Mixed Martial Arts Ontology developed in OWL and SWRL

Ali Khudiyev

November 2021

1 Introduction & Modelling

Mixed Martial Arts (MMA) is a full-contact combat sport based on striking, grapling and ground fighting. Top MMA organizations in the world are *Ultimate Fighting Championship (UFC)*, *Bellator MMA*, *Absolute Championship Akhmat (ACA)* which allow mixed-gender championships by separating male and female events. Such companies gain profit by selling fights as the end-user products each of which goes through the process of organization(i.e., fighter matching, place selection, pay-per-view prices, etc.) and evaluation(i.e., fight winner/drawer, fighter scoring). As in other combat sports, there are divisions among fighters participating in MMA. These divisions are to group fighters in the same weight class to make the championships fair since extra weights can be advantage and/or disadvantage depending on various factors. For example, UFC has the following weight classes[1]:

Weight class	**Weight**
Heavyweight	120.2 kg
Light heavyweight	102.1 kg
Middleweight	83.9 kg
Welterweight	77.1 kg
Lightweight	70.3 kg
Featherweight	65.8 kg
Bantamweight	61.2 kg
Flyweight	56.7 kg
Strawweight	52.5 kg

Table 1: UFC Weight Classes

Building an ontology is all about observing the relevant existing concepts and the relationships between them. For MMA ontology, there are whole bunch of concepts and relationships, however, I will illustrate the ones that are among the most important ones that make MMA what it is. The first concepts that come to the mind in MMA are the notion of organization and person. An MMA organization has a name and pays monthly salary to the organization workers. Organization workers can be CEO, judges, referees, organizers and fighters. Fighters participates at fights which is organized by organizers, controlled by referees and judged by judges. Judges gives points to both fighters based on which it is decided which fighter won/lost/drawed the fight. The fight takes place at some location and at a specific time, costs some amount of money which can be paid by a non organization worker to buy the fight. Every fight has a division that restricts how much the participating fighters weigh. Any two fighters have to have gender that is either male or female, in other words, (male vs female) match is impossible due to unfairness. The organization have certain age restrictions for its workers and also pays bonus money to a winning fighter. In this briefly written text, green words refer to concepts and orange ones refer to relations between them. Notion of classes, object and data properties become obvious by just reading this paragraph. The whole information described here can be represented as the knowledge graph shown below.

[1]https://wayofmartialarts.com/ufc-weight-classes-divisions

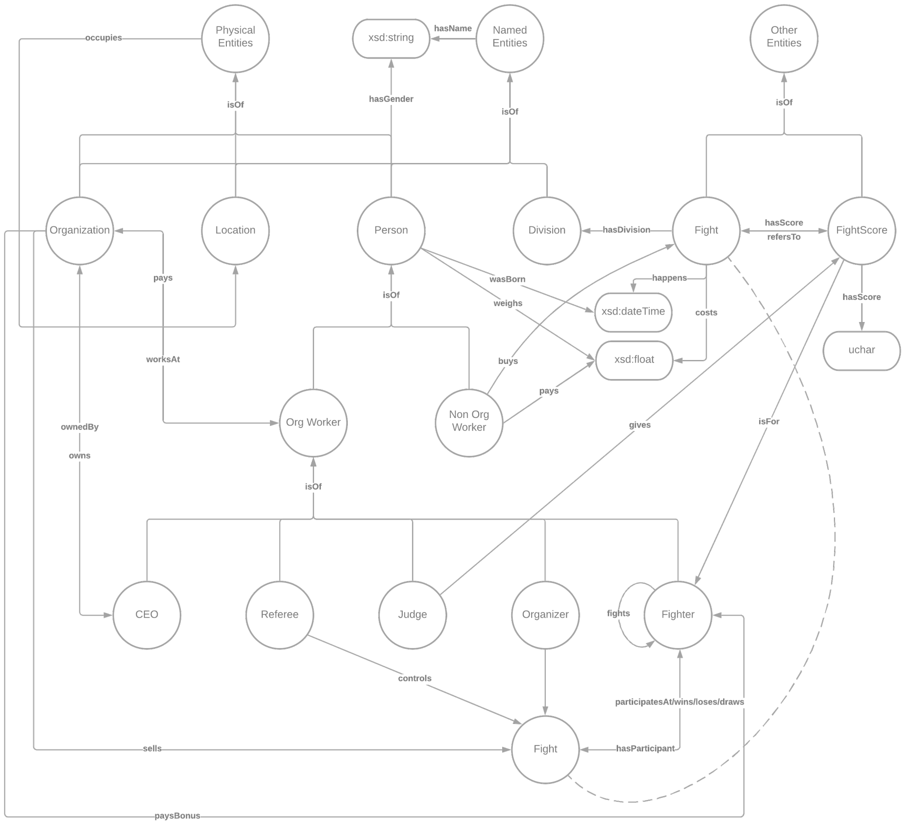


Figure 1: MMA Ontology Graph

In figure 1(see appendix C for the complete graph generated by *Ontograf* plugin), arrows represent relations or properties, circles represent objects and rectangles represent data types. The properties that link 2 objects are called object properties whereas the ones that link an object with a data type are called data properties. However, it is not enough to represent the whole just by using OWL alone and in fact, there are complementary axioms which represent some of the relationships hidden in the OWL graph. There are several rules that are among the most relevant ones as far as this project concerns and they are shown below in DL.

$$\text{Fight} \sqsubseteq = 2\text{hasParticipant.Fighter} \sqcap = 1\text{hasDivision.Division} \tag{1}$$

$$\text{Person} \equiv \text{OrgWorker} \sqcup \text{NonOrgWorker} \tag{2}$$

$$\text{OrgWorker} \equiv \text{Person} \sqcap \exists\text{worksAt.Organization} \tag{3}$$

$$\text{CEO} \equiv \text{OrgWorker} \sqcap \exists\text{owns.Organization} \tag{4}$$

$$\text{Fighter} \equiv \text{OrgWorker} \sqcap \exists\text{participatesAt.Fight} \tag{5}$$

$$\text{Division} \equiv \text{NamedEntities} \sqcap \{D1, D2, \ldots, D9\} \tag{6}$$

$$\text{Referee} \sqsubseteq \exists\text{controls.Fight} \tag{7}$$

$$\text{Judge} \sqsubseteq \exists\text{gives.FightScore} \tag{8}$$

$$\text{Organizer} \sqsubseteq \exists\text{organizes.Fight} \tag{9}$$

Axiom (1) indicates that any fight has to have exactly 2 participating fighters and a single division, however, the concept of fight is not strictly defined by the number of participants. The axioms (2-6) are to define equivalence between concepts that have strict definitions in this ontology. For example, the second axiom states that any person is either an organization worker or a non organization worker which is inherently true, and the sixth one is to define the concept of division(weight class) as one of 9 divisions(i.e., heavyweight, light heavyweight, lightweight and so on). The axioms from 7 to 9 show relationship between referees/judges/organizers and people who control/give/organize some fight/fightscore/fight with the minimum cardinality of 1.

$$\text{Organizer}(?o) \wedge \text{Fight}(?f) \wedge \text{sells}(?o, ?f) \wedge \text{wins}(?x, ?f) \rightarrow paysBonus(?o, ?x) \tag{10}$$

$$\text{Fight}(?f) \wedge \text{hasDivision}(?f, D1) \wedge \text{hasParticipant}(?f, ?x) \rightarrow \text{weighs}(?x, 52.5) \tag{11}$$
$$\text{Fight}(?f) \wedge \text{hasDivision}(?f, D2) \wedge \text{hasParticipant}(?f, ?x) \rightarrow \text{weighs}(?x, 56.7) \tag{12}$$
$$\ldots \rightarrow \ldots \tag{13}$$
$$\text{Fight}(?f) \wedge \text{hasDivision}(?f, D9) \wedge \text{hasParticipant}(?f, ?x) \rightarrow \text{weighs}(?x, 120.2) \tag{14}$$
$$\tag{15}$$

$$\begin{aligned} \text{Organizer}(?o) \wedge \text{Fight}(?f) \wedge \text{sells}(?o, ?f) \wedge \text{hasParticipant}(?f, ?x) \wedge \\ \wedge \text{hasParticipant}(?f, ?y) \wedge \text{differentFrom}(?x, ?y) \wedge \\ \wedge \text{hasGender}(?x, ?g) \rightarrow \text{hasGender}(?y, ?g) \end{aligned} \tag{16}$$

$$\begin{aligned} \text{isFor}(?fs1, ?x) \wedge \text{isFor}(?fs2, ?y) \wedge \text{differentFrom}(?x, ?y) \wedge \text{refersTo}(?fs1, ?f) \wedge \\ \wedge \text{refersTo}(?fs2, ?f) \wedge \text{hasScore}(?fs1, ?s1) \wedge \text{hasScore}(?fs2, ?s2) \wedge \\ \wedge\, swrlb : equal(?s1, ?s2) \rightarrow \text{draws}(?x, ?f) \wedge \text{draws}(?y, ?f) \end{aligned} \tag{17}$$

$$\begin{aligned} \text{isFor}(?fs1, ?x) \wedge \text{isFor}(?fs2, ?y) \wedge \text{differentFrom}(?x, ?y) \wedge \text{refersTo}(?fs1, ?f) \wedge \\ \wedge \text{refersTo}(?fs2, ?f) \wedge \text{hasScore}(?fs1, ?s1) \wedge \text{hasScore}(?fs2, ?s2) \wedge \\ \wedge\, swrlb : greaterThan(?s1, ?s2) \rightarrow \text{wins}(?x, ?f) \wedge \text{loses}(?y, ?f) \end{aligned} \tag{18}$$

$$\text{refersTo}(?fs1, ?f) \wedge \text{refersTo}(?fs2, ?f) \wedge \text{gives}(?j, ?fs1) \rightarrow \text{gives}(?j, ?fs2) \tag{19}$$

The SWRL rule 10 states that any fighter with the minimum record of 1 won fight is paid bonus money by the organization, rules for weight classes(till 15) asserts fighters' weights fighting in a certain division. The rule 16 indicates that the fighters with different genders cannot fight against each other and likewise, it is true for the weight classes. To be able to obtain winners/losers/drawers, rules 18-17 compare the scores of both fighters and conclude one of three possibilites: winner/loser, loser/winner and drawer/drawer with regard to both fighters. The last rule indicates that a judge who gives a score to a fighter for particular fight is also the one who gives a score to the other fighter for the same fight. There are a few more rules that are discussed later on with implementation details in Protégé.

2 Implementation in Protégé

The implementation goes through several stages: *creating class hierarchy and defining class equivalance(s)/ subsumption(s)/ disjuntion(s)*, then *defining relations*, *defining SWRL rules* and finally *creating individuals* to test and observe the implications. The class hierarchy is created by simply creating concepts used in the ontology and the class properties such as equivalance[2], subsumption, disjuction between them are defined within the scope of the same classes. Relations are to link different concepts with their relevant predicates and have properties(i.e., reflexivity, transivity, etc.) as well. These 2 steps are what can be done using only OWL and they are carefully developed to avoid unintentional situations later on. The third step, which is defining SWRL rules, is developed by using SWRLTab in Protégé and tested with the OWL individuals that are created at the final development step of the ontology.

[2]Although equivalance is rarely used as it is mainly for merging two separately developed ontologies, there are classes which make it relevant to use it in this project.

2.1 Creating and Defining Classes

Concepts or classes used in this ontology have been shown with UML class diagram in the previous modelling section. The diagram also covers object and data properties for all classes and it can be observed that there are several concepts that share the same properties. For example, *organization*, *person* and *location* are concepts that have names and occupy some place on the Earth described by the couple of properties *hasName* and *occupies* respectively. In addition to these 3 classes, the *division* has also a name but it does not occupy any place since it is an intangible object. Due to such similarities and/or dissimilarities between such concepts, 3 top-level concepts have been implemented to encapsulate the relavant or similar subclasses. These top-level classes are *NamedEntities*, *PhysicalEntities* and *OtherEntities* and the rest are defined as shown below.

- NamedEntities
  - Organization
  - Location
  - Division
  - Person
    - NonOrgWorker
    - OrgWorker
      - CEO
      - Judge
      - Referee
      - Organizer
      - Fighter
- PhysicalEntities
  - Organization
  - Location
  - Person
    - NonOrgWorker
    - OrgWorker
      - CEO
      - Judge
      - Referee
      - Organizer
      - Fighter
- OtherEntities
  - Fight
  - FightScore

Figure 2: owl:Thing subclasses

It is more relevant to have a class hierarchy as shown in figure 2 since it is obvious that the domain of *hasName* object property is just *NamedEntities* and the domain of *occupies* object property is *PhysicalEntities* class where as elements of *OtherEntities* can neither have names nor occupy locations.

2.1.1 Disjoint classes

Disjoint classes are useful when we semantically separate distinct sets whose elements are guaranteed to be different from each other. The benefits of doing so is to be able to increase accuracy and sufficency of inference made with non-unique name assumption. List of the disjoing classes in this ontology is shown below.

Class	**Disjoint with**
OtherEntities	NamedEntities, PhysicalEntities
NamedEntities	OtherEntities
PhysicalEntities	OtherEntities
NonOrgWorker	OrgWorker
OrgWorker	NonOrgWorker
CEO	Fighter, Referee, Judge, Organizer
Fighter	CEO, Referee, Judge, Organizer
Referee	CEO, Fighter, Judge, Organizer
Judge	CEO, Fighter, Referee, Organizer
Organizer	CEO, Fighter, Referee, Judge

Table 2: Disjoint classes in Protégé

Equivalence of classes is not commonly used while developing an ontology since it is often useful for merging separately developed ontologies. However, it is useful for this particular ontology when we talk about the classes *Person*, *OrgWorker*, *CEO*, *Fighter* and *Division*. In Protégé, equivalence relations are defined by using Equivalent To description. Here are the equivalence relations rewritten in Protégé:

Class	Equivalent To
Person	OrgWorker or NonOrgWorker
OrgWorker	Person and (worksAt some Organization)
CEO	OrgWorker and (owns some Organization)
Fighter	OrgWorker and (participatesAt some Fight)
Division	NamedEntities and {D1, D2, . . . , D9}

Table 3: Equivalence in Protégé

2.1.2 Classes under subclass-of

From the hierarchy of classes shown previosly, subclass-of relations between classes are already obvious. In addition to obvious ones, the implementation is a bit more detailed; there are some classes that are not simply a subclass of another user-defined class but also *dynamically created classes*[3] by Protégé. For example, *Fight* is of *OtherEntities* and subclass of a class each of whose elements has 2 different participating fighters. To add a new subclass-of relation in Protégé, SubClass Of description is used.

Class	SubClass Of
Fight	OtherEntities hasParticipant exactly 2 Fighter hasDivision exactly 1 Division
FightScore	OtherEntities isFor exactly 2 Fighter refersTo exactly 1 Fight
Person	NamedEntities and PhysicalEntities
Judge	OrgWorker and (gives gives FightScore)
Referee	OrgWorker and (controls some Fight)
Organizer	OrgWorker and (organizes some Fight)
Organization	NamedEntities and PhysicalEntities
Location	NamedEntities and PhysicalEntities

Table 4: Subclass-of in Protégé

2.2 Defining Relations

A relation in logic has some arity, which is usually called k-ary relation. When k is 1, it is called unary and when k is 2, it is called binary relation. In Protégé, only binary relations are developed and therefore, we give off some level of expressiviness as a trade-off with reduced computation complexity. Due to such expressivity, any relation with more than arity 2 has to be reformulated with the help of additional concepts and relations. It is the same case with the classes *Fight*, *FightScore* and *Fighter* in this project; as it is impossible to have a 3-ary relation describing a *score* of *fighter* in a particular *fight*. There are also several properties of binary relations such as functional, symmetric, reflexive, transtive and so on that describes the nature of the relation. These properties are also used in the developement when it is relevant and necessary to avoid illogical KB(Knowledge Base) and/or inferences.

[3]Class all of whose elements satisfy the given set of formulas.

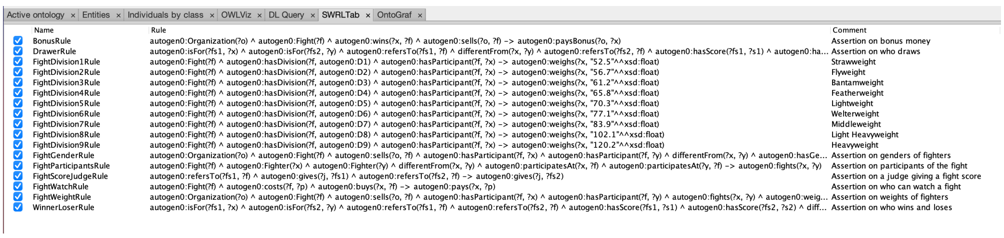
Relation	Domain	Range	Type	Description
costs	Fight	xsd:float	functional	Fight costs some amount of money
happens	Fight	xsd:dateTime	functional	Fight happens at some date
hasGender	Person	xsd:string	functional	Person has a gender
hasName	NamedEntities	xsd:string	functional	Any concept has a name
hasScore	FightScore	xsd:unsignedByte	functional	Fight score is an 8 bit positive integer
pays	NonOrgWorker	xsd:float	-	Non organization worker pays some amount of money to buy fights
wasBorn	Person	xsd:dateTime	functional	Person was born at some date time
weighs	Person	xsd:float	functional	Person has a weight

Table 5: Data properties

The MMA ontology does not concern about changing facts through time and therefore, it can contain functional relations such as *weighs* that represent state of an object at a particular slice of time. In addition to data properties shown in the table 5, all object properties are listed in the table 8(see appendix A).

2.3 Defining SWRL Rules

SWRL rules can be defined in Protégé by accessing *SWRLTab* plugin since the software does not inherently support such rules. The rules have to be in horn clauses, in other words, only one positive literal is allowed in the formula. Here are the complete list of rules created in Protégé:



Name	Rule	Comment
BonusRule	autogen0:Organization(?o) ^ autogen0:Fight(?f) ^ autogen0:wins(?x, ?f) ^ autogen0:sells(?o, ?f) -> autogen0:paysBonus(?o, ?x)	Assertion on bonus money
DrawerRule	autogen0:isFor(?fs1, ?x) ^ autogen0:isFor(?fs2, ?y) ^ autogen0:refersTo(?fs1, ?f) ^ differentFrom(?x, ?y) ^ autogen0:refersTo(?fs2, ?f) ^ autogen0:hasScore(?fs1, ?s1) ^ autogen0:ha...	Assertion on who draws
FightDivision1Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D1) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "52.5"^^xsd:float)	Strawweight
FightDivision2Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D2) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "56.7"^^xsd:float)	Flyweight
FightDivision3Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D3) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "61.2"^^xsd:float)	Bantamweight
FightDivision4Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D4) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "65.8"^^xsd:float)	Featherweight
FightDivision5Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D5) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "70.3"^^xsd:float)	Lightweight
FightDivision6Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D6) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "77.1"^^xsd:float)	Welterweight
FightDivision7Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D7) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "83.9"^^xsd:float)	Middleweight
FightDivision8Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D8) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "102.1"^^xsd:float)	Light Heavyweight
FightDivision9Rule	autogen0:Fight(?f) ^ autogen0:hasDivision(?f, autogen0:D9) ^ autogen0:hasParticipant(?f, ?x) -> autogen0:weighs(?x, "120.2"^^xsd:float)	Heavyweight
FightGenderRule	autogen0:Organization(?o) ^ autogen0:Fight(?f) ^ autogen0:sells(?o, ?f) ^ autogen0:hasParticipant(?f, ?x) ^ autogen0:hasParticipant(?f, ?y) ^ differentFrom(?x, ?y) ^ autogen0:hasGe...	Assertion on genders of fighters
FightParticipantsRule	autogen0:Fight(?f) ^ autogen0:Fighter(?x) ^ autogen0:Fighter(?y) ^ differentFrom(?x, ?y) ^ autogen0:participatesAt(?x, ?f) ^ autogen0:participatesAt(?y, ?f) -> autogen0:fights(?x, ?y)	Assertion on participants of the fight
FightScoreJudgeRule	autogen0:refersTo(?fs1, ?f) ^ autogen0:gives(?j, ?fs1) ^ autogen0:refersTo(?fs2, ?f) -> autogen0:gives(?j, ?fs2)	Assertion on a judge giving a fight score
FightWatchRule	autogen0:Fight(?f) ^ autogen0:costs(?f, ?p) ^ autogen0:buys(?x, ?f) -> autogen0:pays(?x, ?p)	Assertion on who can watch a fight
FightWeightRule	autogen0:Organization(?o) ^ autogen0:Fight(?f) ^ autogen0:sells(?o, ?f) ^ autogen0:hasParticipant(?f, ?x) ^ autogen0:hasParticipant(?f, ?y) ^ autogen0:fights(?x, ?y) ^ autogen0:weig...	Assertion on weights of fighters
WinnerLoserRule	autogen0:isFor(?fs1, ?x) ^ autogen0:isFor(?fs2, ?y) ^ autogen0:refersTo(?fs1, ?f) ^ autogen0:refersTo(?fs2, ?f) ^ autogen0:hasScore(?fs1, ?s1) ^ autogen0:hasScore(?fs2, ?s2) ^ diff...	Assertion on who wins and loses

Figure 3: SWRL rules implemented in Protégé

There are extra implemented rules shown in the figure 3 which are also helpful for certain situations. For example, the rule *FightWeightRule* asserts that any couple of fighters, who fight against each other, have to be in the same weight class and *FightParticipantsRule* reads that 2 fighters fight against each other if they are participants of the same fight. In the other hand, the rule *FightWatchRule* is used to derive the fact that any non-organization worker, who has bought the fight, has also paid money for it.

2.4 List of Individuals

There are a bunch of individuals in the ontology, however, a few are discussed in this paper. The important individuals that are shown below(table 6) are the ones for which reasoner inferred more/interesting information than others. The list of some created individuals and inferences on them are given in the table 9(see appendix B).

Individual	Asserted
Org	Organization sells Fight1, sells Fight2 hasName UFCA
OrgCEO	Person owns Org hasName Nihad
OrgCEO2	owns Org
OrgFighter1	participatesAt Fight1 hasGender male hasName Telman
OrgFighter2	Fighter hasName Bashir
Fight1	hasDivision D1 hasGivenScore FightScore1_2 hasParticipant OrgFighter2, costs 25.0
FightScore1_1	FightScore isFor OrgFighter1 refersTo Fight1, hasScore 15
FightScore1_2	FighScore isFor OrgFighter2, hasScore 17

Table 6: Individuals

2.5 SPARQL Queries

SPARQL helps to send queries to the knowledge base and it is not supposed to work with the inferred data unlike Snap SPARQL. A simple query is shown in the figure below which essentially asks for the data that matches the given conditions. In this example, the query reads that any individual(fighter) who has a specified gender, name and a given score according to the fights fought. Only single fighter was returned since *OrgFighter1* is the only one that has an explicitly specified gender and *participates at* some fight(*Fight1*).

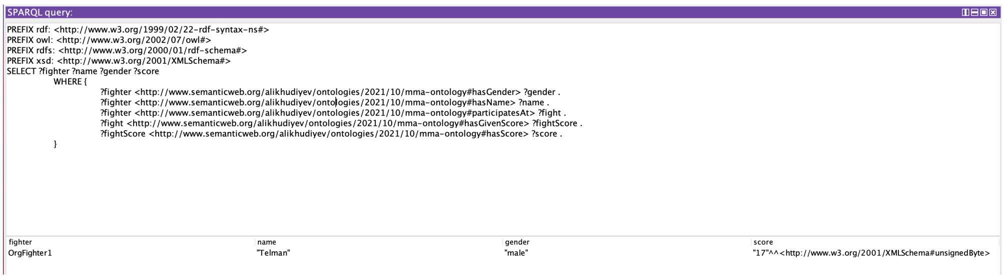


Figure 4: Simple Query

The SPARQL query shown below uses UNION and FILTER keywords that helps to group and filter individuals respectively. First, all individuals named *UFCA*(Org), *Nihad*(OrgCEO), *Narmin*(OrgFighter4) and *Gulshan*(-) are grouped together and then filtered according to their specified genders. In this example, all males are filtered while females or any individuals with any unspecified gender are dropped.

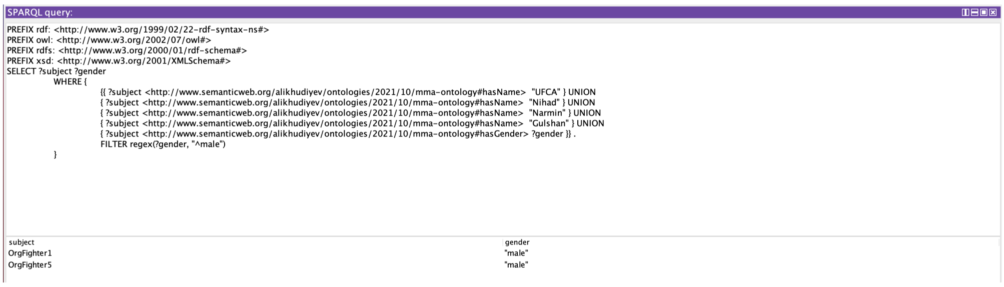


Figure 5: Query 2

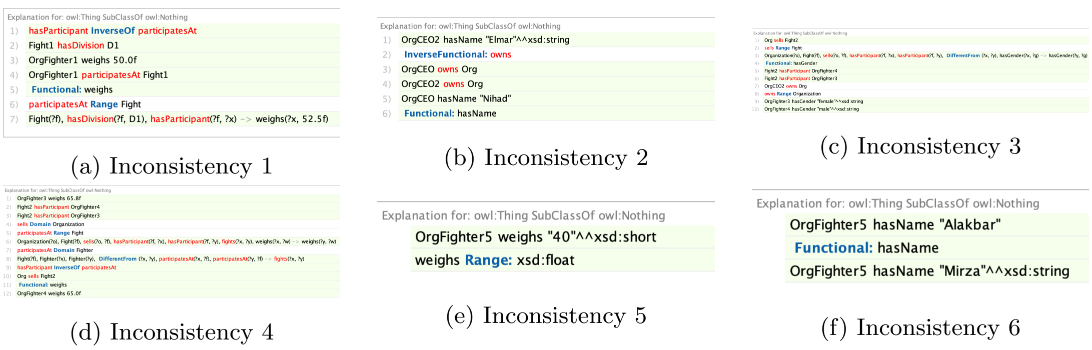
3 Reasoning with Pellet

3.1 Consistency check

First thing after the initial development of an ontology is to test it by running the reasoner. There are 2 possibilities after running the reasoner: the ontology is inconsistent or everyhing works just fine. The MMA ontology is simply consistent for the matter of well-developed concepts and relations. However, I demonstrate a few cases in which the system would be inconsistent:

1. Assigning irreleant weight to a fighter fighting in a certain division (violates *division weight* rule)
2. Creating a new CEO with a unique name and assigning him/her to the already assigned Organization (violates functional *ownedBy* relation or functional *hasName* relation)
3. Making a male vs. female fight (violates *same fighter gender* rule)
4. Making different weight classes to fight (violates *equal fighter weights* rule)
5. Assigning wrong data type value to an individual through some data property (violates data type restriction)
6. Giving more than 1 name/birth date/gender to a person (violates max cardinality rule of functional relations)
7. Making 3 different fighters fight in the same fight (violates max cardinality rule of a fight)

There are many situtations in which the knowledge base would be considered inconsistent, however, underlying principles of consistency are not as many as specific cases. To illustrate some of those principles in MMA ontology, I demonstrate examples for *data type violation* and *max cardinality violation* in different cases. The firt 6 cases shown above are illustrated in the figure given below.



(a) Inconsistency 1 (b) Inconsistency 2 (c) Inconsistency 3

(d) Inconsistency 4 (e) Inconsistency 5 (f) Inconsistency 6

Figure 6: Inconsistency explanations given by Protégé

3.2 Inferences made by Pellet

Pellet is OWL 2 DL reasoner developed in Java that provides functionality for consistency checking of ontologies, computing classification hierarchy, explaining inferences and answering to SPARQL queries. *Pellet* can also be used with Jena or OWL-API libraries although none of them are explicitly used in the MMA ontology development. Giving the list of individuals, *Pellet* can infer their types, object properties, data properties, relations with other individuals(i.e., *sameAs* or *differentFrom*). The most important and/or interesting inferences for individuals are shown in the table below[4].

[4]For more inferences made by Pellet, see table 9(appendix B).

Individual	Inferred
Org	occupies LocAz ownedBy OrgCEO2, ownedBy OrgCEO paysBonus OrgFighter2
OrgCEO	CEO sameAs OrgCEO2
OrgCEO2	CEO sameAs OrgCEO hasName Nihad
OrgFighter1	Fighter fights OrgFighter2 loses Fight1 weighs 52.5
OrgFighter2	fights OrgFighter1 participatesAt Fight1 wins Fight1 hasGender male, weighs 52.5
Fight1	Fight hasGivenScore FightScore1_1 hasParticipant OrgFighter1

Table 7: Individuals

The individual *OrgCEO* was found to be a CEO by Pellet because of equivalence relation - an individual is CEO if it is a person and it owns an organization. In the other hand, due to subsumption the reaoner was able to find that the individuals *OrgCEO*, *OrgFighter1*, *OrgJudge1*, etc. are of type *NamedEntities*. These results concerning subsumption relations can be observed by using *DL Query* in Protégé since it does not display this information by default which is probably an interface bug. The reasoner also found out that *OrgCEO* and *OrgCEO2* are actually the same instances and the reason is the maximum cardinality of the property called *owns* which is one. Since each individual owned the same organization, it was deduced that these individuals are the same and therefore, *OrgCEO2* has also the same name(Nihad) as *OrgCEO*.

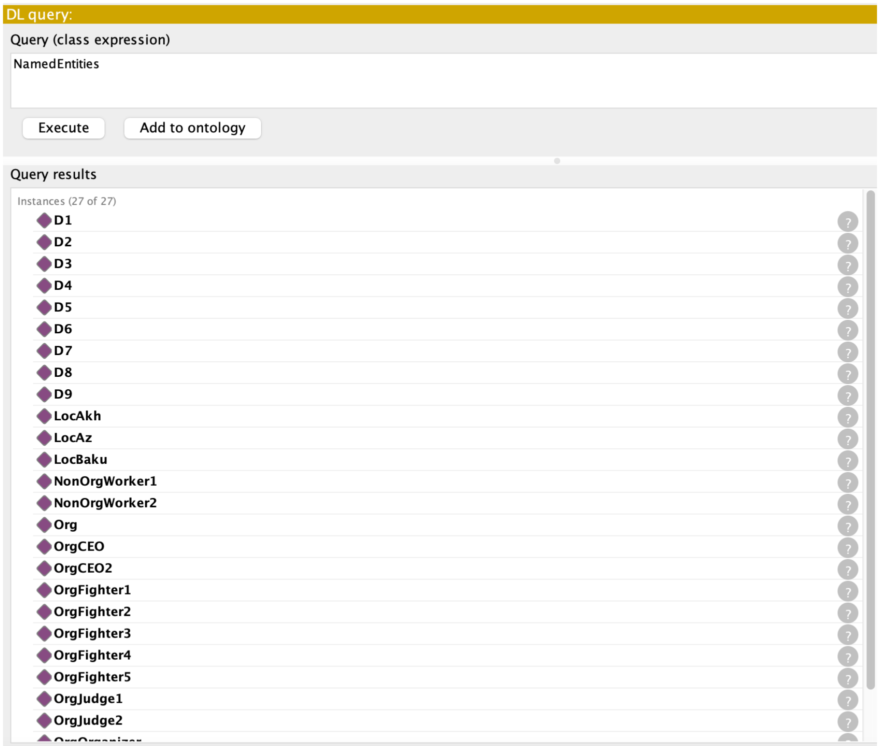


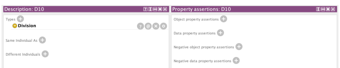
Figure 7: DL Query results

Other than using equivalences, subsumptions, disjutions, Pellet is able to infer the types of individuals just by looking at the defined properties on them. For example, the type of *OrgFighter1* was found to be Fighter not limited to but simply because of the property *participatesAt* which has a domain initialized as Fighter. Likewise, *OrgCEO* is CEO not only because of class equivalence but also because of the given domain of *owns* property.

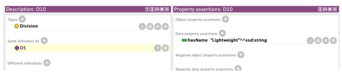
3.3 Assumptions used in OWL

OWL has a couple of assumptions about any ontology: *open world assumption* and *non-unique name assumption*. These assumptions alter the process of inference significantly. *Open world assumption* states that an unobserved piece of information does not necessarily imply its falsity and *non-unique name assumption* points us to the possbility of two different names/representations referring to the same object. Contrary to *closed world assumption*, where unobserved fact is false, and *unique name assumption*, where every object has a unique name, these assumptions help us to be more expressive in trade-off with the increased computational cost.

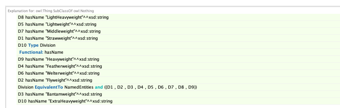
As most of the ontologies developed in OWL, the MMA ontology has also showcases for OWA(Open World Assumption) and Non-unique Name Assumption. For example, creating an extra tenth individual of class *Division* would result with an inconsistency in the knowledge base with CWA(Closed World Assumption) whereas it is totally fine in our case due to OWA and Non-unique Name Assumption. Since in the definition(see 6) of the *Division* class, the individual D10 is not explicitly mentioned, CWA leads to inconsistency. However, it is totally fine with OWA since D10 could be the same individual as as any D0-9 which is not restricted by UNA(Unique Name Assumption). In fact, when the reasoner is started in Protégé, it tries to find new information about D10 and if not found, it has the ability to say "I don't know" by not giving an inconsistency error and not infering its equivalent individual at the same time. However, if its name is given through the predicate *hasName*, it is able to tell whether the knowledge base has inconsistency or D10 is actually the same as some other D0-9 individual. The figures below illustrate this process:



(a) Not enough information to infer anything



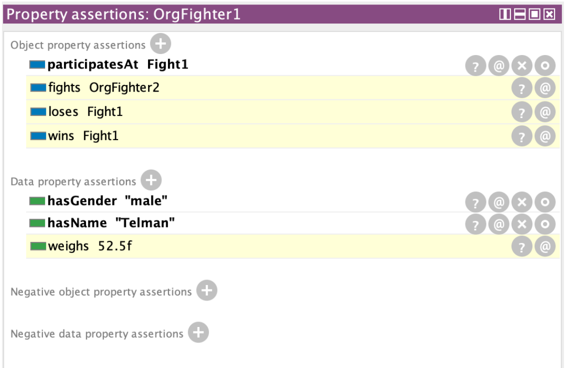
(b) Enough information to infer the individual the same as with D10 since *hasName* has functional property



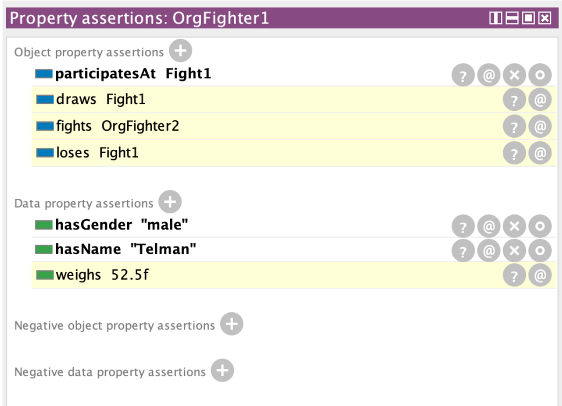
(c) Enough information to deduce inconsistency since there have to be 9 individuals in *Division* each of which has to have unique name(*xsd:string*) and there are 10 individuals of *Division* with 10 different names

Figure 8: OWL Assumptions Showcases

Another showcase for non-unique name assumption would be the use *differentFrom* relation in the relevant rules such as *FightWinnerLoserRule*, *FightDrawerRule* and so on. *FightWinnerLoserRule*(see 18) is to detect the loser of the fight based on the winner whereas *FightDrawerRule*(see 17) is to decide whether the fighters finish the fight with a draw based on their post fight scores. If the *differentFrom(?x, ?y)* was not included explicity in *FightWinnerLoserRule*, it would imply that any same fighter loses the fight which is won by him/her which is illogical and is prevented by indicating that the participants of the fight have to different from each other. Likewise, if the *fights(?x, ?y)* was not included explicitly in *FightDrawerRule*, it would imply that any fighter draws the fight without any affect of his/her score since when $x = y$, both fighter seem to have the same score and thus, draws the participated fight. The use of relation *fights(?x, ?y)* is just as good as using *differentFrom(?x, ?y)* since it is irreflexive, meaning that it cannot take the same element(fighter) as a domain and range at the same time.



(a) *FightWinnerLoserRule* without *differentFrom* relation



(b) *FightDrawerRule* without *fights* relation

Figure 9: OWL Assumptions Showcases

4 Evaluation & Conclusion

OWL and SWRL help us to create ontologies with certain subset of first order logic that makes the inference decidable. To do so, it sacrifices some sort of expressiviness that could be otherwise used to develop certain ontologies(e.g. an ontology that requires creation of new individuals under some conditions). Another type of problem may arise when trying to represent object states that is dynamic over time since there is no concept of time in OWL and SWRL. Consider we want to simulate *Conway's Game of Life*[5] by using SWRL rules. Since there are cells that are either alive or dead depending on the state of the neighbours, it is not very intuitive to represent a cell which can transform from being dead to alive or vice versa as iterations are developed. However, we could still simulate the GoL by representing the concept of time/iteration as an extra dimension. After thinking on the proper way of representing time, I settled in the following representation: *each individual cell has its own (current) time or iteration rather than a global notion of time.* This decision comes from the realization that rules can be applied randomly when relevant which interrupts the synchronization between cell states and the iteration that they are in. So, rather than having a global notion of iteration, each cell has its own sense of iteration which is developed as the reasoner processes or does some transformation upon the cell.

$$
\begin{aligned}
\mathrm{Time}(T)\\
\mathrm{AliveCell}(c1)\\
\mathrm{DeadCell}(c2)\\
\mathrm{DeadCell}(c3)\\
\mathrm{AliveCell}(c4)\\
\mathrm{Time}(T) \wedge \mathrm{hasValue}(T, t) &\implies \mathrm{hasValue}(T, t+1)
\end{aligned}
$$

Figure 10: Notion of global time

[5] https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

$$
\begin{gathered}
Cell(c1)\\
Cell(c2)\\
Cell(c3)\\
Cell(c4)\\
isLiveAt(c1,1)\\
isLiveAt(c2,-1)\\
isLiveAt(c3,-1)\\
isLiveAt(c4,1)\\
\text{Cell}(c) \wedge \text{isLiveAt}(c,t) \wedge \text{hasSufficientNeighbours}(c,t) \wedge \text{isNegative}(t) \implies \text{isLiveAt}(c,1-t)\\
\text{Cell}(c) \wedge \text{isLiveAt}(c,t) \wedge \text{hasSufficientNeighbours}(c,t) \wedge \text{isPositive}(t) \implies \text{isLiveAt}(c,1+t)\\
\text{Cell}(c) \wedge \text{isLiveAt}(c,t) \wedge \text{hasInsufficientNeighbours}(c,t) \wedge \text{isNegative}(t) \implies \text{isLiveAt}(c,-1+t)\\
\text{Cell}(c) \wedge \text{isLiveAt}(c,t) \wedge \text{hasInsufficientNeighbours}(c,t) \wedge \text{isPositive}(t) \implies \text{isLiveAt}(c,-1-t)\\
\text{Cell}(c) \wedge \text{hasNeighbour}(c,c1) \wedge \text{isLiveAt}(c1,t) \wedge \cdots \wedge \text{isPositive}(t) \implies \text{hasSufficientNeighbours}(c,t)\\
\text{Cell}(c) \wedge \text{hasNeighbour}(c,c1) \wedge \text{isLiveAt}(c1,t) \wedge \cdots \wedge \text{isPositive}(t) \implies \text{hasInsufficientNeighbours}(c,t)
\end{gathered}
$$

Figure 11: Notion of local time

The predicate isLiveAt(cell , t) represents an alive cell when $t > 0$ and a dead cell when $t < 0$. The analogy here is that time is bidirectional; positive time represents the time in our world and negative time represents the time in afterlife. So, isLiveAt(cell , 3) indicates that the cell is alive at the iteration 3 and isLiveAt(cell , -4) means that the cell is alive at the iteration 4 in afterlife, in other words, the cell is dead at iteration 4 (in our world). A cell is alive in our world when it is dead in the afterlife and vice versa. This representation avoids the creation of new cells (as it is also prohibited in DL) and helps us to illustrate the notion of iteration/time. However, the above shown rules are not the only ones required to build such interesting ontology and the missing pieces that I have not intentionally mentioned is the formulas describing hasSufficientNeighbours(cell, t) and hasInsufficientNeighbours(cell, t); these are also updates through the process. In fact, hasSufficientNeighbours(cell , t) is updated to indicate that if cell has sufficient number of neighbours at iteration t to keep living (if live before) or turn to an alive cell (if dead before) at the consequent($t+1$) iteration. It is the opposite case for hasInsufficientNeighbours(cell , t) which is updated to indicate whether a cell has less or more than enough live neighbours at the iteration t.

Working on this project made me realize some facts about *difficulty of developing a good ontology*, *computation complexity of logic*, *integration of time to ontologies* and finally *bugs of Protégé software*. Developing a good ontology is not as easy as it may seem at the first glance, concepts and relations between them have to be carefully observed before discussing implementation details. While observing the true nature of a given problem, we have to also think about the assumptions that we make because our predictions lies behind our assumptions. Although DL is a decidable subset of FoL(First Order Logic), it is not as expressive which is an obvious trade-off. Finally, I gave Protégé a try to develop this particular project and encountered several problems along the way. I believe, there is a potential to develop the software and reduce the bugs.

Appendix

A Properties

Relation	Domain	Range	Type	Description
hasDivision	Fight	Division	functional	A fight has only one division
buys	NonOrgWorker	Fight	-	Any non organization worker can buy a fight
draws	Fighter	Fighter	regular	Fighters can draw
fights	Fighter	Fighter	symmetric, irreflexive	A fighter fights against another fighter
gives	Judge	FightScore	-	Judge gives the final score(s)
hasParticipant	Fight	Fighter	inverse (participatesAt)	Fight has a participating fighter
sells	Organization	Fight	regular	Organization sells fights
hasScore	Fight	FightScore	inverse (refersTo), inverse functional	Fight has a final score
isFor	FightScore	Fighter	functional	Fight score is given to a particular fighter
loses	Fighter	Fighter	inverse (wins)	Fighter can lose to another fighter
occupies	PhysicalEntities	Location	transitive	Physical entity occupies(located at) some location
organizes	Organizer	Fight	-	Organizer(s) organize(s) fight(s)
ownedBy	Organization	CEO	inverse (owns), functional	Organization is owned by a single CEO
owns	CEO	Organization	inverse (ownedBy), inverse functional	Organization is owned by a single CEO
participatesAt	Fighter	Fight	inverse (hasParticipant)	Fighter participates at a fight by fighting
pays	Organization	OrgWorker	inverse (worksAt)	Organization pays salary to the organization workers
refersTo	FightScore	Fight	inverse (hasScore), functional	Every fight score is given to a particular fight
sells	Organization	Fight	inverse functional	Organization sells fight(s)
wins	Fighter	Fight	-	Fighter can win a fight
worksAt	OrgWorker	Organization	inverse (pays)	Organization worker works at an organization

Table 8: Object properties

B Individuals

Individual	Asserted	Inferred
Org	Organization sells Fight1 sells Fight2 occupies LocBaku hasName UFCA	occupies LocAz ownedBy OrgCEO2 ownedBy OrgCEO paysBonus OrgFighter2
OrgCEO	Person owns Org hasName Nihad	CEO sameAs OrgCEO2
OrgCEO2	owns Org	CEO sameAs OrgCEO hasName Nihad
OrgFighter1	participatesAt Fight1 hasGender male hasName Telman	Fighter fights OrgFighter2 loses Fight1 weighs 52.5
OrgFighter2	Fighter hasName Bashir	fights OrgFighter1 participatesAt Fight1 wins Fight1 hasGender male weighs 52.5
OrgFighter3	Fighter hasName Daniz weighs 65.8 hasGender female warBorn 1990-01-01T00:00:00	draws Fight2 fights OrgFighter4 participatesAt Fight2
OrgFighter4	Fighter hasName Narmin	draws Fight2 fight OrgFighter3 participatesAt Fight2 hasGender female weighs 65.8
Fight1	hasDivision D1 hasGivenScore FightScore1_2 hasParticipant OrgFighter2 costs 25.0	Fight hasGivenScore FightResult1_1 hasParticipant OrgFighter1
Fight2	Fight hasGivenScore FightScore2_1 hasGivenScore FightScore2_2 hasParticipant OrgFighter3 hasParticipant OrgFighter4 costs 18.0	
FightScore1_1	FightScore isFor OrgFighter1 refersTo Fight1 hasScore 15	
FightScore1_2	FighScore isFor OrgFighter2 hasScore 17	refersTo Fight1
FightScore2_1	FighScore isFor OrgFighter 3 hasScore 10	refersTo Fight2
FightScore2_2	FighScore isFor OrgFighter4 hasScore 10	refersTo Fight2
LocBaku	occupies LocAz hasName Baku	Location

Table 9: Individuals

C Graph by Ontograf

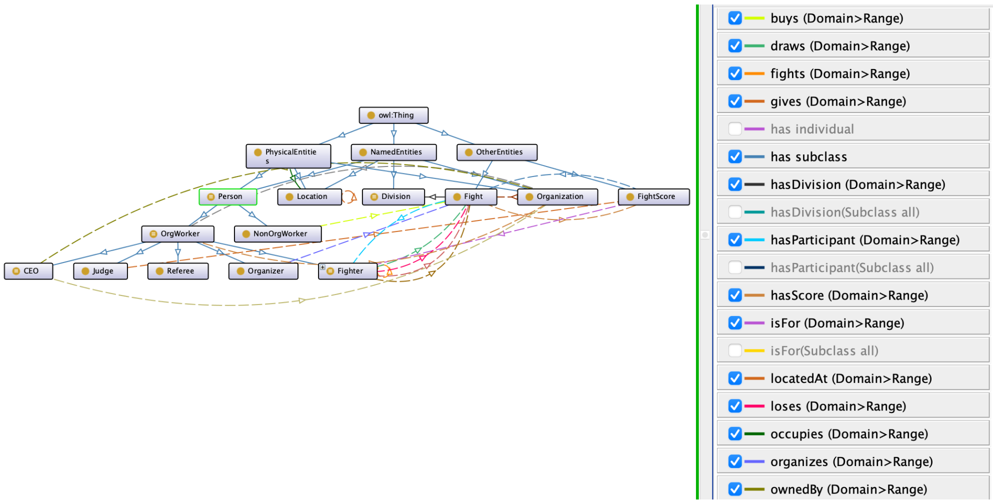


Figure 12: Complete graph with arc labels