Chapter 7
Metamodelling and Ontologies(*)

Gerd Gröner, Nophadol Jekjantuk, Tobias Walter, Fernando Silva Parreiras, and Jeff Z. Pan

Abstract Metamodelling plays an interesting role in MDSD. Following the discussions on metamodelling in Sect. 2.2.2, this chapter presents technologies on integrating metamodelling and ontologies. On the one hand, there are metamodels for standard ontology languages, such as OWL 2. On the other hand, there have been efforts enriching standard ontology languages with metamodelling capabilities. Chapter 11 will provide further discussions on metamodelling on how it can be useful for ontology-integrated modelling.

This chapter is organised as follows. In the first part (Sect. 7.1), we present the three most dominant state of the art (linguistic) metamodels for OWL, namely the *OMG ODM* [150], the *Neon OWL Metamodel* [34] and the *OWL 2 Metamodel* [140]. In Section 7.2, We present OWL FA [157], a metamodelling extension of OWL. Section 7.3 discusses an important aspect of metamodelling, i.e. open world and closed world assumption. The discussions on OWL FA and the open world/closed world assumption are related to earlier discussions on ontology reasoning (cf. Chaps. 3 and 5).

7.1 Metamodelling for Ontologies

The notion of metamodelling is well established and commonly used in the realm of model-driven engineering. There are standards which describe the model-driven architecture [149] and provide language and modelling specifications as UML [151]. However, *metamodelling* is only weakly considered in the realm of ontologies and ontology engineering.

This might be a weakness in two dimensions. Firstly, from an ontology engineering point of view, the modelling and design principles of model-driven engineering,

J.Z. Pan et al. (eds.), *Ontology-Driven Software Development*,
DOI 10.1007/978-3-642-31226-7_7,

including metamodelling, benefit from abstraction of a concrete system. These principles are only weakly exploited in ontology engineering. Secondly, if we consider metamodelling more generally as modelling with metadata, this becomes a relevant issue in various domains, and this has been considered as a challenging issue in OWL (cf. [138,157]). Moreover, metamodelling is already supported by the OWL 2 language [160]).

A key issue of metamodelling is that there is no clear-cut distinction between classes and individuals. Let us take the knowledge base about animals as an example [216] "Ted is an Eagle" and "Eagles are an endangered species". The first statement can be represented in OWL by asserting the individual Ted to be an instance of the class Eagle. The second statement cannot be modelled in OWL, because it does not allow one symbol to refer both, to a class as well as to an individual. One may model the class Eagle as a subclass of the class EndangeredSpecies. However, an Eagle is a species and not a set of all living eagles. Thus, this could lead to consequence that is doubtful (Ted is an endangered species) or out-right unwanted. Therefore, it should be modelled by stating the individual Eagle to be an instance of the class EndangeredSpecies. Hence, the symbol Eagle should be used to refer both to a class as well as to an individual. This style of modelling is often called metamodelling.

The rest of this section presents a short description of the three most prominent states of the art OWL metamodels, namely the *OMG ODM* [150], the *Neon OWL Metamodel* [34] and the *OWL 2 Metamodel* [140]. The contribution of this section is the comparison of these metamodels.

7.1.1 Ontology Definition Metamodel

The *OMG OWL metamodel* is part of the *ontology definition metamodel (ODM)* OMG Adopted Specification. The ODM [150] includes six metamodels that are grouped together according to their representation formalism like (1) logic formalism (DL and FOL), (2) structural or descriptive representation and (3) conceptual or object-oriented modelling style. The OMG *OWL metamodel* and the OMG *RDF metamodel* belong to the group of structural or descriptive representations. ODM provides relationships between the RDF and OWL packages.

ODM offers a high number of classes due to the import of the *OMG RDF metamodel*. Accordingly, some relations between classes are described in the RDFS metamodel and are reused in the OWL metamodel. It was one of the motivation of ODM, to provide metamodels that capture W3C standardised languages, including OWL, RDF and RDFS, but also ISO languages like Topic Maps.

The justification for defining a new metamodel for ontologies instead of extending the UML metamodel is based on the origin of OWL (especially OWL DL). OWL as well as UML class diagrams are based on set semantics,

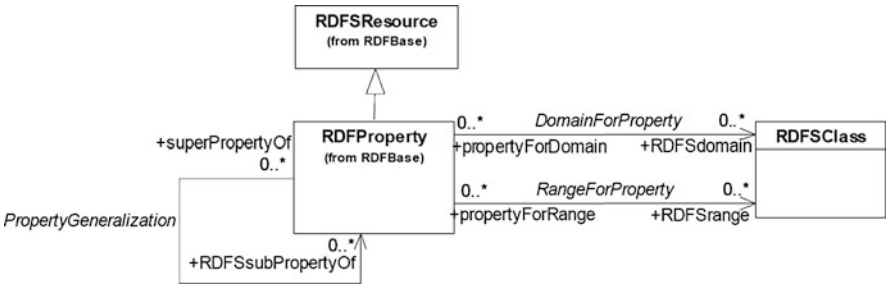

Fig. 7.1 The properties diagram of the RDFS package [150]

The *RDF metamodel* is composed of three metamodels packages: (1) The RDFBase package defines the core concepts of RDF that are specified in the abstract syntax of RDF, like Node, RDFSResource and URIReference. (2) The RDFS package extends the RDFBase package with additional RDFS language constructs. (3) Finally, the RDFWeb package is also an extension of the RDFBase package, but these constructs are not available in the RDFS package.

The RDFS packages provide further definitions like the package for classes that defines, for instance, sub- and superclass relations between classes as well as the package on properties that are defined as relations between pairs of resources. Moreover, it contains specifications on domain and range of properties and sub- and super-property relations. The property package is depicted in Fig. 7.1.

The *OWL metamodel* extends the RDF metamodel. The OWL metamodel consists of a base package that describes common parts for OWL DL and OWL Full and two sub-packages that are specific for the OWL DL and OWL Full dialects. (OWL Lite is covered by the base package and parts of the OWL DL package.) The base package provides various descriptions like for OWL classes and properties. The Class Description Diagram of the base package is depicted in Fig. 7.2.

There are three class constructors, representing intersection, union and complement of class(es). They are adopted from description logics. Moreover, a class can be defined as an enumeration of individuals (EnumeratedClass). OWLRestriction allows the definition of a class as a restriction of a particular property (datatype and object properties) in combination with a value or cardinality.

Another package of the OWL metamodel is the property package, which is displayed in Fig. 7.3. It is a refinement of the RDF property. Compared to the RDF property, there are further means to specify property, like in case of an object property, we can define subclasses like InverseFunctionalProperty, SymmetricProperty or TransitiveProperty. Moreover, properties can be defined as inverse of other properties.

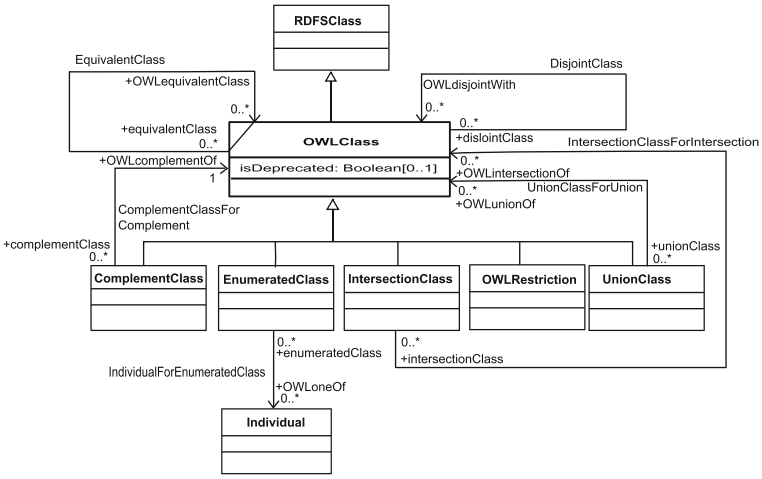

Fig. 7.2 The OWL class descriptions diagram of the OMG OWL metamodel[150]

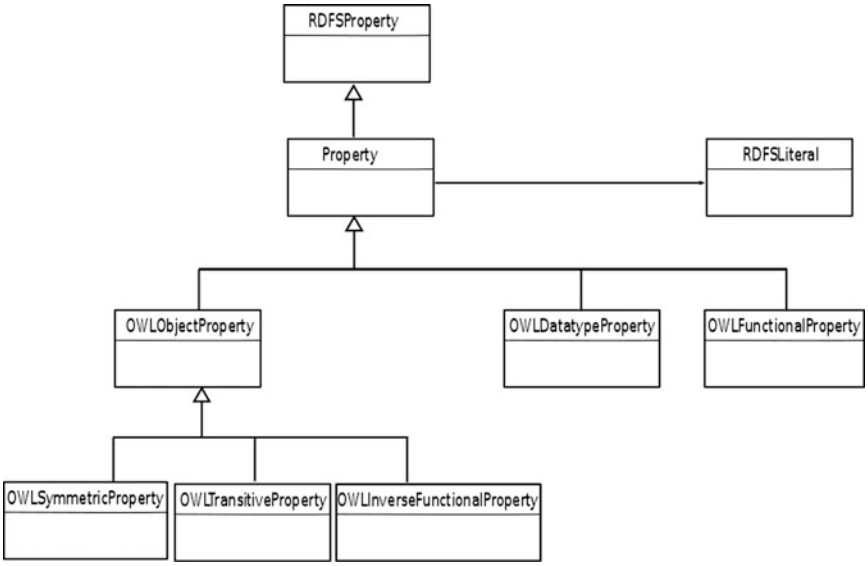

Fig. 7.3 The OWL properties diagram of the OMG OWL metamodel [150]

7.1.2 NeOn OWL Metamodel

The *NeOn Metamodel* [33, 34] is a concise metamodel which is able to cover the OWL DL functional syntax. The NeOn OWL Metamodel is based on MOF.

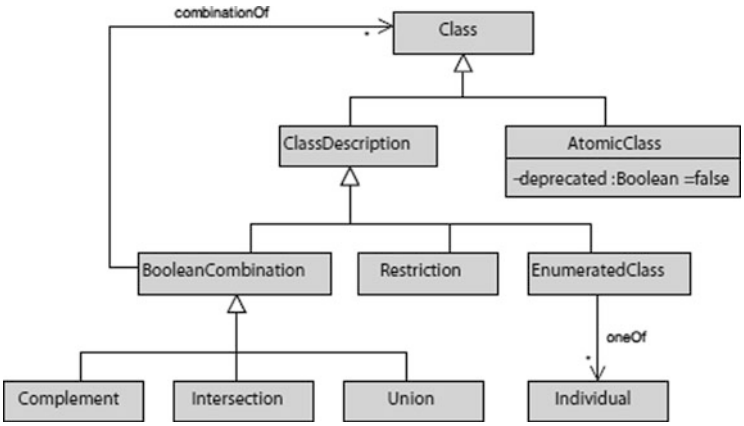

Fig. 7.4 The OWL class descriptions diagram of the NeOn metamodel

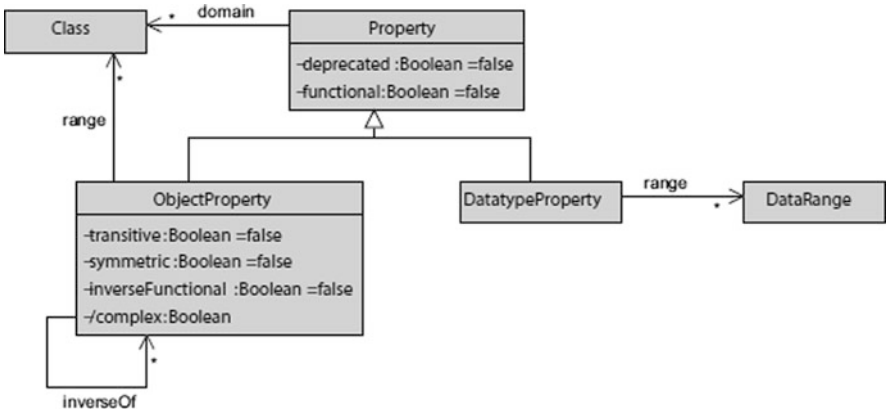

Fig. 7.5 The OWL properties diagram of the NeOn metamodel

The motivation of this work was the ability to use UML-like notations and tools in ontology engineering. Compared to the ODM, the NeOn OWL Metamodel supports the specification of networked ontologies, where networked ontologies are distributed ontologies that are connected and related to each other.

Figure 7.4 and 7.5 depict the OWL class hierarchy and the property diagram of NeOn, respectively [33]. The relationship between Class and Property is direct, since the NeOn OWL Metamodel does not provide support for RDFS like ODM.

According to the intention of the NeOn OWL Metamodel as a metamodel for networked ontologies, it offers further modelling constructs like Mapping classes that allow the definition of mappings between two ontologies.

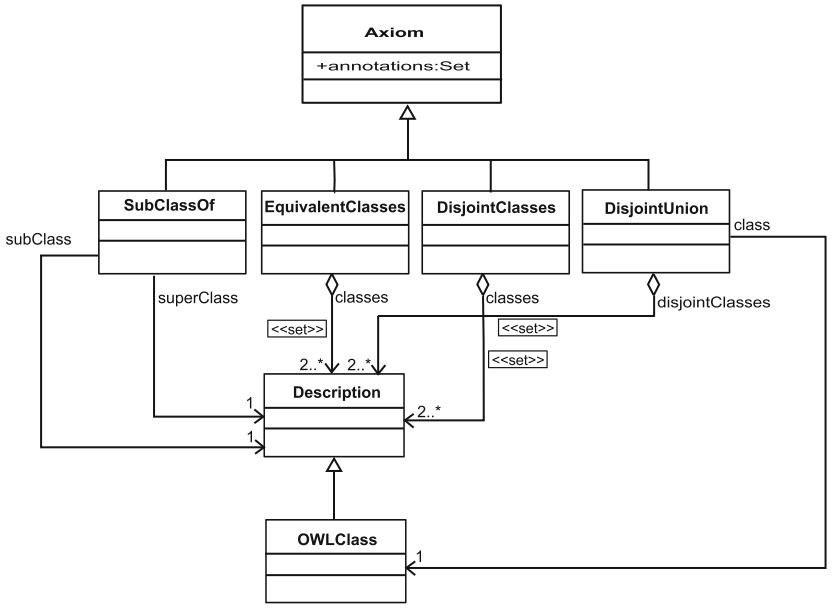

Fig. 7.6 The OWL class descriptions diagram of the OWL 2 metamodel

7.1.3 OWL 2 Metamodel

Improvements on the OWL language led the W3C OWL Working Group to publish working drafts of a new version of OWL: *OWL 2* [140]. OWL 2 is fully compatible with OWL DL and extends the latter with limited complex role inclusion axioms, reflexivity and irreflexivity, role disjointness and qualified cardinality restrictions. Moreover, OWL 2 uses a new XML serialisation and provides a set of profiles with different levels of expressiveness.

As one may note, the OWL 2 metamodel is considerably different from the available metamodels for OWL. Constructs like axiom and OWLEntity play central roles, and associations between classes and properties are done by axioms. There are different kinds of axioms provided like class or description axioms for describing class relations like equivalence and disjointness, object property axioms, data property axioms and axioms for facts. Facts express statements on individuals like same and different individual expressions and class assertions.

Figure 7.6 depicts the OWL class description diagram and the usage of axioms to describe class expressions like expressing the equivalence of classes.

In Fig. 7.7 the OWL properties diagram is depicted. It exemplifies the usage of the construct OWLEntity as a central building block.

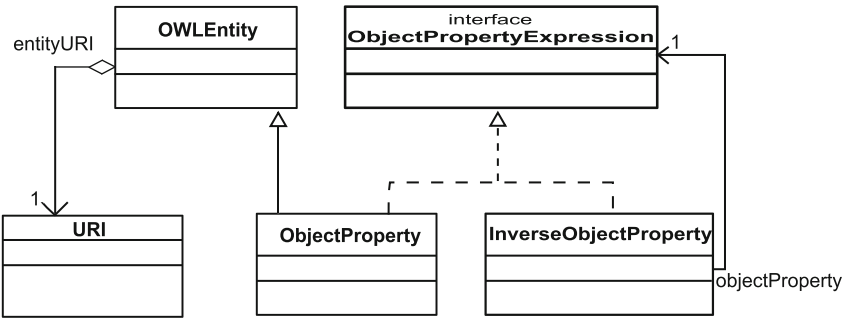

Fig. 7.7 The OWL properties diagram of the OWL 2 metamodel

7.2 Ontologies for Metamodelling: OWL FA

We already motivated metamodelling in the beginning of this chapter. We come back to this issue in this section and provide deeper investigations in metamodelling in OWL. Besides a motivating illustration of metamodelling in OWL, we present OWL FA[157], a metamodelling enabled ontology language. OWL FA is an extension of OWL DL, which has the description logics expressivity $\mathcal{SHOIN}(\mathcal{D})$. Ontologies in OWL FA are represented in a layered architecture. This architecture is mainly based on the architecture of RDFS(FA) [155].

7.2.1 Motivating Example

The running example in this section aims at clarifying the problem that is tackled by OWL FA and is used throughout this section for demonstrating how the example model is represented in OWL FA. Models are depicted in UML-like notations [153]. Metamodels are more than a syntactic language description of a modelling language; a metamodel is a description of the concepts of a modelling language specifying the structure and the kind of information that can be handled [144]. Models and metamodels are commonly used in model-driven software engineering (MDSE). In order to improve software development processes, new technologies, which provide reasoning support like consistency checking of models and metamodels, are beneficial. In MDSE, each model layer can contain both class and object definitions (cf. [8]). In ontology engineering, ontologies for metamodelling like OWL FA separate classes and objects into different layers in order to maintain the decidability of the language. In the rest we use ontologies for metamodelling to improve software development processes by validating, consistency checking and classifications of models with respect to their metamodels.

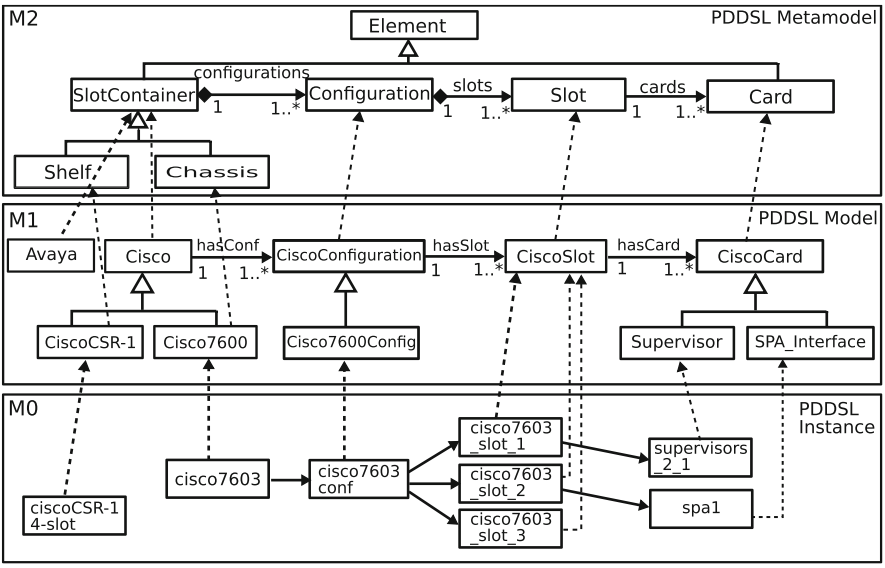

Fig. 7.8 Layered architecture for a physical device model

In order to use these benefits, a transformation of software models and metamodels, which are mainly graphical models to OWL DL ontologies, is not sufficient, since OWL DL does not support reasoning in a layered modelling architecture. This may introduce incomplete meaning of the model or it does not provide a valid semantic base at all. OWL FA provides more expressive power for metamodelling, and the semantics of OWL FA are well defined.

Example 1. In Fig. 7.8, a layered modelling architecture is demonstrated for physical device modelling (an application of configuration management). A physical device domain-specific language (PDDSL) is a domain-specific language (DSL) for physical devices which is used in business IT system modelling. The figure depicts three layers M_0, M_1 and M_2. M_3 is a meta-metamodelling layer, which is not included in the example. The arrows between the layers demonstrate instance relationships; the arrows within a layer are relations between concepts (like object properties and subclass hierarchies).

Cisco, CiscoConfiguarion, CiscoSlot and CiscoCard are instances of some classes from the M_2 layer. At the same time, these artefacts are concepts in the layer M_1 and each of these concepts can contain some instances from layer M_0. The rest of the relationships in a layer like concept subsumption and object properties are represented by arrows.

There are additional modelling constraints imposed on different layers. In layer M_2, a model designer requires that each SlotContainer has a Configuration, expressing that the ObjectProperty configurations. Each Configuration may contain Slot, which is expressed by the ObjectProperty slots as a functional

property. And each Slot may contain Card, this is also expressed as an ObjectProperty cards. Moreover, the modeller requires the disjointness of the two classesChassis and Shelf. There are also three ObjectProperty assertions from the layer M_2 to M_1: configuration(Cisco, CiscoConfiguration), slots(CiscoConfiguration, CiscoSlot) and cards(CiscoSlot, CiscoCard).

In layer M_1 a model designer requires that each Cisco has at least one Configuration. Each CisoConfiguration may contain one or more CiscoSlot and each Slot may contain one or more CiscoCard. hasConfig, hasSlot and hasCard are expressed as ObjectProperty.

Additionally, assume the modeller requires the disjointness of the classes Supervisor and SPA_interface. Moreover, Cisco7600Config. can have only three CiscoSlot and Cisco7600_Slot_1 can contain only card from Supervisor class.

A crucial task in model-driven engineering is the validation of models and metamodels. A valid model refers to its metamodel and satisfies all the restrictions and constraints. However, the validation of multiple layers may lead to inconsistency even if the consistency is satisfied between all adjacent layers.

For instance, in the previously described scenario, the model on layer M_1 and also the corresponding metamodel on layer M_2 are consistent. The inconsistency occurs when they are combined, i.e. consider the modelling restrictions like equivalence or disjointness for the whole model, covering multiple layers simultaneously instead of only two adjacent layers. If one would like to add, Avaya is a Shelf and Avaya is equivalent to concept Cisco7600 in M_1 and then Avaya and Cisco7600 are concepts in layer M_1. This equivalence condition does not cause any inconsistency of the modelling layer (M_1 and M_0 for instances).

However, combined with the constraints on the metamodel layer M_2 which requires the disjointness of Shelf and Chassis, this leads to a contradiction and therefore to an inconsistent ontology. Without capturing multiple layers in combination, this inconsistency is not detected since the adjacent layers M_1, M_0 and M_2, M_1 are consistent on its own.

7.2.2 OWL FA Syntax and Semantics

OWL FA specifies a layer number in class constructors and axioms to indicate the strata they belong to. Let $i \geq 0$ be an integer. OWL FA consists of an alphabet of distinct class names $\mathbf{V}_{\mathrm{C_i}}$ (for layer i), datatype names $\mathbf{V}_{\mathrm{D}}$, abstract property names $\mathbf{V}_{\mathrm{AP_i}}$ (for layer i), datatype property names $\mathbf{V}_{\mathrm{DP}}$ and individual (object) names ($\mathbf{I}$), together with a set of constructors (with subscriptions) to construct class and property descriptions (also called *OWL FA-classes* and *OWL FA-properties*, respectively).

The semantics of OWL FA are a model-theoretic semantics, which is defined in terms of interpretations. Given an OWL FA alphabet $\mathbf{V}$, a set of built-in datatype

names $\mathbf{B} \subseteq \mathbf{V}_{\mathrm{D}}$ and an integer $\mathrm{k} \geq 1$, an *OWL FA interpretation* is a pair $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$, where $\Delta^{\mathcal{J}}$ is the domain (a non-empty set) and $\cdot^{\mathcal{J}}$ is the interpretation function, which satisfy the following conditions below (where $0 \leq \mathrm{i} \leq \mathrm{k}$):

1. $\Delta^{\mathcal{J}} = \bigcup_{0 \leq \mathrm{i} \leq \mathrm{k}-1} {\Delta_A}^{\mathcal{J}_{\mathrm{i}}} \cup \Delta_{\mathbf{D}}$, where ${\Delta_A}^{\mathcal{J}_{\mathrm{i}}}$ is the domain for layer i and $\Delta_{\mathbf{D}}$ is the datatype domain.
2. ${\Delta_A}^{\mathcal{J}_{\mathrm{i}+1}} = 2^{{\Delta_A}^{\mathcal{J}_{\mathrm{i}}}} \cup 2^{{\Delta_A}^{\mathcal{J}_{\mathrm{i}}} \times {\Delta_A}^{\mathcal{J}_{\mathrm{i}}}}$ and $\Delta_{\mathbf{D}} \cap {\Delta_A}^{\mathcal{J}_{\mathrm{i}}} = \varnothing$.
3. $\forall\, \mathsf{a} \in \mathbf{V}_{\mathrm{I}} :\ \mathsf{a}^{\mathcal{J}} \in {\Delta_A}^{\mathcal{J}_0}$ and $\forall\, \mathsf{C} \in \mathbf{V}_{\mathrm{C_{i+1}}} :\ \mathsf{C}^{\mathcal{J}} \subseteq {\Delta_A}^{\mathcal{J}_{\mathrm{i}}}$.
4. $\forall\, R \in \mathbf{V}_{\mathrm{APi+1}} :\ R^{\mathcal{J}} \subseteq {\Delta_A}^{\mathcal{J}_{\mathrm{i}}} \times {\Delta_A}^{\mathcal{J}_{\mathrm{i}}}$ and $\forall\, T \in \mathbf{V}_{\mathrm{DP}} :\ T^{\mathcal{J}} \subseteq {\Delta_A}^{\mathcal{J}_0} \times \Delta_{\mathbf{D}}$.
5. $\bigcup_{\forall\, d \in \mathbf{B}} V(d) \subseteq \Delta_{\mathbf{D}}$, where $V(d)$ is the value space of d.
6. $\forall\, d \in \mathbf{V}_{\mathrm{D}}$, if $d \in \mathbf{B}$, then[1]
   (a) $d^{\mathcal{J}} = V(d)$, where $V(d)$ is the value space of d.
   (b) if $v \in L(d)$, then $(\text{“}v\text{”}\hat{}\hat{}d)^{\mathcal{J}} = L2\,V(d)(v)$, where $L(d)$ is lexical space of d and $L2\,V(d)$ is the lexical-to-value mapping of d.
   (c) if $v \notin L(d)$, then $(\text{“}v\text{”}\hat{}\hat{}d)^{\mathcal{J}}$ is undefined;

   otherwise, $d^{\mathcal{J}} \subseteq \Delta_{\mathbf{D}}$ and $\text{“}v\text{”}\hat{}\hat{}d \in \Delta_{\mathbf{D}}$.

In the rest, we assume that i is an integer such that $1 \leq i \leq k$. The interpretation function can be extended to give semantics to OWL FA-properties and OWL FA-classes. Let $RN \in \mathbf{V}_{\mathrm{APi}}$ be an abstract property name in layer i and R an abstract property in layer i. Valid OWL FA abstract properties are defined by the abstract syntax: $R ::= RN \mid R^{-}$, where for some $x, y \in {\Delta_A}^{\mathcal{J}_{\mathrm{i}-1}}$, $\langle x, y \rangle \in R^{\mathcal{J}}$ iff $\langle y, x \rangle \in R^{-\mathcal{I}}$. Valid OWL FA datatype properties are datatype property names.

Let $\mathsf{CN} \in \mathbf{V}_{\mathrm{C_i}}$ be an atomic class name in layer i, R an OWL FA-property in layer i, $o \in \mathbf{I}$ an individual, $T \in \mathbf{V}_{\mathrm{DP}}$ a datatype property name and C, D OWL FA-classes in layer i. Valid OWL FA-classes are defined by the abstract syntax:

$$
\begin{aligned}
C ::= {} & \top_{\mathrm{i}} \mid \bot \mid \mathsf{CN} \mid \neg_{\mathrm{i}} C \mid C \sqcap_{\mathrm{i}} D \mid C \sqcup_{\mathrm{i}} D \mid \{o\} \mid \exists_{\mathrm{i}} R.C \\
& \forall_{\mathrm{i}} R.C \mid \leqslant_{\mathrm{i}} nR \mid \geqslant_{\mathrm{i}} nR \\
& (\text{if} i = 1)\, \exists_1 T.d \mid \forall_1 T.d \mid \leqslant_1 nT \mid \geqslant_1 nT
\end{aligned}
$$

The semantics of OWL FA-classes are presented in Table 7.1.

A class C is *satisfiable* iff there exist an interpretation $\mathcal{I}$ s.t. $C^{\mathcal{J}} \neq \varnothing$; C subsumes D iff for every interpretation $\mathcal{I}$ we have $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$.

An OWL FA ontology $\mathcal{O}$ consists of $\mathcal{O}_1, \ldots, \mathcal{O}_{\mathrm{k}}$. Each $\mathcal{O}_{\mathrm{i}}$ consists of a TBox $\mathcal{T}_{\mathrm{i}}$, an RBox $\mathcal{R}_{\mathrm{i}}$ and an ABox $\mathcal{A}_{\mathrm{i}}$. An OWL FA *TBox* $\mathcal{T}_{\mathrm{i}}$ is a finite set of class inclusion axioms of the form $C \sqsubseteq_{\mathrm{i}} D$, where C, D are OWL FA-classes in layer i. An interpretation $\mathcal{I}$ satisfies $C \sqsubseteq_{\mathrm{i}} D$ if $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$. Let R, S be OWL FA abstract properties in layer i. An OWL FA *RBox* $\mathcal{R}_{\mathrm{i}}$ is a finite set of property axioms, namely property inclusion axioms ($R \sqsubseteq_{\mathrm{i}} S$), functional property axioms ($\mathsf{Func}_{\mathrm{i}}(R)$) and

[1] To simplify the presentation, we do not distinguish datatype names and datatype URIrefs here.

Table 7.1 OWL FA-classes

Constructor	DL syntax	Semantics
Top	$\top_{\mathrm{i}}$	$\Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}}$
Bottom	$\bot$	$\varnothing$
Concept name	CN	$\mathsf{CN}^{\mathcal{J}} \subseteq \Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}}$
General negation	$\neg_{\mathrm{i}} C$	$\Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}} \setminus C^{\mathcal{J}}$
Conjunction	$C \sqcap_{\mathrm{i}} D$	$C^{\mathcal{J}} \cap D^{\mathcal{J}}$
Disjunction	$C \sqcup_{\mathrm{i}} D$	$C^{\mathcal{J}} \cup D^{\mathcal{J}}$
Nominals	$\{\mathtt{o}\}$	$\{\mathtt{o}\}^{\mathcal{J}} = \{\mathtt{o}^{\mathcal{J}}\}$
Exists restriction	$\exists_{\mathrm{i}}\, R.C$	$\{x \in \Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}} \mid \exists\, y.\langle x,y\rangle \in R^{\mathcal{J}} \wedge y \in C^{\mathcal{J}}\}$
Value restriction	$\forall_{\mathrm{i}}\, R.C$	$\{x \in \Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}} \mid \forall\, y.\langle x,y\rangle \in R^{\mathcal{J}} \rightarrow y \in C^{\mathcal{J}}\}$
Atleast restriction	$\geqslant_{\mathrm{i}} mR$	$\{x \in \Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}} \mid \sharp\{y \mid \langle x,y\rangle \in R^{\mathcal{J}}\} \geq m\}$
Atmost restriction	$\leqslant_{\mathrm{i}} mR$	$\{x \in \Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}} \mid \sharp\{y \mid \langle x,y\rangle \in R^{\mathcal{J}}\} \leq m\}$
Datatype exists restriction	$\exists_1\, T.d$	$\{x \in \Delta_{A^{\mathcal{J}}_{0}} \mid \exists\, t.\langle x,t\rangle \in T^{\mathcal{J}} \wedge t \in d^{\mathcal{J}}\}$
Datatype value restriction	$\forall_1\, T.d$	$\{x \in \Delta_{A^{\mathcal{J}}_{0}} \mid \forall\, t.\langle x,t\rangle \in T^{\mathcal{J}} \rightarrow t \in d^{\mathcal{J}}\}$
Datatype atleast restriction	$\geqslant_1 mT$	$\{x \in \Delta_{A^{\mathcal{J}}_{0}} \mid \sharp\{t \mid \langle x,t\rangle \in T^{\mathcal{J}}\} \geq m\}$
Datatype atmost restriction	$\leqslant_1 mT$	$\{x \in \Delta_{A^{\mathcal{J}}_{0}} \mid \sharp\{t \mid \langle x,t\rangle \in T^{\mathcal{J}}\} \leq m\}$

transitive property axioms ($\mathsf{Trans}_{\mathrm{i}}(R)$). An interpretation $\mathcal{I}$ satisfies $R \sqsubseteq_{\mathrm{i}} S$ if $R^{\mathcal{J}} \subseteq S^{\mathcal{J}}$; $\mathcal{I}$ satisfies $\mathsf{Func}_{\mathrm{i}}(R)$ if, for all $x \in \Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}}$, $\sharp\{y \in \Delta_{A^{\mathcal{J}}_{\mathrm{i}-1}} \mid \langle x,y\rangle \in R^{\mathcal{J}}\} \leq 1$ ($\sharp$ denotes cardinality); $\mathcal{I}$ satisfies $\mathsf{Trans}_{\mathrm{i}}(R)$ if, for all $x, y, z \in \Delta^{\mathcal{J}}\, i-1$, $\{\langle x,y\rangle, \langle y,z\rangle\} \subseteq R^{\mathcal{J}} \rightarrow \langle x,z\rangle \in R^{\mathcal{J}}$. The semantics for datatype property inclusion axioms and functional axioms can be defined in the same way as those in OWL DL. Like in OWL DL, there is no transitive datatype property axioms.

Let $\mathtt{a}, \mathtt{b} \in \mathbf{I}$ be individuals, C_1 a class in layer 1, R_1 an abstract property in layer 1, l a literal, $T \in \mathbf{V}_{\mathrm{D}}$ a datatype property, X, Y classes or abstract properties in layer i;, E a class in layer i + 1 and S an abstract property in layer i+1. An OWL FA *ABox* $\mathcal{A}_1$ is a finite set of individual axioms of the following forms: $\mathtt{a} :_1 C_1$ called *class assertions*, $\langle \mathtt{a}, \mathtt{b}\rangle :_1 R_1$ called *abstract property assertions*, $\langle \mathtt{a}, l\rangle :_1 T$ called *datatype property assertions*, $\mathtt{a} = \mathtt{b}$ called *individual equality axioms* and $\mathtt{a} \neq \mathtt{b}$ called *individual inequality axioms*.

An interpretation $\mathcal{I}$ satisfies $\mathtt{a} :_1 C_1$ if $\mathtt{a}^{\mathcal{J}} \in C_1^{\mathcal{J}}$; it satisfies $\langle \mathtt{a}, \mathtt{b}\rangle :_1 R_1$ if $\langle \mathtt{a}^{\mathcal{J}}, \mathtt{b}^{\mathcal{J}}\rangle \in R_1^{\mathcal{J}}$; it satisfies $\langle \mathtt{a}, l\rangle :_1 T$ if $\langle \mathtt{a}^{\mathcal{J}}, l^{\mathcal{J}}\rangle \in T^{\mathcal{J}}$; it satisfies $\mathtt{a} = \mathtt{b}$ if $\mathtt{a}^{\mathcal{J}} = \mathtt{b}^{\mathcal{J}}$; it satisfies $\mathtt{a} \neq \mathtt{b}$ if $\mathtt{a}^{\mathcal{J}} \neq \mathtt{b}^{\mathcal{J}}$.

An OWL FA *ABox* $\mathcal{A}_{\mathrm{i}}$ is a finite set of axioms of the following forms: $X : E$ called *meta-class assertions*, $\langle X, Y\rangle : R$ called *meta-property assertions* and $X =_{\mathrm{i}-1} Y$ called *meta-individual equality axioms*. An interpretation $\mathcal{I}$ satisfies $X : E$ if $X^{\mathcal{J}} \in E^{\mathcal{J}}$; it satisfies $\langle X, Y\rangle : R$ if $\langle X^{\mathcal{J}}, Y^{\mathcal{J}}\rangle \in R^{\mathcal{J}}$; it satisfies $X =_{\mathrm{i}-1} Y$ if $X^{\mathcal{J}} = Y^{\mathcal{J}}$.

An interpretation $\mathcal{I}$ satisfies an ontology $\mathcal{O}$ if it satisfies all the axioms in $\mathcal{O}$. $\mathcal{O}$ is *satisfiable* (*unsatisfiable*) iff there exists (does not exist) such an interpretation $\mathcal{I}$ that satisfies $\mathcal{O}$. Let C, D be OWL FA-classes in layer i, C is *satisfiable* w.r.t. $\mathcal{O}$ iff there exist an interpretation $\mathcal{I}$ of $\mathcal{O}$ s.t. $C^{\mathcal{J}} \neq \varnothing_{\mathrm{i}}$; C subsumes D w.r.t. $\mathcal{O}$ iff for every interpretation $\mathcal{I}$ of $\mathcal{O}$ we have $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$.

7.2.3 Metamodelling with OWL FA

In this section, we present the way of expressing metamodelling with OWL FA. Although the layer numbers can/should be encapsulated by tools, there are two rules of thumb to help users to get the number right. Firstly, the subscript numbers are only used to indicate a sub-ontology (e.g. $\mathcal{O}_2$), a constructor (e.g. $\exists_2$) or axiom symbols (e.g. $\sqsubseteq_2$, $:_2$) in a sub-ontology. Secondly, subscript numbers for constructors and axiom symbols indicate the sub-ontology that the class descriptions constructed by these constructors and axioms belong to.

The following example shows how to model a physical device ontology with OWL FA notation. The main reason for the functional syntax is that it is obvious to see which layer or layer they belong to. Below, we show how to express a multilevel model with an OWL FA notation.

According to the PDDSL modelling architecture from Sect. 4.1, we can represent the class, property and instance relations in the M_2 layer. Example 2 shows how to describe class constructs by using OWL FA. Examples 3 and 4 show how to describe constraint, class and property assertions from the layer M_2 to M_1, respectively.

Example 2. Class construct in M_2 layer:

$$\mathsf{SlotContainer} \sqsubseteq_2 \mathsf{Element} \tag{7.1}$$

$$\mathsf{Shelf} \sqsubseteq_2 \mathsf{SlotContainer} \tag{7.2}$$

$$\mathsf{Chassis} \sqsubseteq_2 \mathsf{SlotContainer} \tag{7.3}$$

$$\mathsf{Configuration} \sqsubseteq_2 \mathsf{Element} \tag{7.4}$$

$$\mathsf{Slot} \sqsubseteq_2 \mathsf{Element} \tag{7.5}$$

$$\mathsf{Card} \sqsubseteq_2 \mathsf{Element} \tag{7.6}$$

Example 3. Constraint in M_2 layer:

$$\mathsf{SlotContainer} \sqsubseteq_2 \exists_2 \mathsf{configurations.Configuration} \tag{7.7}$$

$$\mathsf{Configuration} \sqsubseteq_2 \exists_2 \mathsf{slots.Slot} \tag{7.8}$$

$$\mathsf{Slot} \sqsubseteq_2 \exists_2 \mathsf{cards.Card} \tag{7.9}$$

Example 4. Classes and properties assertion from the layer M_2 to M_1:

$$(\mathsf{Cisco}, \mathsf{CiscoConfiguration}) :_2 \mathsf{configurations} \tag{7.10}$$

$$(\mathsf{CiscoConfiguration}, \mathsf{CiscoSlot}) :_2 \mathsf{slots} \tag{7.11}$$

$$(\mathsf{CiscoSlot}, \mathsf{CiscoCard}) :_2 \mathsf{cards} \tag{7.12}$$

$$\mathsf{CiscoConfiguration} :_2 \mathsf{Configuration} \tag{7.13}$$

$$\mathsf{CiscoCard} :_2 \mathsf{Card} \tag{7.14}$$

$$\mathsf{CiscoSlot} :_2 \mathsf{Slot} \tag{7.15}$$

$$\mathsf{Cisco7600} :_2 \mathsf{Chassis} \tag{7.16}$$

$$\mathsf{Avaya} :_2 \mathsf{SlotContainer} \tag{7.17}$$

$$\mathsf{CiscoCSR-1} :_2 \mathsf{Shelf} \tag{7.18}$$

This layered representation, i.e. representing the OWL FA knowledge base in sub-ontologies allows to represent entities that are classes and instances at the same time. Due to limitations of space, we could not show the complete OWL FA ontology in this chapter. However, the class, property and instance relations in M_1 layer can be described in the same manner.

7.2.4 Reasoning in OWL FA

Now, we briefly discuss some reasoning tasks in OWL FA. According to the layered architecture, the knowledge base $\mathcal{O}$ in OWL FA is divided into a sequence of knowledge bases $\mathcal{O} = \mathcal{O}_1, \ldots \mathcal{O}_\mathrm{k}$, whereas k is the number of layers. Since individuals in layer $i+1$ can be classes and properties in layer i, this also affects the axioms of the layer below. Hence, individual axioms in the knowledge base $\mathcal{O}_{\mathrm{i}+1}$ can be considered as class axioms in the knowledge base $\mathcal{O}_\mathrm{i}$.

In an OWL FA knowledge base $\mathcal{O}$, $\mathcal{O}_2$, …, $\mathcal{O}_\mathrm{k}$ are $\mathcal{SHIQ}$ knowledge bases, i.e. nominals are not allowed. A nominal in a higher layer can lead to unsatisfiability of the knowledge bases. An interesting feature of $\mathcal{O}$ is that there could be interactions between $\mathcal{O}_\mathrm{i}$ and $\mathcal{O}_{\mathrm{i}+1}$.

7.2.5 Preprocessing

In this section, we discuss how to reduce the reasoning problem in OWL FA into a reasoning problem in OWL DL.

Definition 1. Let $\mathcal{O} = \langle \mathcal{O}_1, \ldots, \mathcal{O}_\mathrm{k} \rangle$ be an OWL FA knowledge base, where each of $\mathcal{O}_1, \ldots, \mathcal{O}_\mathrm{k}$ is consistent. $\mathcal{O}^* = \langle \mathcal{O}_1^*, \ldots, \mathcal{O}_\mathrm{k}^* \rangle$, called the *explicit knowledge base*, is constructed by making all the implicit atomic class axioms, atomic property axioms and individual equality axioms explicit.

As we have a finite set of vocabulary, we have the following lemma:

Lemma 1. *Given an OWL FA knowledge base* $\mathcal{O} = \langle \mathcal{O}_1, \ldots, \mathcal{O}_\mathrm{k} \rangle$. *The explicit knowledge base* $\mathcal{O}^*$ *(OWL DL knowledge base) can be calculated from* $\mathcal{O}$ *in finite number of steps.*

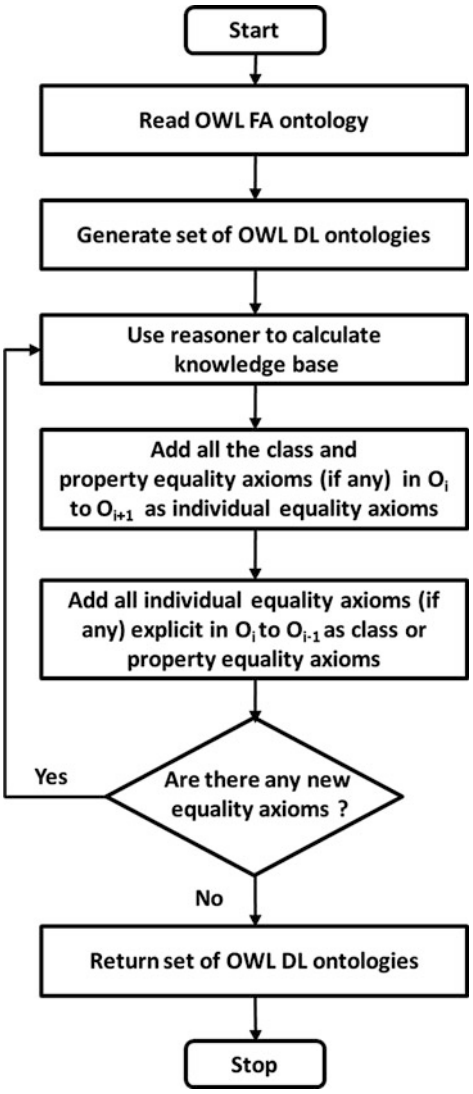

Fig. 7.9 Reduce algorithm

We now present the algorithm `Reduce` that will reduce an OWL FA knowledge base $\mathcal{O}$ into a set of OWL DL knowledge bases $\langle \mathcal{O}_1^*, \ldots, \mathcal{O}_k^* \rangle$. This algorithm is based on Definition 1 and Lemma 1. The algorithm takes an OWL FA KB $\mathcal{O}$ as input and returns a set of OWL DL KB $\langle \mathcal{O}_1, \ldots, \mathcal{O}_k \rangle$. The Algorithm `Reduce` is shown in Fig. 7.9.

The following theorem shows the termination of the algorithm `Reduce`, applied to an OWL FA KB $\mathcal{O}$:

Theorem 1. *Given an OWL FA knowledge base* $\mathcal{O} = \langle \mathcal{O}_1, \ldots, \mathcal{O}_k \rangle$, *then* `Reduce`$(\mathcal{O})$ *terminates.*

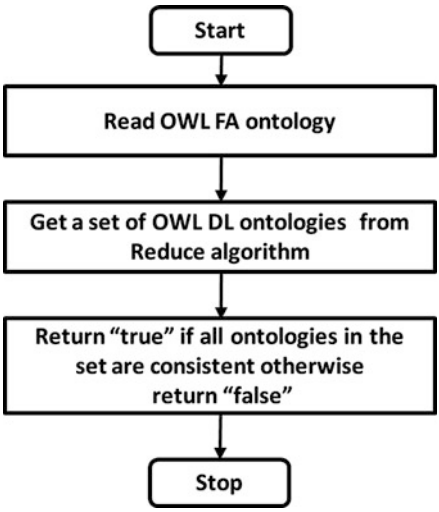

Fig. 7.10 Consistent algorithm

7.2.6 Consistency Checking

In this section, we present the algorithm `Consistent` that checks the consistency of an OWL FA knowledge base $\mathcal{O}$. We can reduce an OWL FA knowledge base to a collection of OWL DL knowledge bases; therefore, existing DL reasoner capabilities can be used. Consistency checking for OWL FA is done in two steps: First, we check the syntax of OWL FA. For example, $\Sigma = \{C \sqsubseteq_2 D, C \sqsubseteq_3 E\}$ is not well formed in OWL FA, because we do not allow OWL class construct between layers except an instance-of relation. Secondly, we check the consistency of each OWL DL knowledge base that is computed from the OWL FA knowledge base with an existing DL reasoner. The Algorithm `Consistent` is shown in Fig. 7.10.

We invite the reader to note that *check-dl-consistent* is a function call to a DL Reasoner.

Theorem 2. *Given an OWL FA knowledge base* $\mathcal{O} = \langle \mathcal{O}_1, \ldots, \mathcal{O}_k \rangle$. $\mathcal{O}$ *is consistent iff each* $\mathcal{O}_i^*$ $(1 \leq i \leq k)$ *is consistent.*

Theorem 2 shows we can reduce the OWL FA knowledge base consistency problem to the OWL DL knowledge base consistency problem.

7.2.7 Instance Retrieval

Instance retrieval in OWL FA is trivial because after the reduction process, we get a set of OWL 2 DL ontologies. Then, we could perform instance retrieval against those ontologies. However, without specifying a target ontology, it is not efficient since we have to go through all ontologies in a set. Therefore, we need a smart algorithm for instance retrieval for OWL FA in order to select the right ontology

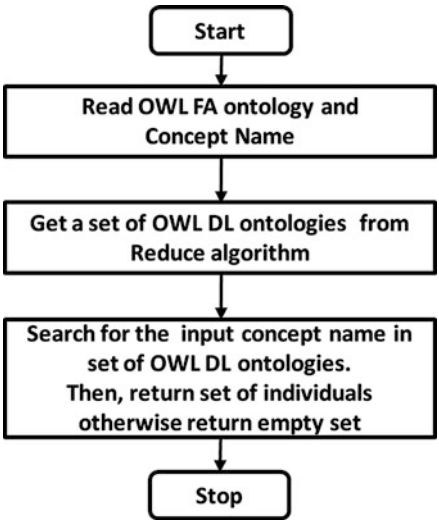

Fig. 7.11 InstanceOf algorithm

that contains a target concept. Firstly, we need to search for a target concept in each ontology of the set. This step does not require any DL reasoner. Then, we could perform instance retrieval against a selected ontology with a DL reasoner. A formal definition of instance retrieval for OWL FA is given in Definition 2.

Definition 2. Given an $ABox$ $\mathcal{A}_i$ and a query Q, i.e. a concept expression, find all individuals a such that a is an instance of Q, i.e. $\{a \mid \forall\ a \in \mathcal{A}_i, a : Q\}$.

We present the instance retrieval for OWL FA in Fig. 7.11. The algorithm *instanceOf* will take an OWL FA ontology $\mathcal{O}$ and a concept C as input. The algorithm returns a set containing the instances of concept C.

7.2.8 Justification on OWL FA

A justification for an entailment in an OWL FA ontology can be extended from justification for an entailment in an OWL DL ontology because we can reduce the reasoning problem in OWL FA to a reasoning problem in OWL DL. However, in the reduction process, a new axiom can be added to an OWL DL ontology. Therefore, if a justification for an entailment axiom in $\mathcal{O}_i$ contains those new axioms which have been added during the reduction process, we need to store an information for that axiom from another ontology in the lookup table. Concerning the ontology at the next higher modelling layer, if the new axiom has been added as class or property equality axioms, we can map those axioms from class or property equality axioms to individual equality axioms in the $\mathcal{O}_{i+1}$. Hence, we can retrieve the further justification from the upper ontology if needed. From a lower ontology, if a justification contains object equality axioms, we can map those object equality

axioms to class or property equality axioms in the $\mathcal{O}_{i-1}$ and then we can retrieve the further justification from the upper ontology if needed.

Definition 3. For an OWL FA ontology $\mathcal{O} = \langle \mathcal{O}_1, \ldots, \mathcal{O}_k \rangle$ and an entailment η_i where i is a layer number, a set of axioms $\mathcal{J}_i$ is a justification for η_i in $\mathcal{O}$ i. $\mathcal{J}_i$ may contain further justifications from $\mathcal{O}_{i+1}$ and/or $\mathcal{O}_{i-1}$ if the ontology $\mathcal{O}$ has added class or property equality axioms and/or added object equality axioms, respectively. The further justification can be retrieved from the information stored in the lookup table.

In order to keep trace of the new axioms that have been added in the reduction process, we extend the algorithm `Reduce` from Sect. 7.2.4. The algorithm takes an OWL FA knowledge base $\mathcal{O}$ as input and returns a set of OWL DL knowledge bases $\langle \mathcal{O}_1, \ldots, \mathcal{O}_k \rangle$ and a lookup table between new axioms and its correspond axiom. The lookup table is indexed by the layer, i.e. the axioms in the table refer to the layer which contains additional axioms after the reduction. These additional axioms affect also the justifications for this layer. Hence, we will later exploit the information from the lookup table about added axioms to compute the justification.

We now present the algorithm `Justification` that will retrieve a justification for an entailment in an OWL FA ontology. The algorithm takes an OWL FA KB $\mathcal{O}$ and an entailment η_i as input and returns a set of justification axioms $\mathcal{J}$. The algorithm `Justification` is shown in Fig. 7.12.

We invite the reader to note that *compute-dl-justification* is a function call to a DL Reasoner. The mapping table MP_i indicates added axioms to the ontology by the reduction. In this case, the adjacent ontology $\mathcal{O}^*_{i+1}$ or $\mathcal{O}^*_{i-1}$ is also relevant for the justifications of the ontology $\mathcal{O}^*_i$.

7.3 Metamodelling in Ontologies and Metamodelling in MOF

In this section, we lead to different possible interpretations. In contrast, in MDA this is realised by having multiple modelling layers according to the layered structure in MOF. Both are different methodologies to handle interpretations of individuals. Hence, in OWL DL (and DL), there are different properties on how instances are interpreted compared to the semantics of instances of a class in MDA.

In order to bridge the gap from OWL DL instance interpretations compared to MDA instance modelling, there are different approaches applied for OWL DL knowledge bases. The basic idea is to use different interpretations for different modelling layers or different universes of discourse [138, 216]. In Sect. 7.2, we already discussed the language OWL FA that provides a comprehensive solution for handling the layered modelling problem by still remaining decidable and using DL reasoning services for the metamodelling knowledge base.

While the previous section already discussed means for handling these different interpretations in order to support MOF-based metamodelling in OWL (DL), we continue in the next chapter with a further crucial difference in ontology modelling

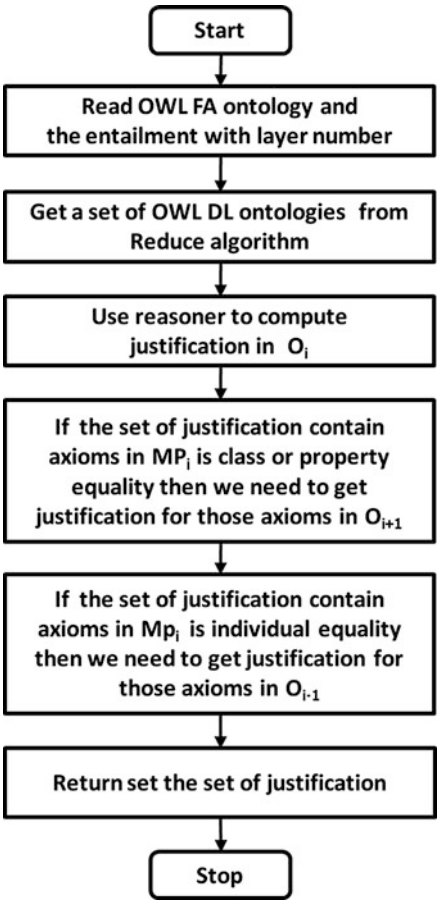

Fig. 7.12 The algorithm for computing a single justification for OWL FA

compared to MOF-based modelling. There are two different assumptions about instance relations or more precisely on missing instance relations. This problem is discussed in the next subsection (Sect. 7.3.1).

7.3.1 Open and Closed World Assumptions

We start this section with the *closed world assumption (CWA)* which is realised in database systems (including deductive database systems) and object-oriented models. The basic assumption is to consider the knowledge base as a closed world, i.e. we consider all statements in the knowledge base as true statements and everything that is not stated in or entailed by the knowledge base is considered as false. Everything that is unknown is considered as false.

A key property that is given by the CWA is the negative results in case of missing statements in the knowledge base. In order to understand the closed world

assumption, an intuitive example is a database query that asks for a certain fact (i.e. individual or instance) in a database. The query returns all statements that are contained in the database. However, if a fact is not in the database, we know that this fact does not hold. Here, the world is closed to those facts that are explicitly mentioned in the database and/or entailed by the knowledge base.

In OWL, the *open world assumption (OWA)* is realised which considers a given knowledge base as an incomplete part of an open world. The intuition behind this assumption is that a knowledge base can be extended by further statements and combined with other knowledge bases. The open world assumption of OWL is monotonic, i.e. adding new statements will never falsify previously inferred statements.

We demonstrate the difference between both assumptions on an axiom that is related to the running example from Fig. 7.8. The following axiom in the knowledge base (ABox) describes that an instance cisco7603 of the class Cisco has a configuration cisco7603conf which is an instance of the class CiscoConfiguration:

$$(cisco6703, cisco7603conf) \in hasConf \tag{7.19}$$

We assume there is no further axiom that describes cisco6703 has another configuration. Hence, in the CWA, it can be inferred that the instance cisco6703 has exactly one configuration, i.e. $cisco6703 \in \exists_{=1} hasConf.CiscoConfiguration$. Obviously, such a result would be also expected for a database query. However, in the OWA, this cannot be inferred since given knowledge base is considered as incomplete, i.e. there could be further configurations of $cisco6703$ that are asserted in another (unknown) knowledge base.

In various applications, it is known that the knowledge base is complete but due to the OWA of an OWL knowledge base, the inference does not take this completeness into account. One argument in favour of closed word reasoning in OWL is integrity constraints that cannot be enforced in the OWA. For instance, according to the running example from Fig. 7.8, each instance of Cisco is either an instance of CiscoCSR-1 or an instance of Cisco7600. This could be described by the following TBox axiom:

$$Cisco \sqsubseteq CiscoCSR-1 \sqcup Cisco7600 \tag{7.20}$$

However, given this TBox axiom in the OWA, we cannot guarantee that each instance of Cisco is an instance of either CiscoCSR-1 or Cisco7600. For the CWA, each instance of Cisco has to satisfy this condition.

In order to solve this problem, there are different approaches that allow closed world reasoning in OWL. Epistemic operators for OWL are described in [58, 81]. These additional operators allow to describe that a certain statement is known or asserted. For instance, using epistemic operators in the previous axiom allows the description of an integrity constraint, e.g. by using the K (known) and A (asserted) operator (cf. [81]):

$$\mathbf{K}Cisco \sqsubseteq \mathbf{A}CiscoCSR-1 \sqcup \mathbf{A}Cisco7600 \tag{7.21}$$

This axiom requires for each known instance of Cisco, i.e. there is assertion in the knowledge base; it is asserted that this instance is also an instance of the class CiscoCSR-1 or Cisco7600. If there is no such assertion in the knowledge base, this would be an inconsistent knowledge base. Obviously, this is not the case without epistemic operators.

Besides epistemic operators, there is another methodology in order to infer certain statements that cannot be derived in case of the OWA. In contrast to the CWA, it is not assumed that the knowledge base is complete. Instead, the domains of classes and object properties are closed. In the *closed domain assumption (CDA)*, the domain is restricted to an enumeration of individuals. This can be realised by progressively adding axioms defining that a class equivalent to an enumeration of individuals and likewise the domain or range of an object property equivalent to an enumeration of individuals. A special case of CDA can be realised by NBox as introduced in Sect. 5.1.5, where the domain of a concept or an object property is restricted to be their inferrable instances.

7.3.2 Ensuring Integrity Constraints in a Closed Domain

Although the OWA has many advantages, we have to ensure that validation of integrity constraints is still possible. In the following, we consider three basic integrity constraints and show how to define a closed domain to ensure the constraints.

We first start with explaining the unique name assumption (UNA) because it is part of the closed domain and a requirement for ensuring the integrity constraints.

Ensuring UNA

The *UNA* requires that instances have different names they are understood as different. The UNA is mainly considered in the CWA and CDA. In OWA the UNA is not considered, since two instances are not declared as different. Two instances i_1 and i_2 are different from each other ($i_1 \neq i_2$) if the nominal concept (provided by description logic $\mathcal{O}$) $\{o_1\}$ is disjoint with the nominal concept $\{o_2\}$ ($\{o_1\} \sqsubseteq \neg\{o_2\}$).

Ensuring Types of Instances

To ensure that a given instance $device1$ only has the asserted type $Device$ all other concepts must be declared as disjoint with $Device$ (e.g. $Device \sqsubseteq \neg(Slot \sqcup Configuration)$). Hence, instances of $Device$ cannot have the types $Slot$ or $Configuration$.

Ensuring Role Start and End Types

The following axiom in the TBox of a knowledge base restricts the instances of $Configuration$ to be connected with some instance of type $Slot$.

$$Configuration \sqsubseteq \exists\, hasSlot.Slot$$

To ensure that the type of instances connected via $hasSlot$ to slots is $Configuration$ we have to add the following axiom to the TBox:

$$\exists\, hasSlot.\top \sqsubseteq Configuration$$

Furthermore, all concepts must be declared as disjoint with $Configuration$ ($Configuration \sqsubseteq \neg(Device \sqcup Slot)$).

To ensure that configuration instances are only connected with slot instances via the $hasSlot$ role we have to introduce the following axiom:

$$Configuration \sqsubseteq \forall\, hasSlot.Slot$$

In addition, all concepts in the TBox must be disjoint with the end type $Slot$. For example, $Slot$ is declared as disjoint with $Configuration$ and $Device$ ($Slot \sqsubseteq \neg(Device \sqcup Configuration)$).

Ensuring Cardinalities

The following axiom in the TBox of a knowledge base describes the instances of $Configuration$ to be connected with exactly two slots.

$$Configuration \sqsubseteq= 2hasSlot.Slot$$

As we have seen in the previous section the knowledge base with the TBox above and the following ABox is consistent, although the number of slots is too low (for $conf1$) or too high (for $conf2$).

$$conf1 \in Configuration \qquad conf2 \in Configuration$$
$$slot1 \in Slot \qquad slot2, slot3, slot4 \in Slot$$
$$(conf1, slot1) \in hasSlot \qquad (conf2, slot2) \in hasSlot$$
$$(conf2, slot3) \in hasSlot$$
$$(conf2, slot4) \in hasSlot$$

To ensure cardinality constraints first we have to apply the UNA on all instances in the ABox as described above.

To avoid the assumption of further instances which are not logically inferrable from the current knowledge base, all concepts should be equivalent to the set of instances they are describing. Thus, we can include concepts $Configuration$ and $Slot$ into the NBox. Semantically, this is equivalent to adding the following definitions into the knowledge base:

$$Configuration \equiv \{conf1, conf2\}$$
$$Slot \equiv \{slot1, slot2, slot3, slot4\}$$

Consequently, $Configuration$ and $Slot$ will be inferred as disjoint ($Configuration \sqsubseteq \neg Slot$) given the UNA of individuals.

To get an inconsistency checking the number of slots for the two configurations ($conf1$ and $conf2$) in the knowledge base given above, we have to declare which instances of $Slot$ are *not* connected with $conf1$ and $conf2$, respectively. This can also be realised by putting $hasSlot$ into the NBox. Semantically, it is equivalent to explicitly asserting that each pair of individuals that are not inferred to have $hasSlot$ relation does not have such a relation. For example, $conf1$ does not $haveSlot\ slot2$ ($\{conf1\} \sqsubseteq \neg hasSlot.\{slot2\}$). With NBox, this and similar assertions for other pairs will be automatically included into the original knowledge base.

In general, it is possible to check the validity of integrity constraints. There are two disadvantages of checking constraints. At first, the number of additional axioms in the knowledge base increases and thus reduces the performance of reasoning tools. Secondly, if the ABox is modified (e.g. by adding, updating or deleting instances and role assertions), all the additional axioms must be rebuild.

Relations Between Models and Metamodels

Model-driven engineering has already been comprehensively discussed in Chap. 2. This section aims at clarifying some terms and definitions concerning models, metamodels and transformations and how they are related to each other in model-driven engineering.

The meaning of models and metamodels is informally described by Seidewitz [187] by describing what models and their relationships are and what their meaning is. Informally, a model is a set of statements describing a certain system. A model is interpreted according to a model theory to that the model belongs to. An interpretation is mainly a mapping of the modelled elements to the elements of the concrete system.

Seidewitz [187] defined a metamodel as a specification model where the considered system (system under study) is a model. According to this definition, a metamodel on what can be expressed in valid models of this metamodel. Metamodels are interpreted like models according to the underlying model theory.

This notion of metamodels, models and the corresponding interpretation is generalised to relations and interpretations between arbitrary adjacent modelling

layers in a layered modelling architecture in order to capture the well-established four-layered metamodelling architecture as it is realised in MOF models. In this case, an element in the modelling layer M_i is interpreted as an instance of an element of the corresponding metamodel above.

Atkinson and Kühne [7] extended the four-layer modelling architecture by adding a further modelling dimension. This two-dimensional modelling architecture allows linguistic and ontological metamodelling. The reasons for this extension are further requirements that occur in metamodelling like the support for deployment platforms where platform-specific artefacts are obtained from platform-independent by describing mappings between them. Another requirement is a high interoperability by allowing different representations of artefacts.

A more detailed consideration of metamodelling requirements in [7] indicates the advantages of more discrimination possibilities than the instance-of relation between layers or using metalevel descriptions to describe predefined concepts. In [7], linguistic and ontological metamodelling is demonstrated as two possible two-dimensional metamodelling techniques.

In linguistic metamodelling, the relation between entities crossing the modelling layers (M_i) is described as linguistic instance-of relations, i.e. the element in the model in layer, M_i is a linguistic instance of the corresponding class in layer M_{i+1}. The second dimension describes the ontological instance-of relation within one layer M_i and is used to define a (logical) type-of relation between entities. These relations are also called an intra-level instantiation between the (ontological) models O_1 and O_0 in layer M_i.

While the described linguistic metamodelling extends the four-layered architecture by these two ontological layers, the ontological metamodelling extends the notion of type-of relations (ontological instance-of relation) by meta-type relations, i.e. the ontological modelling architecture has more than the two layers O_1 and O_0. Traditional metamodelling architecture and techniques do not distinguish between ontological and linguistic instantiations.

Finally, we consider the work of Favre on representing and modelling of MDE in [67] which emphasises models, metamodels and transformation as basic concepts of MDE and gives an overall model of how they are related to each other. This representation of the MDE artefacts and the evolution process is called megamodel.

The proposed megamodel is an abstraction of models, metamodels and languages and their relations. The term system represents a certain element that is modelled in the MDE process. There are four relations between these elements: (1) The decomposedIn relation specifies whether a system is decomposed into subsystems or parts of the system. (2) The relation representationOf describes whether a model represents a certain system (system under study). (3) According to the set theory, the relation elementOf is used to define a set of systems like a language can be defined as a set of sentences. (4) Finally, the conformsTo relation describes whether a model conforms to its metamodel.

7.4 Conclusion

In this chapter, we have considered metamodelling and ontologies from two viewpoints. Firstly, we have presented snippets of common OWL metamodels in order to outline the usage of metamodels in ontology engineering. Moreover, the last part of this chapter has depicted argumentation and comparisons between metamodelling in OWL DL ontologies compared to metamodelling in MOF.

Secondly, OWL FA as a metamodelling enabled extension of OWL DL was introduced. OWL FA extends OWL DL with a multi-layered modelling architecture that allows reasoning across multiple modelling layers, where elements in a particular modelling layer might be considered as classes with respect to the next layer below, but at the same time, they serve as individuals regarding the next higher layer.

At the same time, this chapter concludes Part II. In the next part, we will discuss how to apply scalable reasoning services to provide consistency checking in ODSD.

http://www.springer.com/978-3-642-31225-0

Ontology-Driven Software Development
Pan, J.Z.; Staab, S.; Aßmann, U.; Ebert, J.; Zhao, Y. (Eds.)
2013, XVIII, 338 p., Hardcover
ISBN: 978-3-642-31225-0