriented Ontology Management Tool

Evgeny Kharlamov[1] Bernardo Cuenca Grau[1] Ernesto Jiménez-Ruiz[1]
Steffen Lamparter[2] Gulnar Mehdi[2] Martin Ringsquandl[2]
Yavor Nenov[1] Sebastian Brandt[2] Ian Horrocks[1]

[1] University of Oxford, UK [2] Siemens AG, Corporate Technology, Germany

Abstract. This paper describes the outcomes of an ongoing collaboration between Siemens and the University of Oxford, with the goal of facilitating the design of ontologies and their deployment in applications. Ontologies are mainly used in Siemens to capture the conceptual *information models* underpinning a wide range of applications. We start by describing the key role that such models play in two use cases in the manufacturing and energy production sectors. Then, we discuss the formalisation of information models using ontologies, and the relevant reasoning services. Finally, we present SOMM—a tool that supports engineers with little background on semantic technologies in the creation of ontology-based models and in populating them with data. SOMM implements a fragment of OWL 2 RL extended with a form of integrity constraints for data validation, and it comes with support for schema and data reasoning, as well as for model integration. Our evaluation demonstrates the adequacy of SOMM's functionality and performance for Siemens applications.

1 Introduction

Software systems in the domain of industrial manufacturing 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 [21]. Additionally, a wide range of 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 [4, 10, 20] 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 [26] for a survey). This research has highlighted the need for enterprise-wide *information models*—machine-readable conceptualisations describing the functionality of and information flow between the different assets in a plant, such as equipment and production processes. The development information models based on ISA and IEC standards[1] has now become a common practice in modern companies [16].

Siemens was amongst the first major companies in the manufacturing industry to exploit information models in deployed applications [20]. In practice, however, many different 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.

A key recent development in Siemens has been the adoption of semantic technologies [6, 7, 11, 12, 18], where an important application has been the formalisation of the information models within the company using ontologies. 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. Furthermore, RDF provides a unified data format: 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.

In this paper, we describe the outcomes of an ongoing collaboration between Siemens Corporate Technology in Munich and the University of Oxford, with the goal of facilitating deployment of ontology-based industrial information models. We start by describing the key role that information models play in two use cases in the manufacturing and energy production sectors. Then, we illustrate the information models used in Siemens for describing manufacturing and energy plants, and discuss how they can be captured using ontologies. 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 [13, 22], which are not available in OWL 2. Hence, we discuss in detail what kinds of constraints are needed in Siemens' use cases 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 Siemens' application requirements.

Ontologies are currently being created and maintained by qualified R&D personnel with expertise in ontology languages and ontology engineering. Siemens is, however, interested in widening the scope of application of semantic technologies

[1] International Society of Automation and International Electrotechnical Commission.

within the company, and for this it is crucial to make ontology development accessible to other teams of engineers. To this end, we have developed the Siemens-Oxford Model Manager (SOMM)—a tool that has been designed to fulfil Siemens' needs and which supports engineers with little background on semantic technologies in the creation and use ontologies. SOMM provides a simple interface for ontology development and enables the introduction of instance data via automatically generated forms that are driven by the ontology and which help minimising errors in data entry. SOMM implements a fragment of the OWL 2 RL profile [14] extended with database integrity constraints for data validation; the supported language is sufficient to capture the information models used by Siemens and their use in applications. SOMM is built on top of Web-Protégé [25], which provides built-in functionality for ontology versioning and collaborative development. It relies on the triple store RDFox [15] for query answering and data validation, the reasoner HermiT [5] for ontology classification, and LogMap [8] to support model alignment and merging.

We showcase the practical benefits of our tool using two ontologies in the manufacturing and power generation domains. Both ontologies have been developed using SOMM by Siemens engineers to capture information models currently in use. Based on these ontologies, we conducted an empirical evaluation of SOMM's performance in supporting constraint validation and query answering over realistic manufacturing and gas turbine data. Our evaluation demonstrates the adequacy of SOMM's functionality and performance for Siemens applications.

2 Information Models Used in Siemens

Siemens exploits conceptual information models in a wide range of manufacturing and energy production applications. In this Section, we discuss two concrete use cases and describe the underpinning models and their limitations.

2.1 Applications in Manufacturing and Energy Production

In manufacturing and energy production plants it is essential that all processes and equipment run smoothly and without interruptions.

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 [16].

Similarly, in energy production plants turbines contain sensors and equipment that are continuously generating data. This data is consumed by *remote monitoring systems (RMS)*, which analyse turbine data to prevent faults, report anomalies and ensure that the turbines operate without interruption. In both application scenarios, the use of information models is twofold.

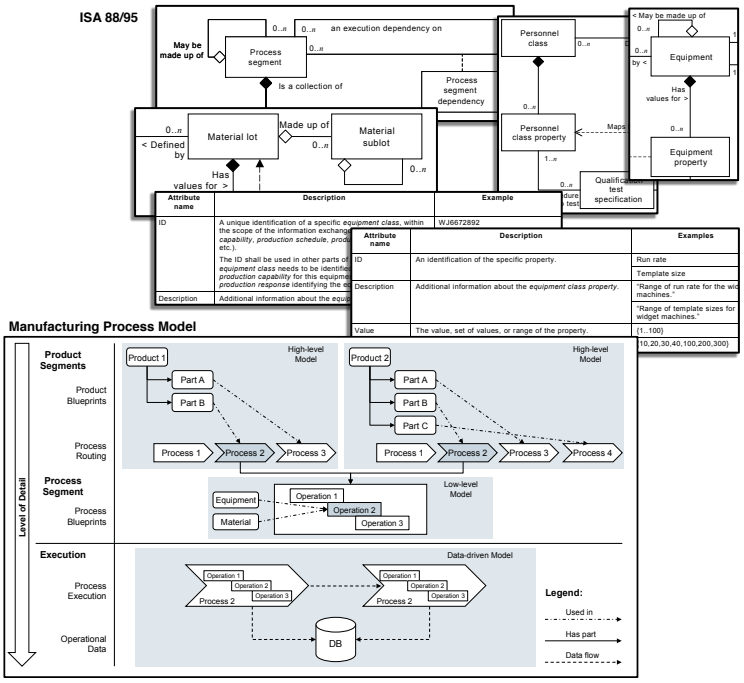


Fig. 1: Fragment of ISA 88/95 and an example model based on it.

1. Models are used to provide machine-readable specifications for the data generated by equipment and processes, and for the data flow across assets and processes in a plant.
2. Models provide a schema for constructing and executing complex queries. In particular, monitoring tasks in MESs are realised by means of queries issued to production machines and data hubs; similarly, anomaly detection in an RMS relies on queries spanning the structure of the turbines, the readings of their sensors, and the configuration of turbines within a plant.

2.2 The Siemens Models

We next describe the information models in Siemens relevant to the aforementioned applications. These models have been developed in compliance with ISA, IEC, and ISO/TS international standards.

Manufacturing Models Siemens has been developing conceptual models for manufacturing applications based on the international standard ISA-88/95.

The ISA-88/95 standard provides general guidelines for specifying the functionality of and interface between manufacturing software systems. The standard consists of UML-like diagrammatic descriptions accompanied with tables and unstructured text, which are used to extend the diagrams with additional information and examples. Figure 1 presents an excerpt of the ISA-88/95 standard modelling materials, equipment, personnel, and processes in a plant. For instance, one of these diagrams establishes that pieces of equipment can be composed by other pieces of equipment and are described by a number of specified 'equipment

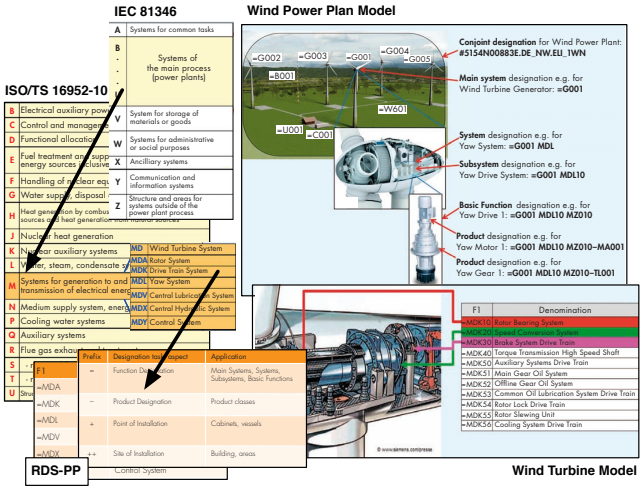


Fig. 2: Designation models IEC 81346, ISO/TS 16952-10, and RDS-PP and example energy information model for an energy plant [17].

properties'. The table complementing this diagram indicates that each piece of equipment must have a numeric ID and may have a textual description; additional properties of equipment can be introduced by providing an ID, a textual description of the property, and a value range.

Figure 1 provides a simplified version of a Siemens information model based on the ISA-88/95 standard. The model is organised in three layers: *product*, *process*, and *execution*. On the product level, we can see the specification of two products and their relationship to production processes; for instance, *Product1* consists of *PartA* and *PartB*, which are manufactured by two consecutive processes. The process segment level provides more fine-grained specifications of the structure of each process; for instance, *Process2* consists of three operations, where the second one relies on specific kinds of materials and equipment. Finally, at the execution level, we can see how data is stored and accessed by individual processes.

Energy Plant Models The Siemens models for energy plants are based on the *Reference Designation System for Power Plants (RDS-PP)* and *Kraftwerk-Kennzeichensysten (KKS)* standards, which are in turn extensions for the energy sector of the IEC 81346 and ISO/TS 16952-10 international standards.

IEC 81346 and ISO/TS 16952-10 provide a generic dictionary of codes for designating and classifying industrial equipment. Figure 2 provides an except of these standards together with their dependencies. For instance, in IEC-81346 letters 'B' to 'U' are used for generically designating systems in power plants. ISO/TS 16952-10 makes this specification more precise by indicating, for example, that letter 'M' refers to systems for generating and transmitting electrical energy, and that we can append 'D' to 'M' to refer to a wind turbine system. RDS PP and KKS provide a much more extensive vocabulary of codes for energy equipment, their functionality and locations, as well a system for combining such codes.

A Siemens energy plant model describes the structure of a plant by providing the functionality and location of each equipment component using RDS PP and

KKS codes. Having this information in a machine-readable format is important for planning and construction, as well as for the software-driven operation and maintenance of the plant. Figure 2 shows how a specific plant is represented in a model; for instance, code *=G001 MDL10* denotes that the yaw drive system number 10 of type *MDL* is located in the wind turbine generator number 001.

2.3 Technical Challenges

The development and use of models in practice poses major challenges.

1. Model development is costly, as it requires specialised training and proprietary tools; as a result, model development often cannot keep up with the arrival of new equipment and introduction of new processes.
2. Models are difficult to integrate and share since they are often independently developed using different types of proprietary software and they are based on incompatible data formats.
3. Monitoring queries are difficult to compose and execute on top of information models: they must comply with the requirements of the models (e.g., refer to specific codes in the energy use case), and their execution requires access to heterogeneous data from different machines and processes.

Semantic technologies have been recently adopted by Siemens in a number of applications [6, 7, 12, 18]. In this paper, our focus in on the use of OWL 2 for describing information models. OWL 2 provides a rich and flexible modelling language that is ideally suited for addressing the aforementioned challenges: it not only comes with an unambiguous, standardised, semantics, but also with a wide range of tools and infrastructure. Furthermore, RDF provides a unified data exchange format, which can be used to seamlessly access and exchange data, and hence facilitate monitoring tasks based on complex queries.

3 From Siemens Models to Ontologies

In this section we describe the ontologies that we have developed to capture the Siemens manufacturing and energy production models. The goal of our ontologies is to eventually replace their underpinning models in applications. Thus, 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 monitoring queries.

In Section 3.1 we discuss the modelling choices underpinning the design of our ontologies and identify a fragment of OWL 2 RL that is sufficient to capture the basic aspects of the Siemens' information models. Our analysis of the models, however, also revealed the need to incorporate database integrity constraints for data validation, which are not supported in OWL 2 [13, 22]. Therefore, we also discuss the kinds of constraints that are relevant to our applications.

Finally, in Section 3.2 we discuss how the OWL 2 RL axioms and integrity constraints can be captured by means of rules with stratified negation for the purpose of data validation and query answering. We assume basic familiarity with Datalog—the rule language underpinning OWL 2 RL and SWRL—as well as with stratified negation-as-failure (see [1] for a survey on Logic Programming).

3.1 Ontology Modelling

From an ontological point of view, the building blocks of the Siemens 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 main challenge that we encountered was to capture the constraints of the Siemens models using ontological axioms. We next describe how this was accomplished using a combination of OWL 2 RL axioms and integrity constraints.

Standard OWL 2 RL Axioms 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* 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:

$$\text{SubClassOf}(\textit{Turbine}\ \textit{Equipment}) \tag{1}$$

$$\text{SubDataPropertyOf}(\textit{hasRotorSpeed}\ \textit{hasSpeed}) \tag{2}$$

$$\text{TransitiveObjectProperty}(\textit{hasPart}) \tag{3}$$

$$\text{InverseObjectProperties}(\textit{hasPart}\ \textit{partOf}) \tag{4}$$

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

$$\text{SubClassOf}(\textit{Conveying}\ \text{ObjectAllValuesFrom}(\textit{followedBy}\ \textit{Packaging})) \tag{5}$$

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 RL profile. This has many practical advantages for reasoning since OWL 2 RL is amenable to efficient implementation using rule-based technologies.

Constraint Axioms In addition to optional relationships, the Siemens 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

$$\text{SubClassOf}(\mathit{Turbine}\ \text{ObjectSomeValuesFrom}(\mathit{hasPart}\ \mathit{Rotor})) \tag{6}$$

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

The Siemens 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

$$\text{SubClassOf}(\mathit{TwoRotorTurbine}\ \text{ObjectMinCardinality}(2\ \mathit{hasPart}\ \mathit{Rotor})) \tag{7}$$

$$\text{SubClassOf}(\mathit{TwoRotorTurbine}\ \text{ObjectMaxCardinality}(2\ \mathit{hasPart}\ \mathit{Rotor})) \tag{8}$$

Such cardinality restrictions are interpreted as integrity constraints in applications: when introducing a specific double rotor turbine, the model requires that we also provide its two rotors. The semantics of axioms (7) and (8) is not well-suited for this purpose: on the one hand, (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 (8) non-deterministically enforces at least two of them to be equal.

There have been several proposals to extend OWL 2 with integrity constraints [13, 22]. 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 RL.

3.2 Data Validation and Query Answering

Our approach to data validation and query answering in the Siemens use cases follows the standard approaches in the literature [13, 22].

Given a user query Q in SPARQL and an OWL 2 ontology $\mathcal{O}$ consisting of a set $\mathcal{S}$ of standard OWL 2 RL axioms, a set $\mathcal{C}$ of axioms marked as constraints and a dataset $\mathcal{D}$, we proceed according to the Steps 1–6 given next.

1. Translate the standard axioms $\mathcal{S}$ into a Datalog program Π_S using the well-known correspondence between OWL 2 RL and Datalog.
2. Translate the integrity constraints $\mathcal{C}$ into a Datalog program Π_C with stratified negation-as-failure containing a distinguished binary predicate *Violation* for recording the individuals and axioms involved in a constraint violation.
3. Compute the materialisation M_I of program Π_I w.r.t. the dataset $\mathcal{D}$.
4. Compute the materialisation M_C of program Π_C w.r.t. the previously computed materialisation M_I.

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 1: OWL 2 RL 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.

5. Retrieve and flag all integrity constraint violations by querying M_C using the SPARQL query '`SELECT` $?X\ ?Y$ `WHERE` $\{?X\ Violation\ ?Y\}$'. The data satisfies the constraints iff this query returns an empty answer.
6. If no constraints are violated, answer the user's SPARQL query Q directly against the dataset M_I, with no further reasoning required.

Steps 3–5 can be implemented on top of RDF triple stores with support for OWL 2 RL and stratified negation. In the remainder of this Section we illustrate Steps 1 and 2, where standard axioms and constraints are translated into rules.

Standard Axioms Table 1 provides the standard OWL 2 RL axioms needed to capture the Siemens models and their translation into negation-free rules. In particular, our example axioms (1)–(5) are equivalent to the following rules:

$$Equipment(?x) \leftarrow Turbine(?x) \tag{9}$$
$$hasSpeed(?x, ?y) \leftarrow hasRotorSpeed(?x, ?y) \tag{10}$$
$$hasPart(?x, ?z) \leftarrow hasPart(?x, ?y) \wedge hasPart(?y, ?z) \tag{11}$$
$$Packaging(?y) \leftarrow Conveying(?x) \wedge followedBy(?x, ?y) \tag{12}$$

Constraint Axioms Table 2 provides the constraint axioms required to capture the Siemens models 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 (6) from Section 3.1 is captured by the following rules:

$$hasPart\_Rotor(?x) \leftarrow hasPart(?x, ?y) \ \wedge \ Rotor(?y) \tag{13}$$
$$Violation(?x, \alpha) \leftarrow Turbine(?x) \wedge \textbf{not}\ hasPart\_Rotor(?x) \tag{14}$$

Rule (13) identifies all individuals with a rotor as a part, and stores them as instances of the auxiliary predicate *hasPart_Rotor*. In turn, Rule (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.

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)	$R\_2(?x) \leftarrow R(?x, ?y_1) \ \wedge \ R(?x, ?y_2) \wedge$ $\textbf{not} \ owl{:}sameAs(?y_1, ?y_2)$ and $Violation(?x, \alpha) \leftarrow R\_2(?x)$
SubClassOf(A MaxCardinality(n R B))	$R\_(n{+}1)\_B(?x) \leftarrow \bigwedge_{1 \leq i \leq n+1} (R(?x, ?y_i) \wedge B(?y_i))$ $\bigwedge_{1 \leq i < j \leq n+1} (\textbf{not} \ owl{:}sameAs(?y_i, ?y_j))$ and $Violation(?x, \alpha) \leftarrow A(?x) \ \wedge \ R\_(n{+}1)\_B(?x)$
SubClassOf(A MinCardinality(n R B))	$R\_n\_B(?x) \leftarrow \bigwedge_{1 \leq i \leq n} (R(?x, ?y_i) \wedge B(?y_i))$ $\bigwedge_{1 \leq i < j \leq n} (\textbf{not} \ owl{:}sameAs(?y_i, ?y_j))$ and $Violation(?x, \alpha) \leftarrow A(?x) \ \wedge \ \textbf{not} \ R\_n\_B(?x)$

Table 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.

Integrity constraints based on cardinalities require the use of the OWL 2 equality predicate *owl:sameAs*. For instance, the constraint axiom (7) from Section 3.1, to which we assign the id β_1, is translated into the following rules:

$$hasPart\_2\_Rotor(?x) \leftarrow \bigwedge_{1 \leq i \leq 2} (hasPart(?x, ?y_i) \wedge Rotor(?y_i)) \wedge$$
$$\wedge (\textbf{not} \ owl{:}sameAs(?y_1, ?y_2))$$
$$Violation(?x, \beta_1) \leftarrow TwoRotorTurbine(?x) \wedge \textbf{not} \ hasPart\_2\_Rotor(?x)$$

The first rule infers that an individual is 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 (7). Similarly, axiom (8), to which we assign the id β_2, is translated as follows:

$$hasPart\_3\_Rotor(?x) \leftarrow \bigwedge_{1 \leq i \leq 3} (hasPart(?x, ?y_i) \wedge Rotor(?y_i)) \wedge$$
$$\wedge \bigwedge_{1 \leq i < j \leq 3} (\textbf{not} \ owl{:}sameAs(?y_i, ?y_j))$$
$$Violation(?x, \beta_2) \leftarrow TwoRotorTurbine(?x) \wedge \ hasPart\_3\_Rotor(?x)$$

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 (8).

To conclude this section, we note that our translation in Table 2 yields a stratified program for any set $\mathcal{C}$ of constraints. We can always define a stratification

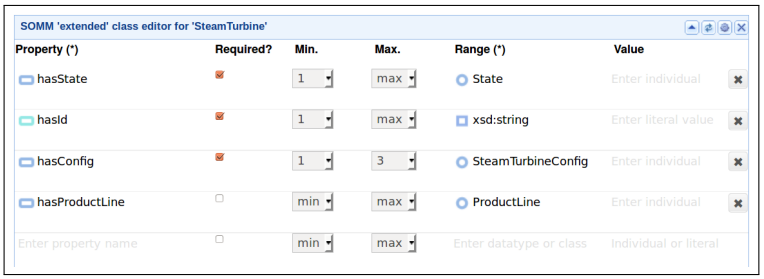


Fig. 3: SOMM editor to attach properties to classes.

where the lowest stratum consists of the predicates in $\mathcal{C}$ and *owl*:*sameAs*, 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.

4 SOMM: an Ontology Management Tool for Siemens

We have developed the *Siemens-Oxford Ontology Managemen (SOMM)* tool to support Siemens engineers 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 RL axioms and constraints discussed in Section 3.

SOMM is built on top of the Web-Protégé platform [25] by extending its front-end with new visual components and its back-end to access RDFox [15] for query answering and constraint validation, HermiT [19] for ontology classification, and LogMap [8] to support ontology alignment and merging. Our choice of WebProtégé was based on Siemens' requirements 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 altrenatives such as Protégé-desktop [24], NeON toolkit [3], OBO-Edit [2], and TopBraid Composer [23], we found that only WebProtégé satisfied all the aforementioned requirements.

In the remainder of this section, we describe the main features of SOMM.

Insertion of axioms and constraints. We have implemented a form-based interface for editing standard axioms and constraints. Figure 3 shows a screenshot of the SOMM class editor representing the following axioms about *SteamTurbine* (abbreviated below as ST), where all but the last axiom represent constraints.

$$\text{SubClassOf}(ST\ \text{ObjectSomeValuesFrom}(hasState\ State))$$
$$\text{SubClassOf}(ST\ \text{DataSomeValuesFrom}(hasId\ xsd{:}string))$$
$$\text{SubClassOf}(ST\ \text{ObjectMinCardinality}(1\ hasConfig\ STConfig))$$
$$\text{SubClassOf}(ST\ \text{ObjectMaxCardinality}(3\ hasConfig\ STConfig))$$
$$\text{SubClassOf}(ST\ \text{ObjectAllValuesFrom}(hasProductLine\ ProductLine))$$

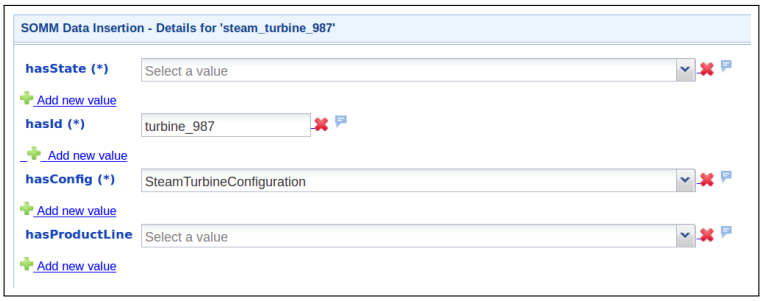


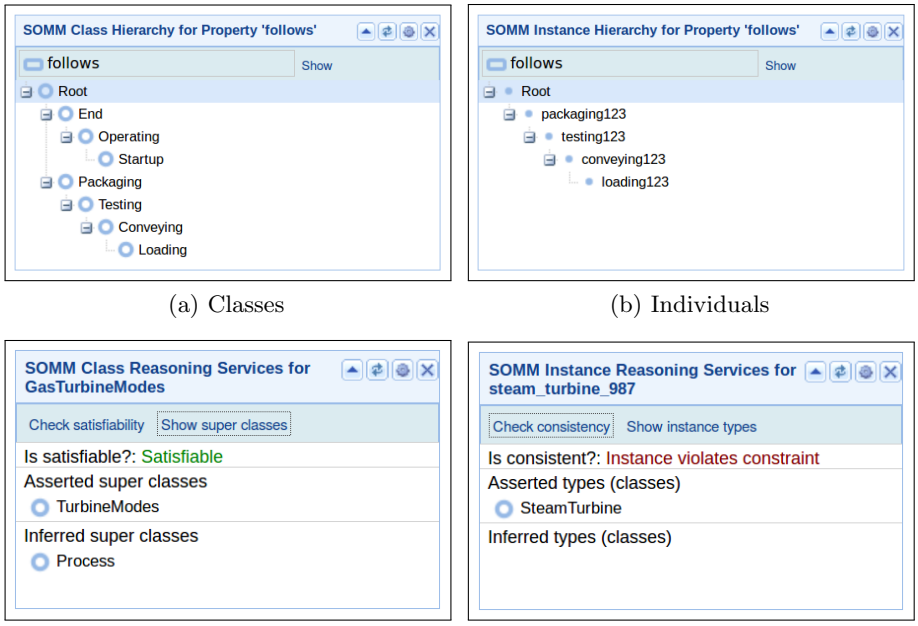
Fig. 4: Data insertion in SOMM.

The interface shows that the class *SteamTurbine* has three mandatory properties (*hasState*, *hasID* and *hasConfig*) marked as 'Required' and interpreted as constraints, and an optional property (*hasProductLine*) which is interpreted as a standard axiom. Object and data properties are indicated by blue and green rectangles, respectively. For each property we can specify their filler using a WebProtégé autocompletion field. Finally, the fields 'Min' and 'Max' are used to represent cardinality constraints on mandatory properties.

Automatically generated data forms. SOMM exploits the capabilities of the 'knowledge acquisition forms' in Web-Protégé to guide engineers during data entry. The forms are automatically generated for each class based on its relevant mandatory and optional properties. For this, SOMM considers (i) the explicitly provided properties; (ii) the inherited properties; and (iii) the properties explicitly attached to its descendant classes. The latter were deemed useful by Siemens engineers, e.g., although *Turbine* does not have directly attached properties, the SOMM interface would suggests adding data for the properties attached to its subclass *SteamTurbine*. Figure 4 shows an example of the property fields for an instance of the class *SteamTurbine*, where required fields (i.e., those for which a value must be provided) are marked with **(*)**.

Extended hierarchies. In addition to classical subsumption hierarchies, SOMM allows also for hierarchies based on arbitrary properties. These can be seen as a generalisation of partonomy hierarchies, and assume that the dependencies between classes or individuals based on the relevant property are 'tree-shaped'. Figures 5a and 5b show the hierarchy corresponding to the *follows* property, which determines which kinds of processes can follow other processes; for instance, *Conveying* follows *Loading* and is followed by *Testing*.

Alignment. SOMM integrates the ontology alignment system LogMap [8] to support model alignment and merging. Users can select and merge two available Web-Protégé projects, or import and merge an ontology into the active Web-Protégé project. Although LogMap supports interactive alignment [9], it is currently used in SOMM in an automatic mode; we are planning to extend SOMM's interface to support user interaction in the alignment process.



(a) Classes (b) Individuals

(c) Classes (d) Individuals

Fig. 5: Above: tree-like navigation of the ontology classes and individuals in SOMM. Below: reasoning services for ontology classes and individuals in SOMM.

Reasoning. SOMM relies on the OWL 2 reasoner HermiT [5] to support standard reasoning services such as class satisfiability and ontology classification. Data validation and query answering support is provided on top of the RDFox reasoner [15] as described in Section 3.2. Figures 5c and 5d illustrates the supported reasoning services. The left-hand-side of the figure shows that the class *GasTurbineModes* is satisfiable and *Process* is an inferred superclass. On the right-hand-side we can see that *steam_turbine_987* violates one of the integrity constraints; indeed, as shown in Figure 4, *steam_turbine_987* is missing data for the property *hasState*, which is mandatory for all steam turbines (see Figure 3).

5 Evaluation

We have evaluated the practical feasibility of the data validation and query answering services provided by SOMM. For this, we have conducted two sets of experiments for the manufacturing and energy turbine scenarios, respectively. In the first experiment, we simulated the operation of a manufacturing plant using a synthetic generator that produces realistic product manufacturing data of varying size; in the second experiment, we used real anonymised turbine data.[2] All our experiments were conducted on a laptop equipped with an Intel Core

[2] We are in the process of sorting out the licenses for the ontologies and data used in our experiments; they cannot be made publicly available at this point.

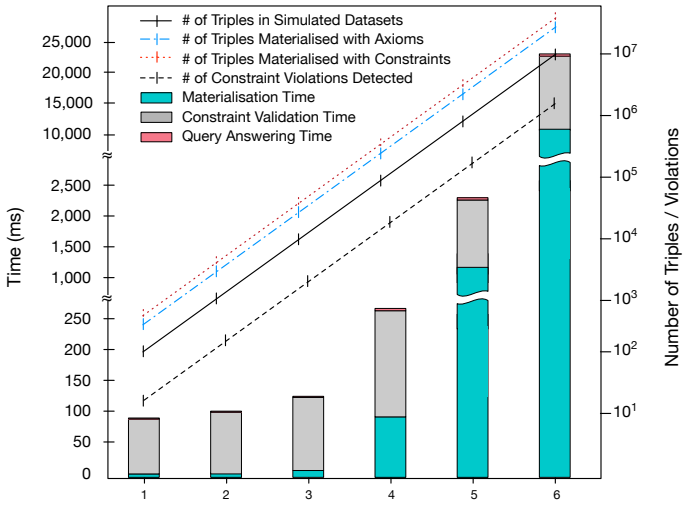


Fig. 6: Experimental results.

i7-4600U CPU at 2.10 GHz and 16 GB of RAM running Ubuntu 14.04 (64 bits). We allocated 6 GB to Java 8 and set up RDFox to use 4 parallel threads.

Manufacturing Experiments. In our experiments for the manufacturing use case we used the ontology, data and queries given next.

- The ontology capturing the manufacturing model illustrated in Figure 1 from Section 2.1. The ontology contains 79 standard axioms and 20 constraints.
- A data generator used by Siemens engineers to simulate manufacturing of products of two types based on the aforementioned model. We used two configurations of the generator: the first one ($C1$) simulates a situation where some products were manufactured in violation of the model specifications (e.g., they used too much material of some kind); in the second one ($C2$), each product is manufactured according to specifications.
- A sample of three monitoring queries commonly used in practice. The first query asks for all products that use material from a given lot; the second asks for all material lots used in a given product; finally, the third one asks for the total quantity of material in lots of a specific kind.

We generated data for 6 different sizes, ranging from 100 triples to 10 million triples. For each size, we generated one dataset for each configuration of the generator. We set up configuration $C1$ so that 35% of the manufactured products violate specification. Our experiments follow Steps 1–6 in Section 3.2. We checked validity of each dataset against the ontology using Steps 1–5; then, for each dataset created using $C2$ we also answered all test queries (Step 6). We repeated the experiment 5 times for each dataset (i.e., 10 times for each data size).

Our results are summarised in Figure 6. Times for each data size are *wall clock* time averages (in milliseconds). Materialisation times (blue bar) correspond to Step 3 in Section 3.2, where only standard axioms are considered. Constraint validation time (grey bar) represents the additional time required for Steps 4 and 5. Query answering times (red bar) measure the additional time for answering all

queries over the relevant materialisation (Step 6); here, only datasets satisfying the constraints (i.e., generated using $C2$) are considered. Finally, the figure also provides the average number of constraint violations in data generated according to $C1$, the number of triples inferred when materialising standard axioms, and the number of additional triples materialised during constraint validation.

Our results demonstrate the feasibility of our ontology-based approach to model validation and query answering in realistic manufacturing scenarios. In particular, constraint validation and query answering were feasible within $25s$ on stock hardware over datasets containing over 10 million triples.

Gas Turbine Experiment. In this experiment we used the following data:

- The ontology capturing the energy plant model illustrated in Figure 2 from Section 2. The ontology contains 121 standard axioms and 25 constraints.
- An anonymised dataset describing the structure of 800 real gas turbines of different types, their sensor readings (temperature, pressure, rotor speed and position), and associated processes (e.g., expansion, compression, start up, shut down). The dataset was converted from a relational DB into RDF, and contains 25090 triples involving 4076 individuals.
- Three commonly used test queries. The first query asks for the core parts, equipment and current state of all turbines of a given type; the second asks for all components involved in a compression process; the last query asks for the temperature readings of turbines of a given type.

We followed the same steps as in the previous experiments, with very positive results. Materialisation (Steps 2-3) took $20ms$ and generated $23,051$ new triples. Constraint checking was completed in $109ms$ and generated $3,790$ triples; we found 1582 constraint violations, which is especially interesting given that the data is real. Query answering over the valid subset took only $2ms$.

6 Conclusion

We have studied the use of ontologies to capture Siemens' information models in manufacturing and energy production applications. Our study of the requirements of information models allowed us to identify an ontology language that extends a fragment of OWL 2 RL with integrity constraints. We have implemented this language in the SOMM tool, which has been used by Siemens engineers to develop ontologies based on information models. The usability feedback we have obtained so far has been very positive, and we are planning to conduct formal user studies in the future. Finally, the key applications of information models can be formalised as data validation and query answering reasoning services; the results of our with realistic manufacturing and turbine data have been very encouraging.

7 References

[1] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In: *ACM Computing Surveys* 33.3 (2001).

[2] J. Day-Richter, M. A. Harris, M. Haendel, and S. Lewis. OBO-Edit - an ontology editor for biologists. In: *Bioinformatics* 23.16 (2007).

[3] M. Erdmann and W. Waterfeld. Overview of the NeOn Toolkit. In: *Ontology Engineering in a Networked World.* 2012.

[4] Forschungsunion. Fokus: Das Zukunftsprojekt Industrie 4.0, Handlungsempfehlungen zur Umsetzung. In: *Bericht der Promotorengruppe KOMMUNIKATION* (Mar. 2012).

[5] B. Glimm et al. HermiT: An OWL 2 Reasoner. In: *J. Autom. Reasoning* 53.3 (2014).

[6] S. Grimm, M. Watzke, T. Hubauer, and F. Cescolini. Embedded $\mathcal{EL}$ + Reasoning on Programmable Logic Controllers. In: *ISWC* (2012).

[7] T. Hubauer, S. Lamparter, and M. Pirker. Automata-Based Abduction for Tractable Diagnosis. In: *DL* (2010).

[8] E. Jiménez-Ruiz and B. Cuenca Grau. LogMap: Logic-Based and Scalable Ontology Matching. In: *ISWC*. 2011.

[9] E. Jiménez-Ruiz, B. Cuenca Grau, Y. Zhou, and I. Horrocks. Large-scale Interactive Ontology Matching: Algorithms and Implementation. In: *ECAI*. 2012.

[10] H. Kagermann and W.-D. Lukas. Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. In: *VDI Nachrichten* (Apr. 2011).

[11] E. Kharlamov et al. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In: *ISWC* (2014).

[12] E. Kharlamov et al. Ontology-Based Integration of Streaming and Static Relational Data with Optique. In: *SIGMOD demo* (2016).

[13] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between OWL and relational databases. In: *J. Web Sem.* 7.2 (2009).

[14] B. Motik et al. *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. Nov. 2012.

[15] Y. Nenov et al. RDFox: A Highly-Scalable RDF Store. In: *ISWC*. 2015.

[16] R. G. Qiu and M. Zhou. Mighty MESs; state-of-the-art and future manufacturing execution systems. In: *IEEE Robot. Automat. Magazine* 11.1 (2004).

[17] J. Richnow, C. Rossi, and H. Wank. Designation of wind power plants with the Reference Designation System for Power Plants - RDS-PP. In: *VGB PowerTech* 94 (7 2014).

[18] M. Ringsquandl et al. Semantic-Guided Feature Selection for Industrial Automation Systems. In: *ISWC* (2015).

[19] R. Shearer, B. Motik, and I. Horrocks. HermiT: a Highly-Efficient OWL Reasoner. In: *OWLED* (2008).

[20] Siemens. Modeling New Perspectives: Digitalization - The Key to Increased Productivity, Efficiency and Flexibility (White Paper). In: *DER SPIEGEL* (6 2015).

[21] R. Stetter. Software Im Maschinenbau-Laestiges Anhangsel Oder Chance Zur Marktfuehrerschaft? In: *VDMA, ITQ* (Mar. 2011). (`http://www.software-kompetenz.de/?21700`).

[22] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity Constraints in OWL. In: *AAAI*. 2010.

[23] Top Quadrant. *TopBraid Composer*. `http://www.topquadrant.com/`.

[24] T. Tudorache, N. F. Noy, S. W. Tu, and M. A. Musen. Supporting Collaborative Ontology Development in Protégé. In: *ISWC*. 2008.

[25] T. Tudorache, C. Nyulas, N. F. Noy, and M. A. Musen. WebProtégé: a Collaborative Ontology Editor and Knowledge Acquisition Tool for the Web. In: *Semantic Web* 4.1 (2013).

[26] V. Vyatkin. Software Engineering in Industrial Automation: State-of-the-Art Review. In: *IEEE Transactions on Industrial Informatics* 9.3 (2013).