Web Semantics: Science, Services and Agents on the World Wide Web 74 (2022) 100717

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

An MDE-based methodology for closed-world integrity constraint checking in the semantic web

Ambreen Hussain [a,*], Wenyan Wu [a], Zhaozhao Tang [b]

[a] Faculty of Computing, Engineering and Built Environment, Birmingham City University, Birmingham B4 7XG, United Kingdom
[b] Water Science and Environmental Engineering Research Centre, College of Chemistry and Environmental Engineering, Shenzhen University, Shenzhen 518060, China

ARTICLE INFO

Article history:
Received 6 April 2021
Received in revised form 6 March 2022
Accepted 11 May 2022
Available online 19 May 2022

Keywords:
Data validation
SWRL
SPARQL
Model-driven engineering
Water supply and distribution systems

ABSTRACT

Ontology-based data-centric systems support open-world reasoning. Therefore, for these systems, Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL) are not suitable for expressing integrity constraints based on the closed-world assumption. Thus, the requirement of integrating the open-world assumption of OWL/SWRL with closed-world integrity constraint checking is inevitable. SPARQL, recommended by World Wide Web (W3C), is a query language for RDF graphs, and many research studies have shown that it is a perfect candidate for closed-world constraint checking for ontology-based data-centric applications. In this regard, many research studies have been performed to transform integrity constraints into SPARQL queries where some studies have shown the limitations of partial expressivity of knowledge bases while performing the indirect transformations, whereas others are limited to a platform-specific implementation. To address these issues, this paper presents a flexible and formal methodology that employs Model-Driven Engineering (MDE) to model closed-world integrity constraints for open-world reasoning. The proposed approach offers semantic validation of data by expressing integrity constraints at both the model level and the code level. Moreover, straightforward transformations from OWL/SWRL to SPARQL can be performed. Finally, the methodology is demonstrated via a real-world case study of water observations data.

1. Introduction

Model-Driven Engineering (MDE) [1], also referred to as Model-Driven Software development (MDSD), is a paradigm shift from code-oriented to model-oriented, where modelling is the main activity in the software development life cycle. Utilising MDE enhances communication by raising the level of abstraction and domain specification to target different audiences. Through modelling, developers can reason at a higher level of abstraction, and model transformation can automate the code generation to relieve the developers from error-prone and repetitive tasks. Numerous artefacts could be generated for different platforms using a single model. The system is modelled first to consider the vulnerabilities from the early stages of the development to reveal potential risks before its implementation. In this regard, specification of the integrity constraints and business rules on the model [2] for the system and data validation can be performed using textual constraint languages.

The W3C recommends Web Ontology Language (OWL) [3] to express ontologies in the semantic web. However, OWL has expressive limitations, such as being undecidable for key inference problems. Another W3C standard, Semantic Web Rule Language (SWRL[1]) [3], is an unrestricted combination of OWL DL and Horn-style rules for modelling added domain knowledge [4]. Both OWL and SWRL inherit Open-World Assumption (OWA) and do not support UNA (Unique Name Assumption). Under OWA, a statement cannot be inferred to be false from the inability to prove that it is true. According to UNA, if we find two individuals with different names, we can assume they are two different individuals.

In contrast, Closed-World Assumption (CWA) allows deriving falsehood from the incapacity to derive truth [5] and offers a unique name for each individual. In other words, it assumes that knowledge of some or all parts of the domain exists. Thus, for the ontology-based data-centric applications, OWL/SWRL is not suitable to check closed-world integrity constraints because of OWA. The closed-world integrity constraints are the conditions to verify whether the knowledge explicitly present in the domain meets specific criteria [6]. The ontology-based systems

* Corresponding author.
E-mail addresses: ambreen.hussain@ieee.org (A. Hussain), wenyan.wu@bcu.ac.uk (W. Wu), zhaozhao.tang@foxmail.com (Z. Tang).

[1] https://www.w3.org/Submission/SWRL/.

https://doi.org/10.1016/j.websem.2022.100717

prefer OWL to check constraints because existing OWL reasoners are open source, practical, easily accessible, provide many standard and extended reasoning, and exhibit competitive performance. OWL is not a triple-based language and requires reification of axioms when using a triple-based language, whereas SPARQL[2] is a triple-based query language for RDF graphs. Research studies have suggested SPARQL as the right candidate for ontology-based applications to check integrity constraints based on the closed-world assumption [7]. In this regard, many previous research studies have attempted to transform OWL/SWRL into SPARQL queries; however, some of these approaches presented a complex indirect transformation of OWL/SWRL axioms to SPARQL queries with limited expressivity [8,9], while others were platform-specific [10].

MDE increases the level of abstraction to help deal with complexity and handles risk by specifying integrity constraints at an early stage of the system development lifecycle [11]. In addition, it allows to build a platform-independent model to generate the code for more than one platform. Therefore, motivated by the work presented by [12] and [13], in this paper, we propose a formal and flexible MDE-based methodology to integrate closed-world constraint checking with open-world reasoning. This work focuses on modelling OWL and SWRL rules to generate SPARQL queries to check closed-world integrity constraints. Some constraints (such as unique and contextual constraints discussed below), cannot be expressed in OWL. These constraints can be expressed with SWRL; moreover, they should be checked with SPARQL. Therefore, SWRL rules are included in the ontology model to capture these constraints, which are then transformed into SPARQL queries at the code level. Thus, the existing OWL reasoners that support SPARQL query answering will be used to check closed-world constraints for semantic validation of data maintaining their integrity.

To demonstrate the usefulness of our methodology, we have implemented it on a real-world water case study. Water Supply and Distribution Systems (WSDS) are experiencing a significant improvement in the availability of observational data due to continuous environmental monitoring via either in situ or remote sensing [14]. Recent advancements in information and communication technologies have provided the environmental community with continuously evolving software solutions for the increasing volumes of data often referred to as big data for their access, integration, sharing, publishing, and analysis [15]. While performing these tasks, the semantics of the data becomes highly heterogeneous as it originated from different data sources [16,17]. For the semantic interoperability of the data, the water case study (detailed in Section 5) developed an ontology. Therefore, the semantic validation of data was crucial by checking integrity constraints imposed on these data to ensure data consistency. Without semantic validation of these data, the integrity of the data could be violated, which could lead to a negative impact on the economy. The requirements of this case study inspired us to perform this research.

Our main contributions are:

1. The use of MDE for closed-world integrity constraint checking for ontology-based data-centric applications. For this purpose, we included SWRL rules in the ontology model to represent integrity constraints based on CWA. Moreover, the semantics of these constraints are made explicit through their annotation.
2. Flexible semantic validation. As discussed in Section 4.4, users can either use textual constraints (OCL) from the model or generated code to check integrity constraints.
3. Direct transformation of the SWRL rules into SPARQL queries at the code level for the closed-world integrity constraint checking using existing reasoners.
4. Finally, we demonstrated our methodology through a real case study using water observations data for their semantic validation.

The rest of the paper is organised as follows. In Section 2, we introduce the syntax and semantics of SWRL, integrity constraints in SWRL/OWL, MDE, and integrity constraints in MDE. The related work is reviewed in Section 3, and stepwise methodology is proposed in Section 4. The methodology is demonstrated via a water case study in Section 5 and Section 6 concludes the paper.

2. Preliminaries

This section overviews the syntax and semantics of SWRL and MDE to represent the integrity constraints in SWRL and OCL, a popular textual constraint language in the MDE field.

2.1. Syntax and semantics of SWRL

SWRL [3] extends OWL with the addition of Horn-style rules. Due to space limitations, we only present the syntax and semantics of SWRL and refer the reader to [18].

The purpose of including the SWRL rules is to capture those integrity constraints, which could not be expressed as OWL axioms, e.g. unique [9] and contextual constraints explained in the next section. Similar to all the other rule-based representation languages, a rule constitutes a body (antecedent) and a head (consequent), where if the rule's body is true, then the rule's head should be true. Both body and head of the rule are conjunctions of atoms. The abstract syntax of SWRL is given in Fig. 2.1. The abstract syntax is in the form of Extended BNF where terminals are quoted, and non-terminals are bold and unquoted; alternatives are either separated by (|) or are given in a different production. The components enclosed in square brackets ([…]) can occur at most once, and those in braces ({…}) can occur any number of times, and whitespaces are ignored. The abstract syntax of SWRL helped us to build the model in Section 4.2.

An atom can be in the form of *C(i)*, *D(v)*, *R(i, j)*, *U(i, v)*, *sameAs(i, j)*, *differentFrom(i, j), or builtIn(p,*$v_{1,\ldots,}v_n$*)*, where *C, D, R, and U* represent a Class, datatype, object property, datatype property respectively; *p* is a built-in predicate, i.e. a built-in function; *i* and *j* are OWL individuals or variables, and *v*, $v_{(n)}$ are data values or variables. The built-in functions in SWRL are for mathematics, string manipulation and value comparison. The SWRL semantics extends the OWL semantics. For an abstract OWL interpretation, *I*, a binding B(*I*) is an abstract OWL interpretation that extends *I* such that variables i and v map to elements of the object or datatype interpretation domains, and built-ins are mapped to a subset of n-ary relations over the datatype interpretation domain, i.e. built-ins have fixed mapping. The notation $\in$ means belongs to or is an element of, $=$ refers to equals and $\neq$ means not equals to. According to the condition on interpretations given in Table 1, an Atom is satisfied by a binding B(*I*). A binding B(*I*) satisfies a body of the rule if it is empty, or B(*I*) satisfies every atom in the body. A binding B(*I*) satisfies the rule's head if it is not empty, and B(*I*) satisfies every atom in the head. A rule is satisfied by an interpretation *I* if for every binding B such that B(*I*) satisfies the body, and B(*I*) *also* satisfies the head [3,9].

2.2. Integrity constraints in SWRL

As mentioned above, SWRL/OWL inherit OWA and non-unique name assumption (NUNA). Let us assume that p is an individual

[2] https://www.w3.org/TR/rdf-sparql-query/.

```
axiom ::= rule
rule ::= 'Implies(' [ URIreference ] { annotation } antecedent consequent ')'
antecedent ::= 'Antecedent(' { atom } ')'
consequent ::= 'Consequent(' { atom } ')'
atom ::= description '(' i-object ')'
       | dataRange '(' d-object ')'
       | individualvaluedPropertyID '(' i-object i-object ')'
       | datavaluedPropertyID '(' i-object d-object ')'
       | sameAs '(' i-object i-object ')'
       | differentFrom '(' i-object i-object ')'
       | builtIn '(' builtinID { d-object } ')'
builtinID ::= URIreference
i-object ::= i-variable | individualID
d-object ::= d-variable | dataLiteral
i-variable ::= 'I-variable(' URIreference ')'
d-variable ::= 'D-variable(' URIreference ')'
```

Fig. 2.1. Abstract syntax of SWRL [3].

Table 1
Conditions on interpretation of SWRL atoms.

SWRL atom	Condition on interpretation
C(i)	$i^I \in C^I$
D(v)	$v^D \in D^D$
R(i, j)	$(i^I, j^I) \in R^I$
U(i, v)	$(i^I, v^D) \in U^I$
builtIn(p, $v_1, \ldots, v_n$)	$(v_1^D, \ldots, v_n^D) \in p^D$
sameAs(i, j)	$i^I = j^I$
differentFrom(i, j)	$i^I \neq j^I$

of type Person, and that class Person is described by the following axiom:

$$Person \sqsubseteq \exists hasID.ID \tag{2.1}$$

Then, using axiom (2.1) OWL reasoning infers that an individual *Id* exists such that object property assertion '*hasID(p, Id)*' and class assertion '*ID (Id)*' are true; if the existence of *Id* is not explicit, then its existence will be implied. CWA assumes that we have a complete knowledge about the domain. Thus, according to CWA, to satisfy the axiom (2.1), all of the instances of Person (p) will need to be given ID (Id) and object property hasID (p, Id) will be true. Suppose a person has an Id1 and Id2, which are not stated as different.

$$Person \sqsubseteq \leq 1 hasID.T \tag{2.2}$$

Then, according to the axiom (2.2), Id1 and Id2 will be inferred to be the same because OWL/SWRL does not follow UNA. Instead of drawing the inference, however, ontology-based systems will prefer to have CWA/UNA treating axiom (2.2) as an integrity constraint for inconsistency detection. Mostly the SWRL/OWL reasoners follow the open-world reasoning only and are not suitable for checking closed-world constraints [6].

The most common constraints are type, cardinality, key, unique, and contextual. These constraints are detailed in [9]. Most of the constraints can be expressed with either OWL or SWRL, whereas some constraints can only/better be expressed with SWRL, i.e. unique constraints and contextual constraints [9] are discussed in this section:

Unique Constraints:

The unique constraints check that the value of a property for an object is unique. For example, the constraint that two persons should not have the same IDs can be expressed in SWRL as:

$$Person\,(?p1) \wedge hasID\,(?p1, ?Id1) \wedge Person\,(?p2) \wedge hasID\,(?p2, ?Id2) \wedge swrlb:equal\,(?Id1, ?Id2) \rightarrow sameAs(?p1, ?p2) \tag{2.3}$$

This constraint (2.3) cannot be expressed as OWL axiom as OWL expresses tree-like interdependencies of variables only [19].

Contextual Constraints

Contextual constraints apply the restriction on the value of one attribute according to the values of other properties. In other words, the value of one property of an object depends on the values of other properties of the same object. The co-occurrence or co-constraints [20] in XML are also contextual constraints. Standard Markup languages could not capture these constraints. To illustrate the constraint that a person with a certain job type (Professor) should be in the department (Education), we can employ the following rule**:**

$$Person\,(?p1) \wedge hasJobtype\,(?p1, Professor) \rightarrow Department(?p1, Education) \tag{2.4}$$

2.3. Model-driven engineering

MDE suggests constructing a model to describe a system's abstraction first, and then transform this model into a real and executable system, i.e. source code. The models are represented by the modelling languages, which are defined by meta-modelling languages. A modelling language comprises an abstract syntax, semantics, and at least one concrete syntax.

In this context, Model Driven Architecture (MDA) of Object Management Group's (OMG) [21] is used widely for MDE. Its main idea is to abstract the Platform Independent Model (PIM), a conceptual model based on visual diagrams such as Unified Modelling Language (UML) [22], which can describe the entire business function and is not concerned with the implementation details and techniques. Then, according to different implementation techniques, multiple transformation rules are formulated. These rules and assistant tools help transform PIM into PSM (Platform Specific Model), and finally, the PSM is transformed into the code. Meta-Object Facility (MOF) [23] is OMG's standard to define meta-models. MOF's version 2.0 provides two metameta-models: Essential MOF (EMOF) and Complete MOF (CMOF). The four-layer architecture of MDA is shown in Fig. 2.2. The Object Constraint Language (OCL) [24] aims to provide a class diagram with additional information that UML diagrams cannot expose. OCL allows the definition of integrity constraints for the model. As a modelling language, modelling constructs are first-class citizens in OCL, and the OCL code is considerably more compact than the other programming languages.

Another meta-model is provided by Ecore [26], which is used in the Eclipse Modelling Framework [27]. Ecore resembles EMOF, a subset of MOF. Therefore, both terms metameta-model as in [26] and meta-model as in [27] are used for Ecore. EMF is a modelling framework and code generation utility for building

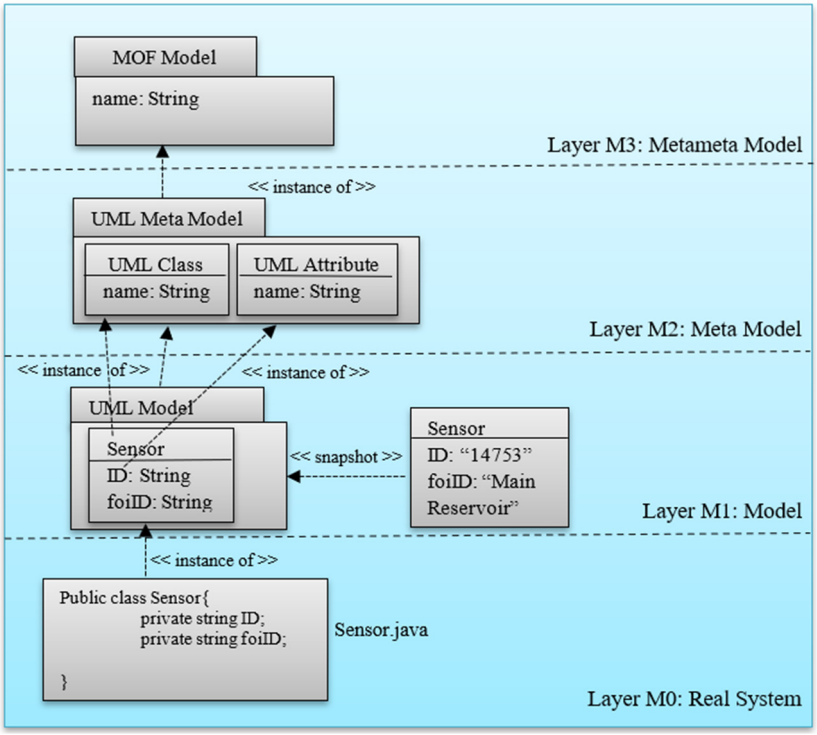

Fig. 2.2. The four-layer architecture of MDA [25].

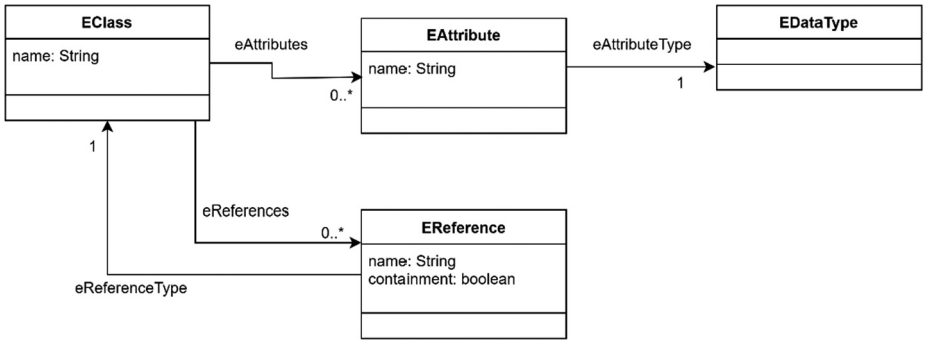

Fig. 2.3. Subset of Ecore meta-model.

applications and tools based on a structured data model. The EMF core's modelling subprojects provide abstract and concrete syntax development, model-to-model transformation, model-to-text transformation, database integration and graphical editor generation [28]. A subset of this Ecore's meta-model is shown in Fig. 2.3 , which is used in this research to build the model. It provides four basic constructs: (1) an EClass is used to represent a modelled class with a name, zero or more attributes and zero or more references; (2) an EAttribute is used to define a modelled attribute with a name and a type; (3) an EReference is used to express one end of the association between classes. It has a name, a Boolean flag to indicate whether it expresses a containment, and a reference (target Class) type; (4) an EDatatype is used to define the type of an attribute. A data type can be a primitive such as int and float or an object type such as java.util.Date. Eclipse OCL is based on OMG's OCL 2.4 specification to be used with the Ecore and UML meta-models [29].

2.4. Integrity constraints in MDE

Modelling languages allow specifying structural constraints such as unique and multiplicity. These are often defined by the Class properties and relations in the model. For example, the requirement that 'a person has exactly two parents' in the model can be expressed by the multiplicity constraint, which utilises the '*Lowerbound*' and '*Upperbound*' properties inherited by class EReference of the Ecore for the EClass association Fig. 2.4(a).

The instance of the model should satisfy the multiplicity constraint by allowing to have exactly two parents. However, complex constraints (attached constraints) are usually defined by textual constraint languages such as OCL. For example, the requirement that 'a person cannot be a parent of him/herself' which, is an irreflexive property of a concept in the ontology, can be satisfied by an attached constraint in OCL expression Fig. 2.4(b). This constraint will be attached with the model. Irreflexive property is elaborated via an instance of the model in Fig. 2.4(c). In terms of constraint expression, the model can express structural closed-world integrity constraints such as unique and typing constraints. The typing constraints restrict the individuals' type that participates in relation or values' type allowed for a property. However, the constraints defined on multiple structural properties must be expressed through OCL on the model, such as contextual constraints or pattern constraints. Even though we can express

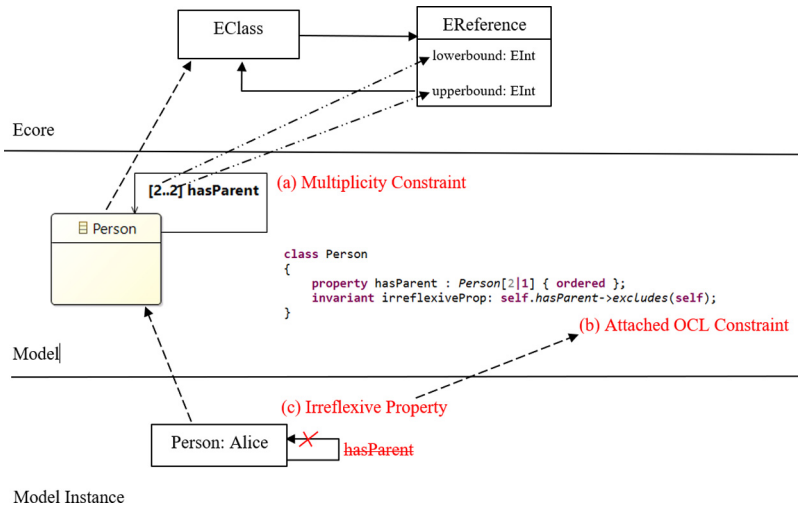

Fig. 2.4. (a) Multiplicity constraint defined on the Person. (b) An Attached Constraint in OCL. (c) Irreflexive property of the class Person.

unique and contextual constraints through OCL, our requirement in this paper is to use existing OWL reasoners to check the closed-world integrity constraints. Thus, we aim at transforming SWRL rules into SPARQL ASK queries at the M0 level. Code generation could be performed only from the entities in the model and not from OCL. Transformation of OCL into SPARQL will require to include entities into the model for OCL instead of ontology, which is an interesting direction for future work. As we are focusing on addressing the problem for ontology-based applications, we included ontology in the model.

3. Related work

Integrity constraints are mostly checked to analyse relational databases. However, research studies in the Semantic web have shown interest to include the integrity constraints to validate the data. One such task is presented by [6]. The authors compared relational databases and OWL on the basis of their schema languages and computation problems and extended the OWL with integrity constraints that behaved similarly to the statements in relational databases. Their work was based on a minimal Herbrand model semantics of OWL. Given an OWL KB K containing an ABox and a TBox, and a constraint TBox C, an axiom in the constraint TBox is satisfied by K if all minimal Herbrand models satisfy it. Another approach given by [30] was to extend DLs with autoepistemic operators to model integrity constraints. The authors presented Description Logics of minimal knowledge and negation as failure (MKNF-DLs) that combined closed-world and open-world reasoning with modal operators for a formal characterisation of non-monotonic features. Kharlamov et al. [31] developed a tool for creating ontology-based models and capturing integrity constraints as OWL 2 RL axioms. The constraints were translated into Datalog programs with stratified negation as a failure for query answering and data validation. They conducted an evaluation of the tool over data from two industrial use cases. Fang [32] presented and applied an approach on biomedical ontologies for maintaining integrity constraints throughout the lifecycle of OWL, including the processes of OWL generation and maintenance. For OWL generation, a paraconsistent model was used for maintaining integrity constraints during the process of translation of relational database to OWL. A rule-based language with a set extension was used as a platform for constraint specification.

An approach by [8] presented the combination of closed world constraint validation with the open-world reasoning of OWL through a query translation technique. This approach represented integrity constraints as OWL axioms that were translated into SPARQL queries to use the existing OWL reasoners for constraint validation. To address this approach's limitations, [10] presented a platform-specific method by extending the translation rules with the inclusion of data types and data values for constraint representation without a formal description of constraint semantics and translation rules. This extended approach was used in [9] to formalise the description of the semantics of OWL/SWRL constraints and translation rules. The strategies presented by [8,9] used the indirect transformation, i.e. from SWRL to $\text{DCQ}^{\text{not+}}$ and from $\text{DCQ}^{\text{not+}}$ to SPARQL. A DCQ^{not} is a Disjointed Conjunctive Query with Negation As Failure (NAF) operator ("not"). Consequently, these approaches had a limitation to express the knowledge base and the constraints pair to either <SROIQ, SROI> or <SRI, SROIQ>. Patel-Schneider [7] demonstrated that DL axioms could be interpreted and be used for both constraint checking and closed-world recognition when information sources are expressed as RDF or RDFS. The constraint checking could be effectively implemented using a translation to SPARQL queries. Lausen et al. [33] extended RDF by a set of constraints such as primary and foreign key restrictions for relational data mapping to RDF and demonstrated that SPARQL was able to deal with all types of constraints. The Shapes Constraint Language (SHACL[3]) [34] is recommended by W3C for validating RDF graphs against a set of constraints. These constraints were provided as SPARQL queries or shapes.

Due to the complexity of commercial products, the verification and validation procedures are quite challenging. To deal with this problem, Model-based Systems Engineering (MBSE) was applied through the formal application of models in the system development life cycle to help with system's verification activities [11]. For managing the complex system monitoring and fault diagnosis and verification, [35] proposed a combination of data-driven and model-based methods to improve fault detection. Given a complicated system such as automotive, a framework was proposed by [36] to use ontology for fault detection. The use of Model-driven technologies for ontology-based systems for semantic unification and verification has emerged in many research studies. Our methodology in the next section is based on the approach presented by [12,13]. The reason for selecting their approach was that the domain model was separated from the ontology (semantic) model, and the annotations were made explicit by using a

[3] https://www.w3.org/TR/shacl/.

separate annotation model. So, the domain ontologies and models could evolve asynchronously. However, our methodology is extended and overcame their limitations, such as domain-specific meta-model and confined constraints expressivity to ontological concepts and properties with no code generation. Similarly, Parreiras and Staab [37] integrated MOF-based meta-models for UML and OWL and presented a framework named Transforming and Weaving Ontologies and UML in Software Engineering (TwoUse). One of the building blocks of this framework was a specification of SPARQL-DL Abstract Syntax (SPARQLAS), which employed OWL Functional-Style Syntax to compose queries. The model-to-model transformation was performed from SPARQLAS to SPARQL [38]. An approach proposed by [2] used constraint-aware model transformation, which supported constraint specification in the definition of transformation rules. The authors used this approach for the specification of data validation constraints [39] in OCL at the model level. The specified constraints were mapped with Java annotations, which were transformed to tests that could be executed by an existing framework for data validation. The authors in [40] presented an ontology-based verification method where the UML class diagram was represented as OWL, and OCL constraints were transformed into SPARQL NAF queries. From research studies mentioned above, it is clear that SPARQL is the right candidate for constraint checking for ontology-based applications. For the validation of integrity constraints at model level only, SWRL rules could be mapped to OCL statements. One such approach was presented by [41] where they described syntactic mapping from SWRL to OCL and as a result of this mapping they introduced a subset of OCL, i.e. OCL-Lite. OCL-Lite could be used to express rules, which later could be mapped into the SWRL. A similar approach was presented by [42], where the authors used REWERSE Rule Markup Language (R2ML) as a pivotal metamodel for interchanging between OWL/SWRL and UML/OCL. For model transformations, they used ATLAS Transformation Language (ATL[4]). The authors of [43] proposed the direct transformation of OWL/SWRL to UML/OCL.

Many research studies have highlighted the advantages of semantic annotation on the design models and are focused on a range of perspectives. For example, sharing and reusing of the knowledge for the data integration [44,45], domain-specific knowledge interoperability such as computer-aided design models [46] and to assist analysts and designers in managing their models [47]. The authors in [48] proposed an approach for semantic interoperability between stakeholders by annotating the enterprise models with an ontology. The structure of semantic annotation was based on its type (decoration, link, Instance Identification, etc.), textual description, location, formal, and graphical description. The informal validation was performed by checking whether the annotations had conveyed the correct meaning to the model. Another method was presented by [49] where the authors used the XMI version of the UML class diagram to annotate class attributes with ontology concepts by generating annotation files using the XSLT templates. They gave the idea of using annotation in the code generation if required. An approach was proposed by [50] in which the authors used semantic annotation to bring the semantic interoperability between the models during the product life cycle management. They used two types of ontologies to semantically enrich the model. Their method of verifying the semantics of the annotated elements was semi-automatic based on the likeness comparability between the two-domain semantics of a common annotated model element. However, rarely semantic annotation of the design models dealt with the semantic validation of the constraints imposed on the domain.

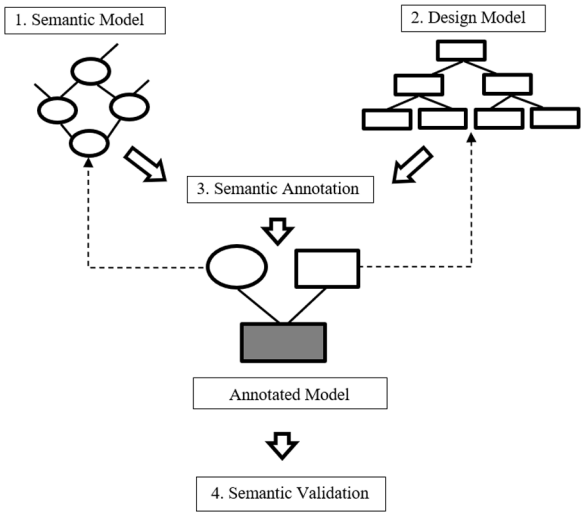

Fig. 4.1. Four-step methodology. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4. Methodology

The methodology proposed in this paper (Fig. 4.1) is based on approaches presented in [12,13], where the authors annotated domain models by references to the domain ontologies. Our methodology is enhanced by making the model generic and including further concepts in the ontology model, such as SWRL rules, to express the integrity constraints. By generic, we mean that it is a domain-independent model. It can be used for any domain with that domain's ontology. The name of the domain is not mentioned in the model. Instead, the "name" is added as an attribute of the domain and ontology model, explained in the next subsection. We express the closed-world semantics of the SWRL rules based on the explicit data present in the ontology, i.e. ABox assertions [51]. These rules are translated into OCL expressions for semantic validation of data at the model level and SPARQL queries are generated for each integrity constraint for the semantic validation of data using existing OWL reasoners.

The methodology consists of the following four steps:
(1) semantic model,
(2) design model,
(3) semantic annotation, and
(4) semantic validation.

4.1. Step 1: Semantic model

The first step involves formalising the domain knowledge in the form of a semantic model, i.e. ontology. The semantic model can be designed by domain engineers with the collaboration of the users using the domain knowledge. The knowledge can be represented by the ontology expressed in languages such as Web Ontology Language (OWL[5]) recommended by the W3C and formulated using software such as Protégé [52]. The ontology consists of entities and relations between those entities. It can include the rules and restriction axioms to define the integrity constraints imposed on the domain. The semantic model is built first so that domain-specific elements could be included in the model, if required, in the next step. This step can be performed by building ontologies from scratch or by using the existing ones. If these are not available already, this step can be eliminated

4 https://www.eclipse.org/atl/.

5 https://www.w3.org/TR/owl-ref/.

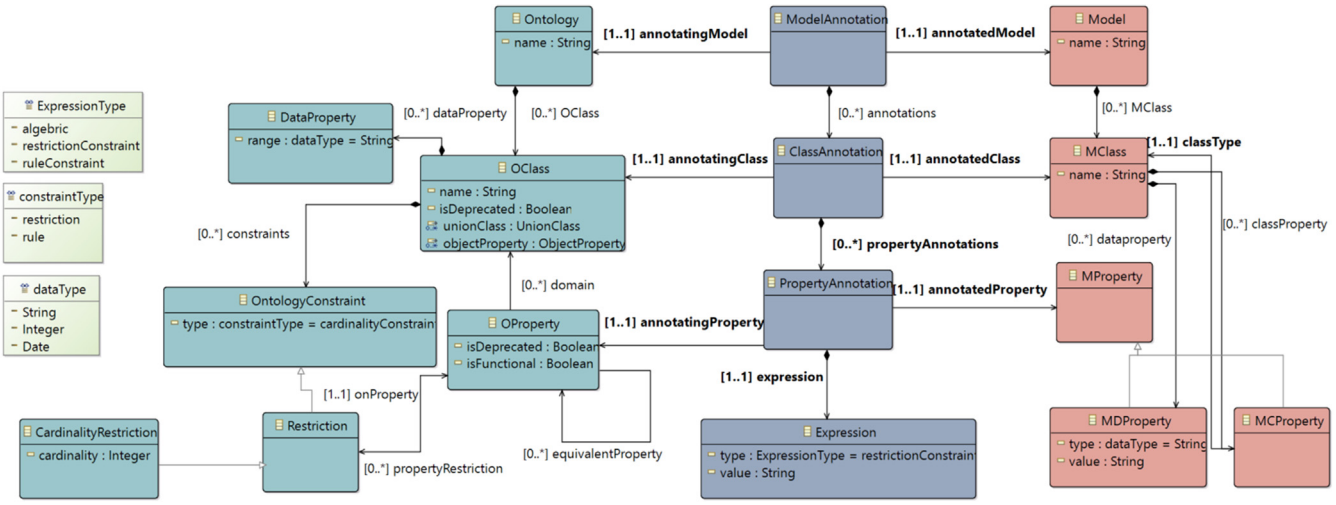

Fig. 4.2. Generic model.

because the ontology is modelled in the next step, which can allow the generation of .owl file as a result of generated code in the last step. Automatic generation of .owl is out of scope for this paper.

4.2. Step 2: Design model

The second step involves designing the model, i.e. Platform Independent Model (PIM). The generic model constitutes an independent technological specification of a system. The model is developed using EMF in Eclipse IDE. It consists of three separate parts shown in Fig. 4.1 Domain's ontology model (left part coded in cyan colour), is used to annotate the domain model, (2) Domain model (right part depicted in red colour), and (3) A separate annotation model (middle part shown in blue), provides a means of domain model linkage with the ontology model for the semantic annotations (discussed in next subsection). The domain's ontology model includes the concepts, relationship between the concepts, properties and constraints in the form of rules and restrictions. SWRL rules are added in the ontology model, shown in Fig. 4.3, to express the integrity constraints with careful consideration of the abstract syntax of SWRL given in Fig. 2.1. The rule contains a body (antecedent) and head (consequent). Both the body and the head consist of several atoms, which could be zero, as depicted by the multiplicity in Fig. 4.3. A rule states that if all the atoms in the body of the rule hold, then the head of the rule also holds. An empty body is regarded as trivially true, whereas an empty head is regarded as trivially false.

In the domain's ontology model, the class *Atom* is made as an abstract class. All the types of *Atom* such as *ClassAtom*, *DataRangeAtom*, *ObjectPropertyAtom*, *DataPropertyAtom*, *SameAs*, *DifferentFrom* and *BuiltIn* are defined as subclasses of class *Atom*. Class *Variable* is made an abstract class. The *IndividualVariable* class and *DataVariable* class are inherited from the *Variable* class to declare variable of type Individual and datatype. Due to the space limitation, some atoms are defined as follows:

- The *ClassAtom* has a relation *classRef*, which is an ontological class (*OClass*) and one *IndividualObject* (iobject), which can be of type *Individual* or *IndividualVariable*.
- The *ObjectPropertyAtom* has an *ObjectProperty* relation between two iobjects.
- The atom *BuiltIn* has a *builtinID*, a *symbol*, and one or more *DataObject* (dobject), which can be of type *DataValue* or *DataVariable*.

The domain's ontology model can be extended with further concepts in the ontology, and the domain model can be extended with domain-specific elements. The key concepts of the domain model are: (1) *Model* with an attribute *name* to represent the domain's name. The *Model* can have zero, one or several classes (*MClass*), (2) *MClass* represents a *Class* in the domain model, which can have zero, one or several properties (MProperty) with their type, (3) *MProperty* represents a property in *MClass* with its type. An *MProperty* can be of two types: *MDProperty,* which can be String, Integer, or Date; and MCProperty, which has an attribute classType, which refers to another *MClass*.

4.3. Step 3: Semantic annotation

In this step, the ontology model is used to annotate the domain model to explicit the semantics and to understand it in the right way. The generic model (Fig. 4.2) offers the linkage of the domain model entities with the concepts in the ontology. The middle part of the model is called *ModelAnnotation,* which is the entry point for the semantic annotation and links the domain model with the ontology model. It annotates the domain model with the ontology model by establishing an association. The *ClassAnnotation* associates the ontological entities with the domain model's classes. *PropertyAnnotation* defines a model's class properties by annotating them with the ontological properties. Finally, the *Expression* class provides the association by its type with *enum ExpressionType* that could be *restrictionConstraint, ruleConstraint,* or *algebraic*, by setting its description in the *value* field. The semantic annotation exposes the relationship between the model's classes and ontology's concepts and helped to interpret the model correctly because it integrates the domain knowledge into the domain model, making it self-explanatory. The annotation will help the programmers to perform the transformation process to get the desired output. It will also help the domain experts, as they are familiar with the domain knowledge.

4.4. Step 4: Semantic validation

The domain model enriched with the ontological annotations obtained in the previous step is validated in this last step. To provide the user with the flexibility of options, two types of semantic validations of data are performed: (1) semantic validation of the data by checking integrity constraints expressed on the domain model, (2) semantic validation of the data by checking integrity constraints generated as the code. For the semantic

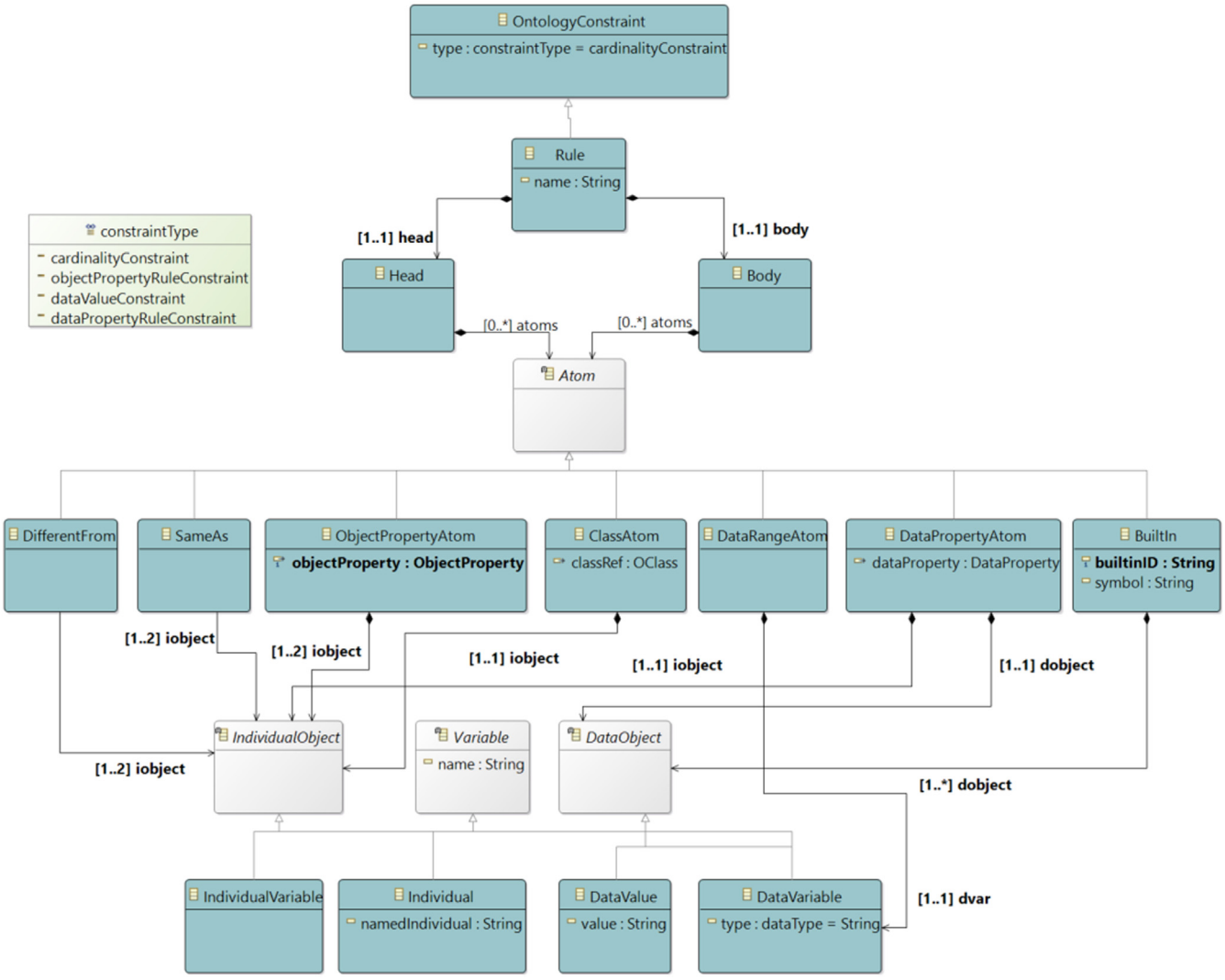

Fig. 4.3. Rules in ontology model.

validation of data at the model level, the integrity constraints are imposed on the domain model by the domain ontology through properties, relations, rules, or restrictions to determine whether they are valid. These constraints are translated into OCL scripts to validate the model when it is instantiated (M1). The translation of constraints into OCL is manual. The automated translation of constraints into OCL will be considered in the future work. This way, the user is given a flexibility in options to perform semantic validation of the data. OCL will allow the constraint checking at the model level in case code generation is not needed. If the requirement is to perform semantic validation of data at the code level, then code can be generated in any desired language.

4.4.1. Code generation

Certain mapping rules are formulated to generate the integrity constraints for the semantic validation of data. In our case, the SWRL rules are transformed into the SPARQL queries. To design the transformation rules, two types of rule constraints (Fig. 4.3): object property rule constraint and data property rule constraint, are added. As the name suggests, the rules that contain class atoms and object property atoms are given the type of "*objectPropertyRuleConstraint*" and the rest are given the type of "*dataPropertyRuleConstraint*". The transformation rules are given in Table 2, and the transformation of SWRL atoms to SPARQL ASK queries is given in Table 3.

The code is generated directly from the PIM using Model2Text (M2T[6]) transformations in the desired programming language, i.e. SPARQL. These transformations use a template-based approach where a *Template* is a text template with placeholders. The data can be extracted from models through these placeholders. The placeholders are expression specified over model entities, with queries being the primary mechanisms for selecting and extracting the values from models. In this paper, Acceleo[7] is used for these transformations. During the mapping, the concepts from the source model are linked to the concepts in the target code.

5. Semantic validation of water observational data

In this section, the four-step methodology proposed in the previous section is implemented for the semantic validation of the water observations data used for a WatERP[8] case study.

A successful WSDS meets the requirements of quality, water demand, and distribution system. These requirements include maintaining water pressure ensuring the durability of the water supply resources. Information collection, system monitoring, and

6 https://www.omg.org/spec/MOFM2T/1.0/PDF.
7 https://www.eclipse.org/acceleo/.
8 https://eurecat.org/en/portfolio-items/waterp/.

Table 2
Transformation rules.

Rule	Transformation rule	Source (PIM)	Target (SPARQL)
1	Rule2Query	Rule	Ask Query
2	RuleName2QueryName	Rule.Name	Query.name
3	RuleComment2QueryComment	EClass = '*Expression*' Attribute = '*Comment*'	# SPARQL Comment
4	Rules of type ObjectPropertyRuleConstraint	Atoms in Body Atoms in Head	Ask WHERE {Atoms in body} Filter EXISTS {Atoms in head}
5	Rules of type DataPropertyRuleConstraint	Atoms in Body Atoms in Head	Ask Where (Atoms in body except dataRange/dataPropertyAtoms) filter (dataRange/dataPropertyAtoms/atoms in head)
	Atoms:		
6	ClassAtom2Triple	IndividualObject ClassRef	?var Predicate = rdf:type Class Name
7	ObjectPropertyAtom2ObjectPropertyTriple	objectProperty IndividualObject	Individual/IndividualVariable Predicate = object property Individual/IndividualVariable
8	DataPropertyAtom2DataPropertyTriple	Individual Object dataProperty data Object	Individual/individualVariable Predicate = data property DataValue/dataVariable
9	DataRangeAtom2FilteronDataType	dvar Datatype	?dataVariable datatype(?dvar) = dataType

Table 3
Transformation of SWRL to SPARQL.

SWRL atom	Source (Model)	Transformation	SPARQL query (Target)
Class Atom = C(i)	i = iobject, C = classRef	?iobject rdf:type classRef	?i rdf:type C
DataRangeAtom = D(v)	v = dvar D = datatype	Filter(datatype(?dvar) = xsd:datatype)	Filter (datatype(?v) = D)
objectPropertyAtom = R(i, j)	i, j = iobject R = objectProperty	?iobject objectProperty ?iobject	?i R ?j
dataPropertyAtom = U(i, v)	i = iobject, v = dataobject U = dataProperty	?iobject dataProperty ?dataobject	?i U ?v
sameAs i = j	i, j = iobject	?iobject owl:sameAs ?iobject	?i owl:sameAs ?j
DifferentFrom i ≠ j	i, j = iobject	?iobject owl:DifferentFrom ?iobject	?i owl:differentFrom ?j
builtIn(p, $v_1, \ldots, v_n$)	p = builtInID $v_1, \ldots, v_n$ = dobject symbol = function symbol	filter (builtInID(?dobject1 symbol ?dobject2))	filter(builtInID(?v1 symbol ?v2))

data exchange occur in each part of the WSDS architecture to inspect water flow, volume, and pressure for ensuring proper operations and detect system abnormalities. The managers require information from different parts of the WSDS system for decision-making and planning procedures. A WSDS aims at supplying water to consumers under sufficient pressure. The water pressure is maintained between the minimum and maximum acceptable levels for reliable and safe operation [53]. High-pressure systems are prone to cause pipe breaks, increased energy usage and leakage, whereas low-pressure systems cause users inconveniences in performing routine tasks [54]. The WatERP data integration framework [55,56] overcame the problems of data heterogeneity, lack of management perspective and unavailability of semantic linkage of data in the ontology by developing Water Management Ontology (WMO) [57,58]. The water data were integrated using hydrological standards presented by Open Geospatial Consortium (OGC), such as Sensor Observation Service (SOS[9]) in the Water Data Warehouse (WDW). As a result, a Knowledge-Based Water Decision Support System (WDSS) was developed to perceive the water cycle from the management perspective using WMO information. The WMO information was then supplied to the different inference engines, such as the Rule-Based reasoning engine (RBR) that was used to support the management objectives. The RBR inference engine aimed to apply water resource manager experience to improve water allocation to satisfy different water usage demands. The water manager experience was represented in the form of a set of rules, and the allocation of water resource (recommendation) was generated by applying RBR. The rules were in the form of ontology restrictions stored in the rule set repository. The facts definition part of RBR generated facts by reading WML-OWL file using SPARQL queries and converted them into JAVA objects [59]. The method of facts generation is out scope in this paper. Our primary motivations to apply the proposed methodology on this case study are:

1. Semantic validation of water data before it was aggregated in WDW.
2. Expressing the rules in SWRL for the rule set repository in RBR

The proposed MDE-based methodology offers the integration of rules in the model to generate the semantic validation code in Java for data integration framework and SPARQL queries for fact definition in RBR. Several integrity constraints imposed on water observations data were identified, such as cardinality, typing, pattern, range and contextual. In this work, we focus on integrity constraints represented by rules to generate SPARQL queries for

[9] https://www.ogc.org/standards/sos.

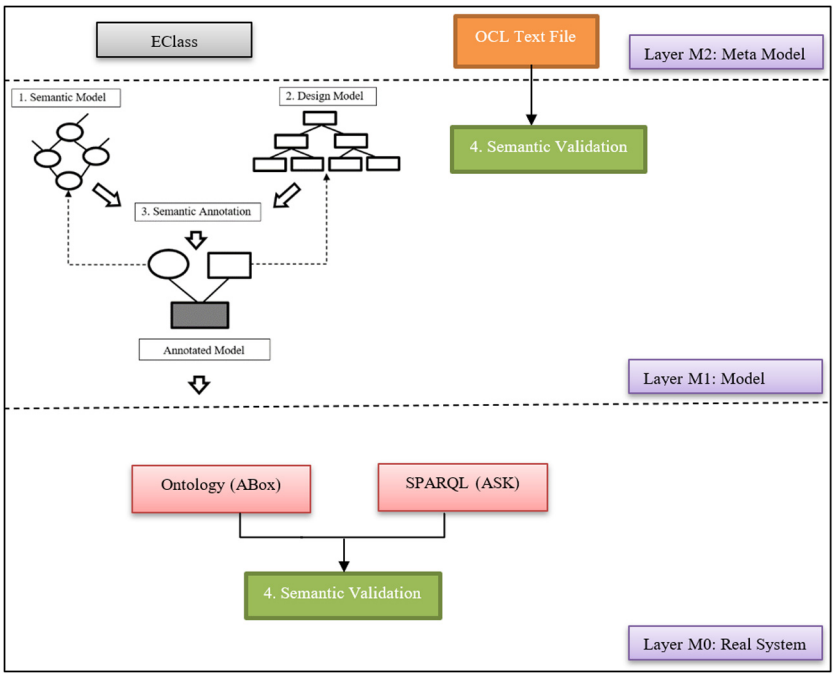

Fig. 5.1. The implementation of four-step methodology.

the semantic validation of water data using existing semantic reasoners by evaluating the SPARQL queries. The contextual constraints regarding maintained water pressure for reliable WSDS are mainly discussed. Since WMO was developed from a management point of view, in case of a high-water pressure in pipes at any particular time, an alert should be notified in the system to take a suitable action. To raise such an alert, rules are added to enhance the WMO in the first step of the methodology. The implementation of the methodology in the context of MDE is given in Fig. 5.1.

5.1. Step one: Semantic model

The first step of the methodology involved building a new semantic model or use existing ontologies. In this step, we have enhanced the existing WMO.

5.1.1. Enhanced WatERP ontology

The semantic structure of WMO mainly was based on SSN ontology.[10] To align the SSN ontology with the OGC-SOS architecture, several standard ontologies were used, such as Geo-SPARQL schema[11] ("Geo ontology") and shecma.org[12] [60]. In the WMO, the '*Unit*' was considered as a data property for the '*Observations*'. To enhance the WMO and to represent the constraints related to a sensor, we consider '*Sensor*' as a separate entity because sensors are deployed at the feature of interest to observe a phenomenon and transmit time-series data. This entity is aligned with the SSN ontology. A sensor has two data properties: '*hasURI*' and '*hasUniqueID*' of types string and integer. The entity '*Sensor*' with its two properties and restrictions is given in Fig. 5.2. Moreover, the '*Unit*' is defined as a separate entity with an object property '*hasUnit*' within the domain '*Results*'. A necessary restriction of the type universal is also defined on '*Results*' (5.1).

$$Results \sqsubset \forall\ hasUnit\ Unit \tag{5.1}$$

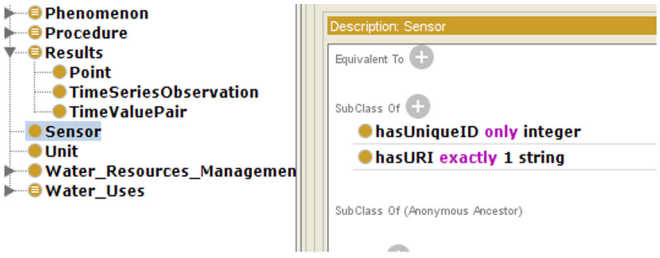

Fig. 5.2. The entity "Sensor".

A segment of WMO [60] shown in Fig. 5.3 is updated with 'Results' and '*Unit*' entities with a relationship of '*hasUnit*' between them and the relation between 'Observation' and 'Results' entities is 'hasObservationResults'. The relation 'ObservationResult' is an obsolete relation from the SSN ontology. The enhanced WatERP ontology consists of 547 Class, 21 object property, 12 data property, 5425 Axiom, 3827 logical axiom, 25 rules and 29 individuals. The ontology covers all the concepts related to water resource management, water uses, water supply and distribution. This ontology is used as a reference ontology for the semantic annotation in the third step of the methodology.

Rules

For the semantic annotation and validation of the contextual constraints, unique constraints, range constraints and type constraints, DL-Safe rules (Fig. 5.4) are added in the ontology because OWL and standard Markup languages cannot capture constraints such as contextual. As mentioned earlier, SWRL is the combination of OWL and function-free Horn logic. The result is very expressive formalism but, unsurprisingly, undecidable. Decidability can be regained if a safety condition can be imposed on them. This safety condition is known as 'DL-Safety', and such rules are called the 'DL-Safe' rules [51]. The closed-world semantics of these constraints involved only named individuals and data values. Similar to all the other Rule-based representation languages, a DL-Safe rule constituted a body and head. The safety

10 https://www.w3.org/2005/Incubator/ssn/ssnx/ssn.
11 http://www.opengeospatial.org/standards/geosparql.
12 http://schema.org/.

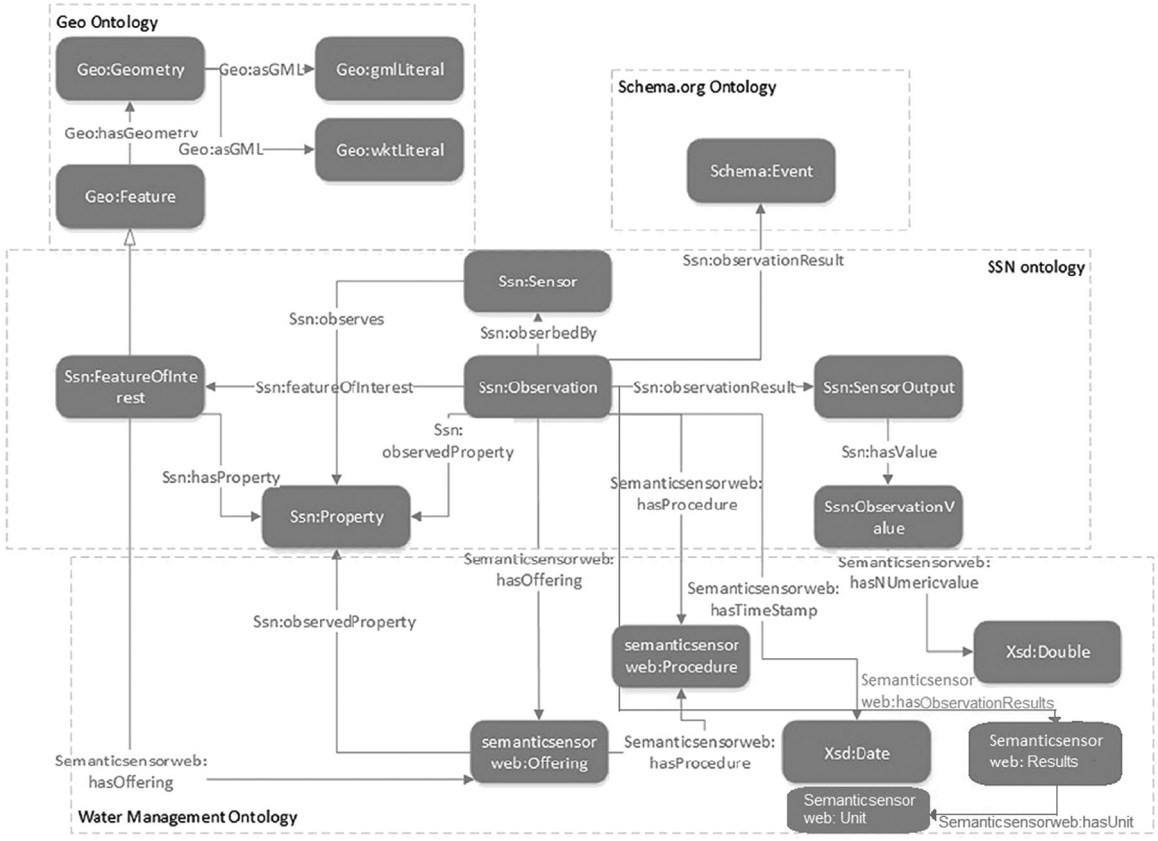

Fig. 5.3. A segment of WMO.

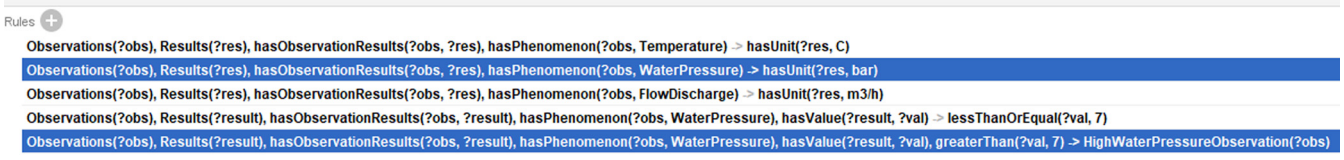

Fig. 5.4. DL-Safe rules.

$$Observations\,(?obs)\,,\ Results\,(?result)\,,\ hasObservationResults\,(?obs,\ ?result)\,,\ hasPhenomenon\,(?obs,\ WaterPressure)\,,$$
$$hasValue\,(?res,\ ?val)\,,\ greaterThan\,(?val,\ 7)\ \longrightarrow\ HighWaterPressureObservation\,(?obs) \tag{5.2}$$
$$Observations\,(?obs)\,,\ Results\,(?res)\,,\ hasObservationResults\,(?obs,\ ?res)\,,\ hasPhenomenon\,(?obs,\ WaterPressure)$$
$$\longrightarrow\ hasUnit\,(?res,\ bar) \tag{5.3}$$

Box I.

condition in the DL-Safe rules states that variables that appeared in the body of the rule are permitted to exist in the head of the rule. Moreover, the individual variables in a rule are bound only to the individuals explicitly named in the ontology [61]. Eqs. (5.2) and (5.3) are given in Box I.

The rules are defined on the entity '*Observations*'. The DL-Safe rule about the maximum water pressure in the pipe is a range constraint with a defined range of data values typed as '*dataValueConstraint*' is shown in (5.2) and a contextual constraint typed as 'objectPropertyRuleConstraint', i.e. if the observation is having phenomenon '*WaterPressure*' the unit in the '*Result*' should be in '*bar*' as given in (5.3).

The rule's body stated the following:

- Observations(?obs), i.e. 'obs', is a variable for the class '*Observations*'.
- Results(?result), i.e. 'res', is a variable for the class '*Result*'.
- hasObservationResults(?obs, ?result), i.e. the object property '*hasobservationResults*', is between the classes '*Observations*' and '*Results*' through the variables '*obs*' and *'result'*.

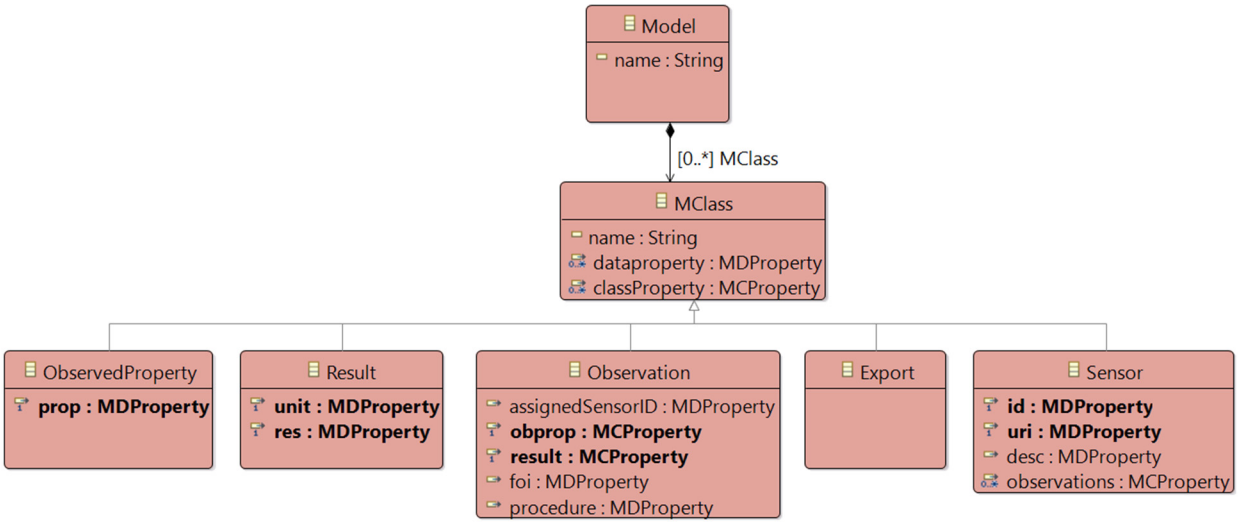

Fig. 5.5. WatERP extended model.

- hasPhenomenon(?obs, WaterPressure), i.e. the '*WaterPressure*', is an individual of the class '*Phenomenon*' when the object property '*hasPhenomenon*' between the classes '*Observations*' and '*Phenomenon*'.
- hasValue(?res, ?val), i.e. the variable '*res*' for the class '*Result*' has data property '*hasValue*' represented by the variable ?val.
- greaterThan(?val, 7), i.e. if the value of the variable ?val is greater than the number 7.
- HighWaterPressureObservation(?obs), i.e. the head of the rule states that the variable '*obs*' for the class '*Observation*' belongs to the class '*HighWaterPressureObservation*'.
- hasUnit(?res,bar), i.e. the head of contextual constraint states that if '*Phenomenon*' is '*Water Pressure*', then the '*Unit*' should be in 'bar'.

5.2. Step two: Design model

This step involves designing the model, i.e. PIM, which can be extended. We have extended the domain model by incorporating domain-specific entities.

5.2.1. Extended model

Extending the model with elements specific to the domain permits the scripting of these elements' constraints by including their properties in OCL to verify semantics during the semantic validation phase.

Domain Model Extension

The domain model is extended with the elements specific to the water field (Fig. 5.5). In this extension, the *MClass* is inherited by five sub-classes: '*Sensor*', '*Observation*', '*ObservedProperty*', '*Result*', and '*Export*' with their attributes. A sensor has attributes: '*AssignedSensorId*', '*URI*', and '*Observation*'. Note here that all the attributes inside '*Sensor*' and '*Observation*' MClasses are of the type MDProperty or MCProperty instead of the primitive types. This is chosen because, in the '*ModelAnnotation*', the attribute '*annotatedProperty*' is of the type MProperty, which is a superclass of MDProperty/MCProperty. Any other property type will not allow the annotation of these properties. This is one of the aspects of the generic model.

Ontological Annotations

To accommodate the ontological annotations in the WMO, the class '*Expression*' is extended with the attributes of '*comment*', '*label*', and '*alignedWith*'. This will allow to add a technical type of description about the ontological resources in the ontology and to include meta-data as a human-readable ontological concept. It also allows linking the ontology to the external resources to acquire and offer ontological resources to third parties making the ontology both more accessible and interoperable. The '*ExpressionType*' enum is also extended with the '*cardinality Constraint*', '*object Property Rule Constraint*', '*data Property Rule Constraint*', and '*data Value Constraint*' options to create the OCL expressions and generate code according to the constraint type. The class '*Expression*' and the enum '*ExpressionType*' are shown in Fig. 5.6.

5.3. Step three: Semantic annotation

For the semantic annotation, the model is instantiated, and annotations are performed manually by the domain user/expert at level M1. Annotation by association is conducted in the annotation model by applying water knowledge, with the '*Ontology Water*', onto the '*Model Water*'.

5.3.1. Rule annotation

In Eclipse IDE, the model is instantiated (M1) with the relevant data such as an observation with results, pressure phenomenon, pressure value and the unit. The DL-Safe rule 'HighPressureValue' is applied on the OClass '*Observation*' from '*Ontology Water*' to demonstrate the range constraint (5.2) shown in (Fig. 5.7). The annotation of this rule is performed in the '*Model Annotation*' on the MClass '*Observation1*' from '*Model Water*' is shown in Fig. 5.8. Please note, the '*Expression*' is attached with the '*ClassAnnotation*' to define the rule on the ontological class.

In this annotation, the 'type' attribute is of '*dataValueConstraint*', and the '*value*' attribute is the rule in SWRL. The '*comment*' attribute explained the constraint in the form of a mathematical expression. The annotation provides the sustainable semantic understanding of the formal semantics, i.e. rules which are human-readable and understandable from a domain modeller and domain user point of view. The semantic interoperability of the rule is sustainable because with an update in the ontology in terms of adding a new rule or with an update in the values of an '*object property*' or '*individual*' for the existing rule, will update/add the annotation only and rest of the model will remain the same. Similarly, the semantic annotation of the contextual constraint (5.3) is performed, which is shown via an object diagram in (Fig. 5.9).

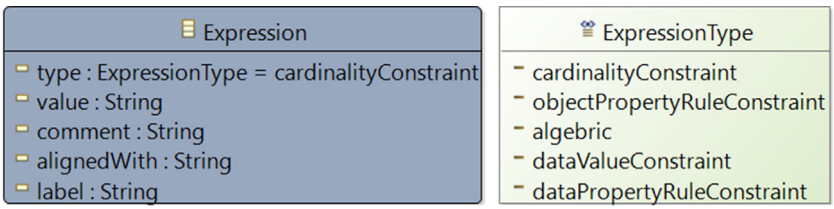

Fig. 5.6. Class expression and enum ExpressionType.

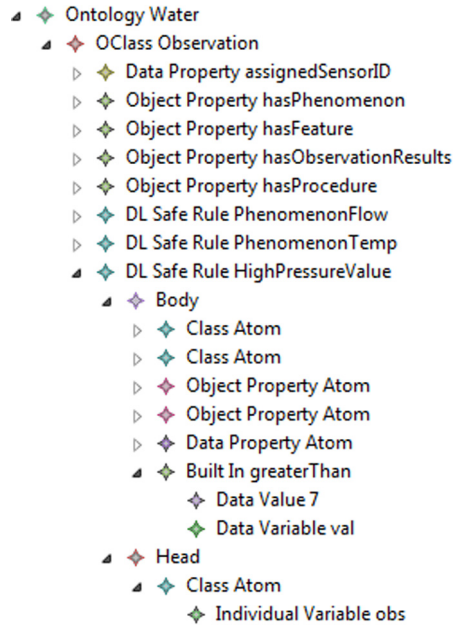

Fig. 5.7. DL-Safe Rule 'HighPressureValue' on OClass observation.

5.4. Semantic validation

This is the last step of the methodology where semantic validation of the water data is performed by checking constraints imposed on these data. Two types of validations are performed. First, the constraint is translated into OCL invariant to validate the data when the model is instantiated at level (M1). An invariant is an OCL expression, which evaluates to true or false. Secondly, to verify the closed-world integrity constraints for ontology-based applications, the code is generated as SPARQL queries. Therefore, the data could be validated by querying the knowledge base. The annotations are also generated from the model for the generated queries for the sustainable semantic interoperability of the rules for the programmers.

5.4.1. Rule validation

The integrity constraints are translated into the OCL invariants to validate the data for model-specific classes, i.e. Sensor and Observation. The OCL invariants shown in (Fig. 5.10) are defined on the MClass '*Observation1*' to validate whether an observation has the phenomenon '*WaterPressure*' and the value of the pressure is greater than '7', in which case the observation will be regarded as '*HighWaterPressureObservation*'. The second invariant checks the unit (c, m3/h/, bar) if they are according to the phenomenon (temperature, flowdischarge, waterpressure) of the observation. The OCL invariants includes the hardcoded values to check and validate the data in the domain model. The EMF allows the execution of these invariants within the model. On success/failure of a constraint, a system notification appears informing if it is violated.

5.4.2. Code generation for contextual constraints

The constraints in the ontology model are automatically transformed into the SPARQL queries by following the transformation rules in Tables 2 and 3. First, to generate a query for a constraint, the annotation model is checked for each class in the model. If the '*MClass*' is annotated, and the type of class '*Expression*' is '*dataValueConstraint*' or '*objectPropertyruleConstraint*', the query is generated for the rule with its name. The object properties of '*ObjectPropertyAtom*' that were used to annotate the model properties in the annotation model are selected to be filled in the generated query, such as '*hasObservationResults*'. The value of the comment in the annotation for the constraint appears as a comment at the top of the query to facilitate the understanding. The code generated in SPARQL for the contextual constraint is shown in Fig. 5.11. The generated queries are executed in Protégé using SPARQL tab. On execution of the query, the result appears in the form of True/False message. The query execution time depends on the number of triple patterns included in it. The time interval increases if there are 'join' operations in the query [62]. At the moment, our queries have four to five triples without a 'join' operation. Therefore, the time to execute the query is less than a second. During the semantic validation of constraints, we did not face any performance or consistency issues and no error in implementation is detected. An excerpt from generate.mtl in the Acceleo module is given in Fig. 5.12.

5.5. Discussion

In this paper, our focus is to demonstrate the transformation of those constraints, which could be exhibited only as SWRL rules, i.e. contextual constraints. We generated the SPARQL queries for contextual and unique constraints. After transformation we had to adjust some of the generated code such as for the range constraint in (5.2) the '?val $>$ 7' was part of the body of the rule but we put it in the head of the generated query in SPARQL. For the transformation of unique constraints, the generated code is adjusted by using '*Union*' and '*&&*' operators. Through our methodology, the translation to SPARQL is complete and sound within the expressivity of the SWRL. For example, SWRL is monotonic and does not support missing values, negative expressions, modification, mathematical functions, and priority relationships, which are frequently used when modelling real situations [63]. SWRL will need to be extended to handle such constraints [64]. The SPARQL queries can be produced with NAF operator "not" with our approach, as it was performed in [8–10]. The approach presented by [9] transformed the range, cardinality, and datatype constraints, whereas our demonstration included contextual, pattern constraints in addition to the above constraints. Our method outperforms these approaches by providing a straightforward transformation of SWRL to SPARQL with predefined modelling elements for composing rules. The rules can be used by domain experts and non-technical users for managing formal systems and complex formalisms. In this work, we have shown the transformation of contextual and range constraints in SPARQL; however, multiple artefacts could be generated from a single model. For example, to generate code in other languages such as VB, C# and

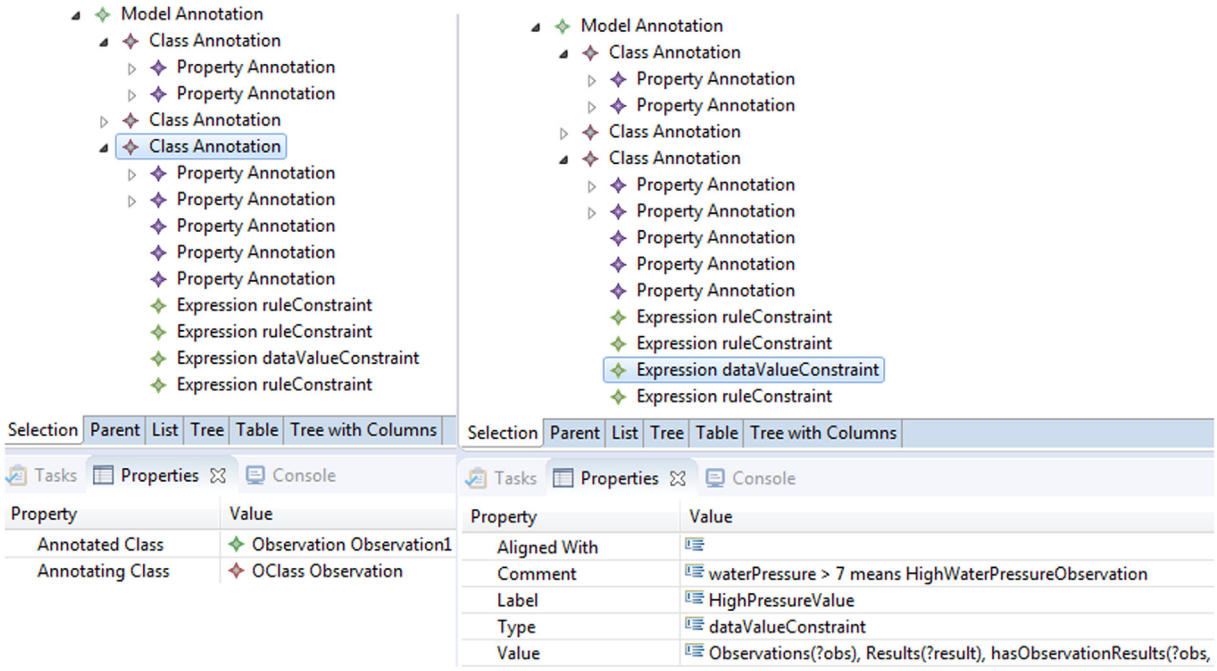

Fig. 5.8. Annotation of rule (dataValueConstraint) on class observation.

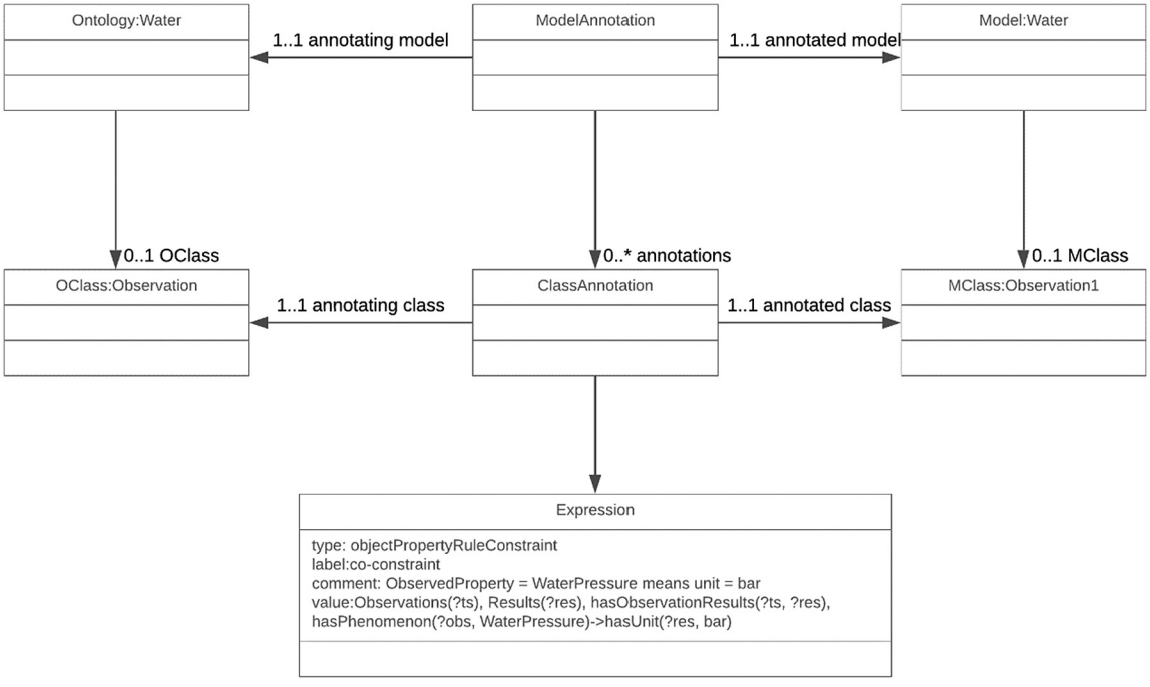

Fig. 5.9. Annotation of rule (contextual constraint) on class observation.

```
invariant HighWPressure:
self.obprop.oclAsType(MCProperty).classType.oclAsType(ObservedProperty).prop.value.toString().toLower() = 'waterpressure' and
self.result.oclAsType(MCProperty).classType.oclAsType(Result).dataproperty.value.toInteger()->at(1) > 7 implies
self.obprop.oclAsType(MCProperty).classType = 'HighWaterPressureObservation';

invariant coconstObsProp:
let correctObsProp : Boolean = self.obprop.oclAsType(MCProperty).classType.oclAsType(ObservedProperty).prop.value.toString().toLower() = 'temperature'
and self.result.oclAsType(MCProperty).classType.oclAsType(Result).unit.value.toString().toLower() = 'c' or
self.obprop.oclAsType(MCProperty).classType.oclAsType(ObservedProperty).prop.value.toString().toLower() = 'flowdischarge'
and self.result.oclAsType(MCProperty).classType.oclAsType(Result).unit.value.toString().toLower() = 'm3/h' or
self.obprop.oclAsType(MCProperty).classType.oclAsType(ObservedProperty).prop.value.toString().toLower() = 'waterpressure'
and self.result.oclAsType(MCProperty).classType.oclAsType(Result).unit.value.toString().toLower() = 'bar' in
 if self.oclIsInvalid() then correctObsProp = false else correctObsProp = true endif;
```

Fig. 5.10. OCL invariants for DataValue (range) and contextual constraints.

```
#comment waterPressure > 7 means HighWaterPressureObservation

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX swka: <http://www.semanticweb.org/acorchero/ontologies/2013/0/ambreenSWKA.owl#>
ASK where {

?obs rdf:type swka:Observations .
?res rdf:type swka:Results .
?obs swka:hasObservationResults ?res .
?obs swka:hasPhenomenon swka:WaterPressure .
?res swka:hasValue ?val .
FILTER EXISTS {?obs rdf:type swka:HighWaterPressureObservation. Filter ((?val > 7) ) }
}
#comment ObservedProperty = waterpressure means unit = bar

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX swka: <http://www.semanticweb.org/acorchero/ontologies/2013/0/ambreenSWKA.owl#>
ASK where {

?obs rdf:type swka:Observations .
?res rdf:type swka:Results .
?obs swka:hasObservationResults ?res .
?obs swka:hasPhenomenon swka:WaterPressure .
FILTER EXISTS {?res swka:hasUnit swka:bar. }

}
```

Fig. 5.11. Code generated in SPARQL for the contextual constraint.

```
[template public generateValidatorForSPARQL(aTopNode : TopNode) {ModelName :String = aTopNode.Model.name.toUpperFirst();}]

[comment @main /]
    [file (ModelName.concat('Constraints.txt'), false, 'UTF-8')]

[for (m : MClass | aTopNode.Model.MClass)]
[for (ca: ClassAnnotation | aTopNode.Annotation.annotations)]
[if (m.name = ca.annotatedClass.name)]
[for (ex: Expression | ca.expression)]
[for (extype: ExpressionType| ex.type)]
[if (extype.toString() = 'dataValueConstraint')]
[ex.comment.concat('\n').prefix('#comment ')/]
    [for (ont : OClass | aTopNode.Ontology.OClass)]
        [if (ca.annotatingClass.name = ont.name)]
            [for (dr : DLSafeRule | ont.constraints.oclAsType(DLSafeRule))]
                [if (dr.name = ex.label)]
                #Query name = [dr.name/]
                PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                PREFIX owl: <http://www.w3.org/2002/07/owl#>
                PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
                PREFIX swka: <http://www.semanticweb.org/acorchero/ontologies/2013/0/WatERPOntologySWKA.owl#>
                ASK WHERE {
                [RuleBodyDataValue(dr._body,ont, m, ca)/]
                [RuleHeadDataValue(dr.head,ont, m, ca,dr._body)/]

                }
                [/if]
            [/for]
        [/if]
    [/for]
[elseif (extype.toString() = 'objectPropertyRuleConstraint')]
[ex.comment.concat('\n').prefix('#comment ')/]
[for (ont : OClass | aTopNode.Ontology.OClass)]
        [if (ca.annotatingClass.name = ont.name)]
            [for (dr : DLSafeRule | ont.constraints.oclAsType(DLSafeRule))]
                [if (dr.name = ex.label)]
                #Query name = [dr.name/]
                PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                PREFIX owl: <http://www.w3.org/2002/07/owl#>
                PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
                PREFIX swka: <http://www.semanticweb.org/acorchero/ontologies/2013/0/WatERPOntologySWKA.owl#>
                ASK WHERE {
                    [RuleBodyDataValue(dr._body,ont, m, ca)/]
                    [RuleHeadDataValue(dr.head,ont, m, ca,dr._body)/]
                    }
```

Fig. 5.12. Generate.mtl in the Acceleo module.

Java, the code in the template (generate.mtl) could be rearranged and replaced. For the WatERP project, we generated code in Java and SPARQL. The transformation rules would help the programmers to understand the template and code replacement. This way, indirect semantic interoperability between the programmers is brought because they share the understanding of abstract knowledge, i.e. model-driven paradigm. On this basis, they can place the code in the programming language of their choice and replace code stubs in the template. For the WatERP case study, the transformation of the extended ontological annotations can be used

for ontology linkage to resources externally to acquire and offer ontological resources to the third parties making ontology both more accessible and semantically interoperable. We have also applied the methodology for the semantic validation of IoT-based healthcare data [65].

6. Conclusion

Semantic validation is crucial for the consistency and integrity of data. This paper proposed a formal and flexible MDE-based methodology to enable open-world assumption based applications to check closed-world integrity constraints for semantic validation of the data. The methodology demonstrated the use of MDE to model integrity constraints as SWRL rules. The rules are expressed using ontology model concepts and are added as semantic annotations on domain model elements. We showed the semantic validation of data by expressing integrity constraints as OCL expressions at the model level. We also showed the automatic transformation of SWRL rules into SPARQL queries for closed-world constraint checking, which is the primary focus of the paper. The SPARQL compatible existing OWL reasoners can evaluate these queries for semantic validation of data. The model is generic and can be extended with additional ontological concepts and domain-specific entities if required. We can apply this methodology to any domain for the semantic validation of data. For the evaluation of our approach, we implemented it on a water case study.

The code can be generated for more than one platform by following the transformation rules and replacing the code stubs. However, the modellers and domain experts will need to have a basic understanding of SWRL, OWL, modelling and transformation techniques. Moreover, further empirical evaluation is required, e.g., in the case of large rules containing hundreds of atoms, advanced techniques such as groupings will need to be incorporated. We include the automated population of ontology/domain models and semantic annotations of the domain model with ontology model in future work. This will allow us to perform quantitative evaluation in terms of number of data violations and time to perform the data validation.

CRediT authorship contribution statement

Ambreen Hussain: Conceptualisation, Methodology, Investigation, Validation, Writing – original draft. **Wenyan Wu:** Supervision, Project administration, Funding acquisition, Review. **Zhaozhao Tang:** WatERP co-worker, Review.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors acknowledge the financial support of the European Commission's Seventh Framework Program (FP7) under the grants WatERP (318603) and FP7 Marie Curie Actions SmartWater, United Kingdom (PIRSESGA-2012-318985). We also acknowledge Staffordshire University, United Kingdom to give us an opportunity and facilities to conduct and disseminate the WatERP project successfully.

References

[1] D.C. Schmidt, Model-driven engineering, Computer (Long. Beach. Calif), 2006, http://dx.doi.org/10.1109/MC.2006.58.
[2] A. Rutle, A. Rossini, Y. Lamo, U. Wolter, A formal approach to the specification and transformation of constraints in MDE, J. Log. Algebr. Program. (2012) http://dx.doi.org/10.1016/j.jlap.2012.03.006.
[3] I. Horrocks, P.F. Patel-schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A semantic web rule language combining OWL and RuleML, W3C Memb. Submiss. 21 (2004) 1–20.
[4] I. Horrocks, P.F. Patel-Schneider, S. Bechhofer, D. Tsarkov, OWL rules: A proposal and prototype implementation, Web Semant. (2005) http://dx.doi.org/10.1016/j.websem.2005.05.003.
[5] P. Hitzler, B. Parsia, Ontologies and rules, in: Handb. Ontol, 2009, http://dx.doi.org/10.1007/978-3-540-92673-3_5.
[6] B. Motik, I. Horrocks, U. Sattler, Bridging the gap between OWL and relational databases, Web Semant. (2009) http://dx.doi.org/10.1016/j.websem.2009.02.001.
[7] P.F. Patel-Schneider, Using description logics for RDF constraint checking and closed-world recognition, in: Proc. Natl. Conf. Artif. Intell, 2015.
[8] J. Tao, E. Sirin, J. Bao, D.L. McGuinness, Integrity constraints in OWL, in: Proc. Natl. Conf. Artif. Intell, 2010.
[9] Y. Shu, A practical approach to modelling and validating integrity constraints in the semantic web, Knowl.-Based Syst. (2018) http://dx.doi.org/10.1016/j.knosys.2018.04.021.
[10] Y. Shu, Q. Liu, K. Taylor, Semantic validation of environmental observations data, Environ. Model. Softw. 79 (2016) 10–21, http://dx.doi.org/10.1016/j.envsoft.2016.01.004.
[11] C. Laing, P. David, E. Blanco, X. Dorel, Questioning integration of verification in model-based systems engineering: an industrial perspective ⋆, Comput. Ind. 114 (2020) 103163, http://dx.doi.org/10.1016/j.compind.2019.103163.
[12] K. Hacid, Y. Ait-Ameur, Annotation of engineering models by references to domain ontologies, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2016, pp. 234–244, http://dx.doi.org/10.1007/978-3-319-45547-1_19.
[13] K. Hacid, Y. Ait-Ameur, Strengthening MDE and formal design models by references to domain ontologies. a model annotation based approach, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2016, pp. 340–357, http://dx.doi.org/10.1007/978-3-319-47166-2_24.
[14] C. Vitolo, Y. Elkhatib, D. Reusser, C.J.A. Macleod, W. Buytaert, Web technologies for environmental Big Data, Environ. Model. Softw. 63 (2015) 185–198, http://dx.doi.org/10.1016/j.envsoft.2014.10.007.
[15] M.A. Regueiro, J.R.R. Viqueira, J.A. Taboada, J.M. Cotos, Virtual integration of sensor observation data, Comput. Geosci. (2015) http://dx.doi.org/10.1016/j.cageo.2015.04.006.
[16] J.S. Horsburgh, D.G. Tarboton, M. Piasecki, D.R. Maidment, I. Zaslavsky, D. Valentine, T. Whitenack, An integrated system for publishing environmental observations data, Environ. Model. Softw. 24 (2009) 879–888, http://dx.doi.org/10.1016/j.envsoft.2009.01.002.
[17] A.P. Sheth, J.A. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases, ACM Comput. Surv. 22 (1990) 183–236, http://dx.doi.org/10.1145/96602.96604.
[18] B. Motik, P.F. Patel-Schneider, B.C. Grau, I. Horrocks, B. Parsia, U. Sattler, OWL 2 web ontology language direct semantics, Direct (2009).
[19] M. Krötzsch, S. Rudolph, P. Hitzler, Description logic rules, in: Front. Artif. Intell. Appl, 2008, http://dx.doi.org/10.3233/978-1-58603-891-5-80.
[20] D. Fiorello, N. Gessa, P. Marinelli, F. Vitali, {DTD}++ 2.0: Adding support for co-constraints, in: Proc. Extrem. Markup Lang. 2004, 2004.
[21] J.D. Poole, Model-driven architecture: Vision, standards and emerging technologies, work, Metamodeling Adapt. Object Model. ECOOP01 (2001).
[22] Omg, UML infrastructure specification, v2.4.1, 2011, p. 34, http://dx.doi.org/10.1007/s002870050092, Omg.
[23] Object Management Group (OMG), OMG meta object facility (MOF) core specification, 2016, OMG Doc. Number Formal/2016-11-01.
[24] OMG-OCL, Object constraint language, Int. Bus. (2014) 58–90, http://dx.doi.org/10.1167/7.9.852.
[25] D. Djurić, D. Gašević, V. Devedžić, The tao of modeling spaces, J. Object Technol. (2006) http://dx.doi.org/10.5381/jot.2006.5.8.a4.
[26] J. Bézivin, Model driven engineerings: An emerging technical space, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2006, http://dx.doi.org/10.1007/11877028_2.
[27] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, J.G. Timothy, Eclipse modeling framework: A developer's guide, 2003, http://dx.doi.org/10.1108/02641610810878585.
[28] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, M. Paternostro, EMF: eclipse modeling framework, 2008, http://dx.doi.org/10.1108/02641610810878585.
[29] C. Damus, A. Sánchez-Barbudo Herrera, A. Uhl, E. Willink, OCL documentation, 2016, http://download.eclipse.org/ocl/doc/6.1.0/ocl.pdf (accessed June 29, 2018).

[30] F.M. Donini, D. Nardi, R. Rosati, Description logics of minimal knowledge and negation as failure, ACM Trans. Comput. Log. (2002) http://dx.doi.org/10.1145/505372.505373.
[31] 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: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2016, http://dx.doi.org/10.1007/978-3-319-46547-0_30.
[32] M. Fang, Georgia state university, 2013.
[33] G. Lausen, M. Meier, M. Schmidt, SPARQLing constraints for RDF, 2008, http://dx.doi.org/10.1145/1352431.1352492.
[34] I. Boneva, J.E. Labra Gayo, E.G. Prud'hommeaux, Semantics and validation of shapes schemas for RDF, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2017, http://dx.doi.org/10.1007/978-3-319-68288-4_7.
[35] M.A. Atoui, A. Cohen, Coupling data-driven and model-based methods to improve fault diagnosis, Comput. Ind. (2021) 103401, http://dx.doi.org/10.1016/j.compind.2021.103401.
[36] D. Rajpathak, Y. Xu, I. Gibbs, An integrated framework for automatic ontology learning from unstructured repair text data for effective fault detection and isolation in automotive domain, Comput. Ind. 123 (2020) 103338, http://dx.doi.org/10.1016/j.compind.2020.103338.
[37] F.S. Parreiras, S. Staab, Using ontologies with UML class-based modeling: The TwoUse approach, Data Knowl. Eng. (2010) http://dx.doi.org/10.1016/j.datak.2010.07.009.
[38] M. Schneider, SPARQLAS - implementing SPARQL queries with OWL syntax, in: CEUR Workshop Proc, 2010.
[39] a. Rossini, K.a. Mughal, U. Wolter, a. Rutle, Y. Lamo, A formal approach to data validation constraints in MDE, in: 5th Int. Harnessing Theor. Tool Support Softw., 2011, pp. 65–76.
[40] A. Hafeez, A.-. Rehman, Ontology based verification of UML class/OCL model, Mehran Univ. Res. J. Eng. Technol. 37 (2018) 521–534, http://dx.doi.org/10.22581/muet1982.1804.07.
[41] S. Lukichev, Defining a subset of OCL for expressing SWRL rules, in: CEUR Workshop Proc, 2008.
[42] M. Milanović, D. Gašević, A. Giurca, G. Wagner, V. Devedžić, Towards sharing rules between OWL/SWRL and UML/OCL, in: Electron. Commun, EASST, 2006, http://dx.doi.org/10.14279/tuj.eceasst.5.38.69.
[43] D. Kalibatiene, O. Vasilecas, On OWL/SWRL mapping to UML/OCL, in: ACM Int. Conf. Proceeding Ser, 2010, http://dx.doi.org/10.1145/1839379.1839391.
[44] F. Song, G. Zacharewicz, D. Chen, An ontology-driven framework towards building enterprise semantic information layer, Adv. Eng. Inf. (2013) http://dx.doi.org/10.1016/j.aei.2012.11.003.
[45] E. Rahm, The case for holistic data integration, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2016, http://dx.doi.org/10.1007/978-3-319-44039-2_2.
[46] C.L. LI, C. McMahon, L. Newnes, Supporting multiple engineering viewpoints in computer-aided design using, in: ICED13 19th Int. Conf. Eng. Des, 2013.
[47] Y. Lin, Semantic annotation for process models: Facilitating Process Knowledge Management via Semantic Interoperability, Fakultet for informasjonsteknologi, matematikk og elektroteknikk, 2008.
[48] N. Zouggar, B. Vallespir, D. Chen, Semantic enrichment of enterprise models by ontologies-based semantic annotations, in: Proc. - IEEE Int. Enterp. Distrib. Object Comput. Work. EDOC. 2008, pp. 216–223,.
[49] Y. Wang, H. Li, Adding semantic annotation to UML class diagram, in: ICCASM 2010-2010 Int. Conf. Comput. Appl. Syst. Model. Proc, Vol. 9, 2010, pp. 187–190, http://dx.doi.org/10.1109/ICCASM.2010.5623055.
[50] Y. Liao, M. Lezoche, H. Panetto, N. Boudjlida, E. Rocha Loures, Formal semantic annotations for models interoperability in a PLM environment, in: IFAC Proc, 2014, pp. 2382–2393, http://dx.doi.org/10.3182/20140824-6-ZA-1003.02551.
[51] S. Staab, R. Studer, Handbook on ontologies - second edition, Int. Handb. Inf. Syst. (2009) 811.
[52] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe, S. Jupp, G. Moulton, R. Stevens, N. Drummond, S. Jupp, G. Moulton, S. Brandt, A practical guide to building OWL ontologies using protege 4 and CO-ODE tools, Matrix (2011) 0–107.
[53] V. Ghorbanian, B.W. Karney, Y. Guo, Minimum pressure criterion in water distribution systems: Challenges and consequences, 2015, http://dx.doi.org/10.1061/9780784479162.072.
[54] A. Lambert, What do we know about pressure: Leakage relationships in distribution systems? in: IWA Conf. Syst. Approach To Leakage Control Water Distrib. Syst, 2000.
[55] A. Hussain, W. Wu, G. Anzaldi, A. Abecker, Implementation of OGC compliant framework for data integration in Water Distribution System, in: Procedia Eng, 2015, pp. 1366–1374, http://dx.doi.org/10.1016/j.proeng.2015.08.984.
[56] A. Hussain, W. Wu, G. Anzaldi, A. Abecker, A generic framework to integrate water supply distribution systems using interoperable standards, 2015, p. 10, https://www.academia.edu/14360335/A_GENERIC_FRAMEWORK_TO_INTEGRATE_WATER_SUPPLY_DISTRIBUTION_SYSTEMS_USING_INTEROPERABLE_STANDARDS (accessed December 15, 2017).
[57] A. Corchero, G. Anzaldi, C. Chomat, WatERP water enhanced resource planning & quot;where water supply meets demand & quot; GA number: 318603 WP1: Water supply knowledge base 1.2: Generic functional model for water supply and usage data, 2013, www.waterp-fp7.eu (accessed July 21, 2018).
[58] G. Anzaldi, Waterp water enhanced resource planning & quot;where water supply meets demand & quot; GA number: 318603 WP 1: Water supply knowledge base 1.3: Generic ontology for water supply distribution chain, 2013, www.waterp-fp7.eu (accessed July 21, 2018).
[59] G. Anzaldi, E. Rubion, A. Corchero, R. Sanfeliu, X. Domingo, J. Pijuan, F. Tersa, Towards an enhanced knowledge-based Decision Support System (DSS) for integrated water resource management (IWRM), Procedia Eng. 89 (2014) 1097–1104, http://dx.doi.org/10.1016/j.proeng.2014.11.230.
[60] A. Corchero, G. Anzaldi, R. Garcia, Towards a semantic and holistic architecture for water sensor data integration, IAHR Conf. 2015 (2015) 1–11.
[61] B. Glimm, M. Horridge, B. Parsia, P.F. Patel-Schneider, A syntax for rules in OWL 2, in: CEUR Workshop Proc, 2009.
[62] K.M. Kyu, A.N. Oo, Enhancement of query execution time in SPARQL query processing, in: Proc. 4th Int. Conf. Adv. Inf. Technol. ICAIT 2020, 2020, http://dx.doi.org/10.1109/ICAIT51105.2020.9261805.
[63] J.M. Alcaraz Calero, A.M. Ortega, G.M. Perez, J.A. Botia Blaya, A.F. Gomez Skarmeta, A non-monotonic expressiveness extension on the semantic web rule language, J. Web Eng. (2012).
[64] H.J. Kim, W. Kim, M. Lee, Semantic Web Constraint Language and its application to an intelligent shopping agent, Decis. Support Syst. (2009) http://dx.doi.org/10.1016/j.dss.2008.12.004.
[65] A. Hussain, W. Wu, Sustainable interoperability and data integration for the IoT-based information systems, in: Proc. - 2017 IEEE Int. Conf. Internet Things, IEEE Green Comput. Commun. IEEE Cyber, Phys. Soc. Comput. IEEE Smart Data, IThings-GreenCom-CPSCom-SmartData 2017, 2018, http://dx.doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2017.126.