OFFICIAL

Extending IES4

I E S 4
information exchange standard er…four

Applicable to all minor versions of IES4

Introduction

This document provides guidance on how to extend IES4 for specific local needs.

You cannot just add an orphaned concept into such a formal ontology, the concept must extend an existing concept in IES. We do this by finding the closest, similar concept. This normally is a more generalised concept of the one you seek to add. It is from this concept we make our extension from. For example, if you wanted to add PassengerShip, the closest concept in IES is the class ies:Ship. Similarly, if you wanted to add a specific relation of motherOf, you would extend from the relationship, ies:isParentOf.

IES utilises the following RDF-Schema relationships for extensions:

- rdfs:subClassOf for extending classes
- rdfs:subPropertyOf for extending attributes or relationships

Through-out this document, we will illustrate how to create extensions using a mix of UML diagrams and RDF triples. The triples presented will utilise the following prefixes:

ies: – referring to things in the IES ontology

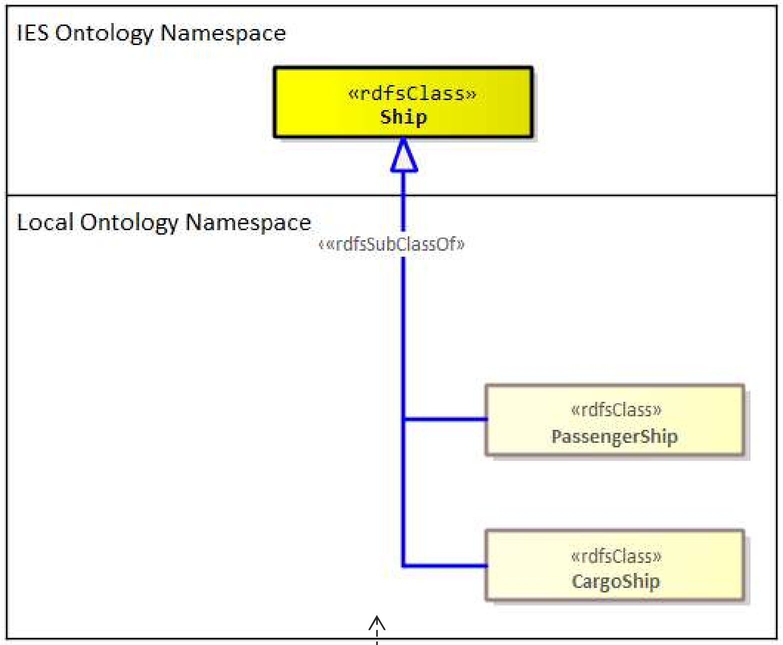
ont: – referring to things in an example, local ontology

data: – referring to things in an example, instance dataset

Simple extensions: Defining new local classes

Let’s say there is a need to extend IES to introduce two new types of Ship: a **PassengerShip** and a **CargoShip**. As discussed in the introduction, first, we want to search the IES ontology to identify the closest class that matches our requirement. Here, ies:Ship is the obvious choice.

Then in our local ontology namespace we add these new classes using the rdfs:subClassOf relation as illustrated here. These new classes will therefore inherit all attributes and relationships and are already associated to ies:Ship.



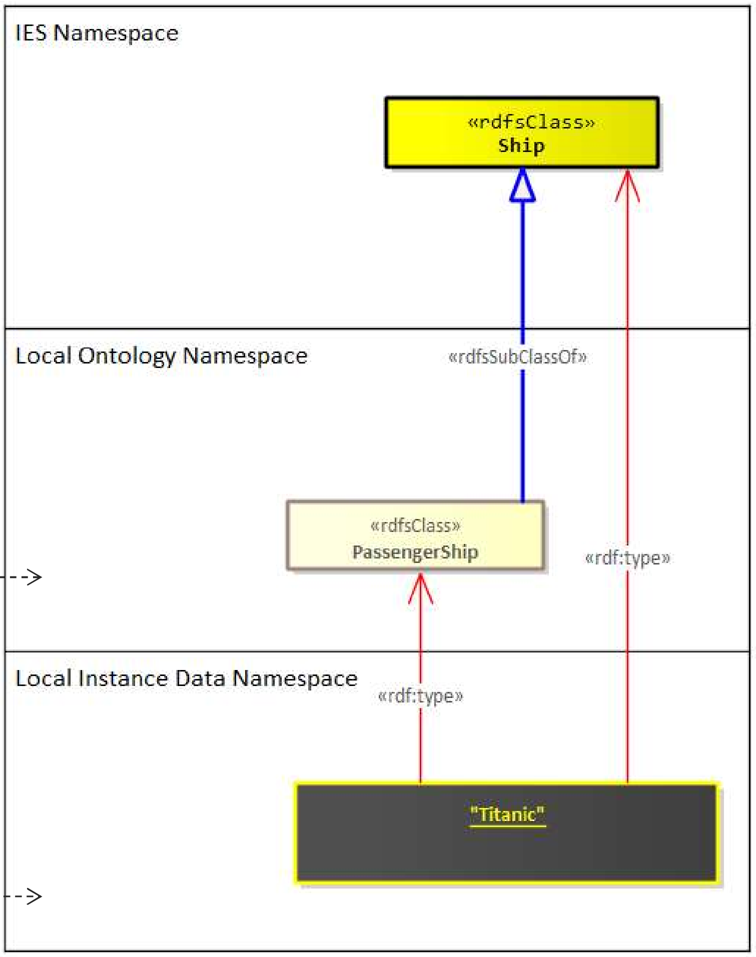
```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ies:  <http://ies.data.gov.uk/ontology/ies4#>.
@prefix ont:  <http://example.com/local-ontology#> .


ont:PassengerShip rdfs:subClassOf ies:Ship .
ont:CargoShip     rdfs:subClassOf ies:Ship .
```

Simple extensions: Using new local classes

Now that we have our new extensions, we can use them in our data. We follow the normal approach to instantiation using rdf:type (or the shorthand “a”) i.e., we make the instance a type of one of our local classes. However, in addition, we also make this instance a type of the nearest class in IES. This caters for consumers which may not have access to our local ontology, so at very least, they can understand what sort of IES thing our instance is meant to be. For example, as consumer, you may have a query for getting all instances of ies:Ship. If you were then given data that only typed things as either ont:PassengerShip or ont:CargoShip, your query would miss those ship instances. By also providing the type of the nearest IES class, we allow for such queries to continue to work i.e., we don’t miss any ships and only miss the extra subtype detail. This provides a form of backward compatibility, giving consumers flexibility to update their queries in their own time without the fear of them missing out on the information they care about.

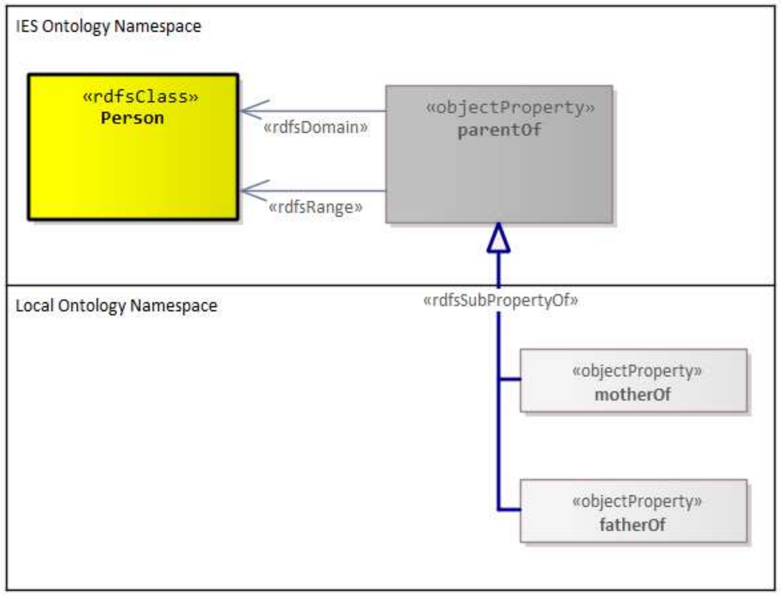
```
ont:PassengerShip  rdfs:subClassOf  ies:Ship .
ont:CargoShip      rdfs:subClassOf  ies:Ship .
```

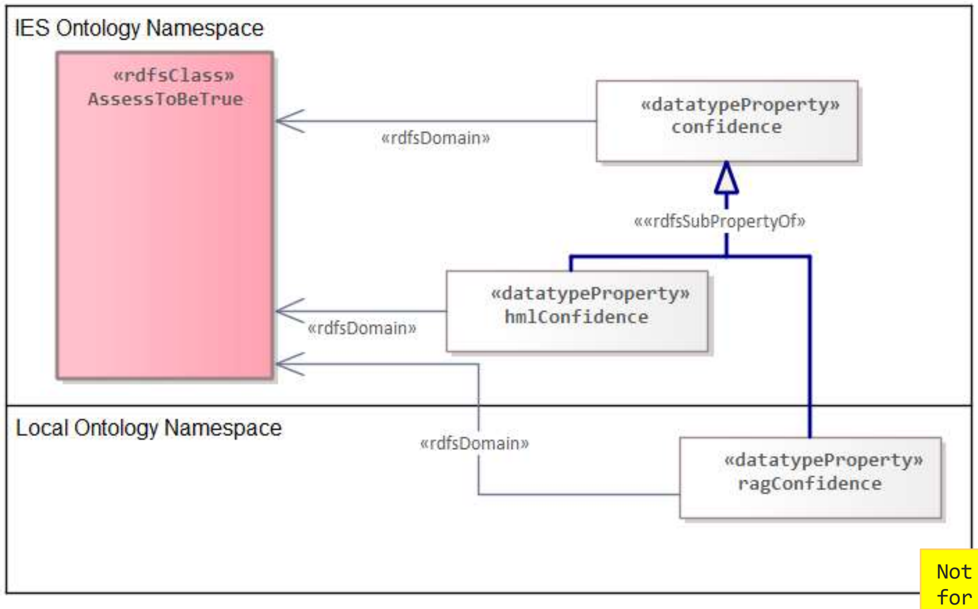
```
data:Titanic  a  ont:PassengerShip .
data:Titanic  a  ies:Ship .
```



Simple extensions: Defining new attributes and relationships

The same approach as already specified applies here at first; find the closest attribute or relationship in IES. But then the extension is created using rdfs:subPropertyOf rather than rdfs:subClassOf. Sometimes you might need to articulate the rdfs:domain (for attributes and relationships) and rdfs:range (only for relationships) of the extension if you want to narrow down it's association to specific classes. As with extending classes and for the same reasons, you MUST also articulate the nearest equivalent in IES. See the examples below.




IES Ontology Namespace
«rdfsClass» AssessToBeTrue
«rdfsDomain»
«datatypeProperty» confidence
««rdfsSubPropertyOf»
«datatypeProperty» hmlConfidence
«rdfsDomain»
Local Ontology Namespace
«rdfsDomain»
«datatypeProperty» ragConfidence

```
ont:motherOf  rdfs:subPropertyOf  ies:parentOf .
ont:fatherOf  rdfs:subPropertyOf  ies:parentOf .
```

```
data:person_1 ont:motherOf data:person_2 .
data:person_1 ies:parentOf data:person_2 .
```

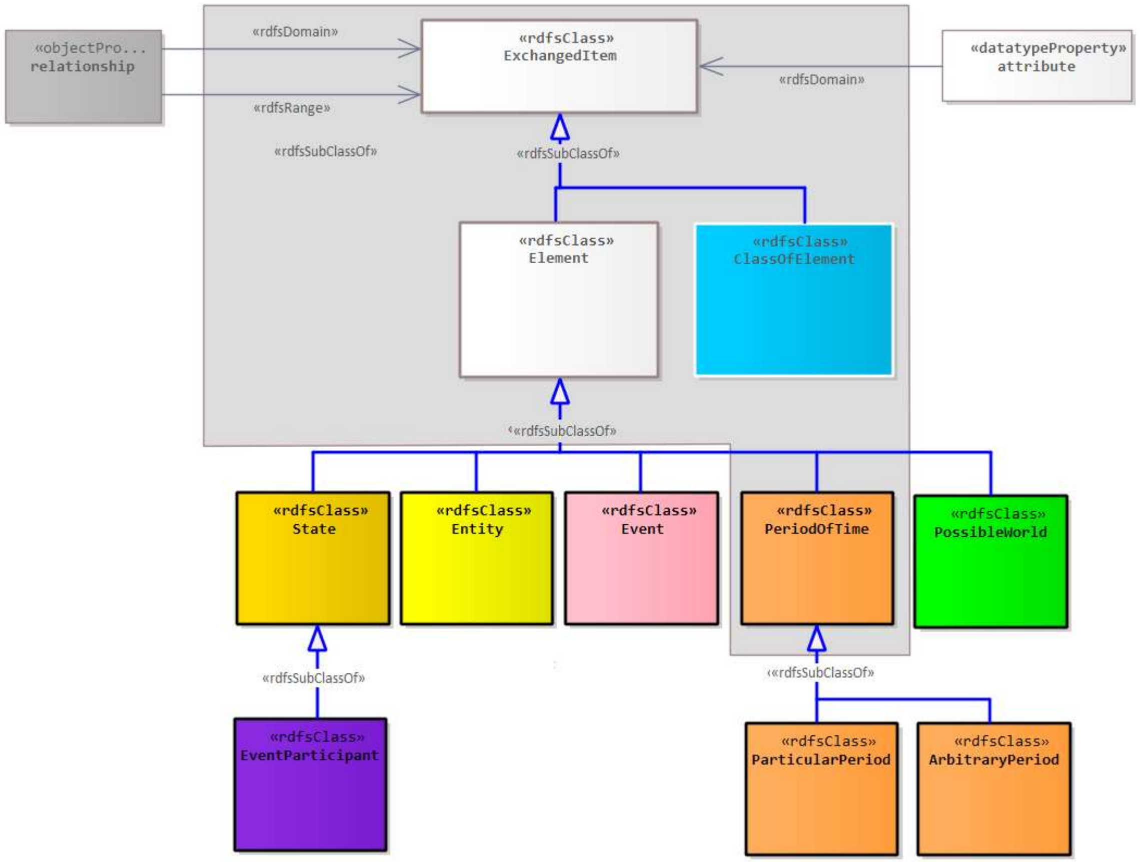
```
ont:ragConfidence  rdfs:subPropertyOf    ies:confidence .
ont:ragConfidence  rdfs:domain           ies:AssessToBeTrue . <----
```

```
data:assessment_1  ont:ragConfidence     "GREEN"^^xsd:string .
data:assessment_1  ies:confidence        "GREEN"^^xsd:string .
```

Not strictly required for this example due to inheritance. However, it is done in the IES standard to avoid ambiguity.

Permissible extension boundary

IES has a small number of concepts at the top of its hierarchy. It not permissible to extend the broader concepts found above this level e.g. ExchangeItem and Element. This ensures, at the very least, consumers of data that use extensions can at least understand what key concepts are being exchanged. The figure here shows those concepts (within the grey boundary) which cannot be extended.


«objectPro... relationship
«rdfsDomain»
«rdfsRange»
«rdfsClass» ExchangedItem
«rdfsDomain»
«datatypeProperty» attribute
«rdfsSubClassOf»
«rdfsSubClassOf»
«rdfsClass» Element
«rdfsClass» ClassOfElement
«rdfsSubClassOf»
«rdfsClass» State
«rdfsClass» Entity
«rdfsClass» Event
«rdfsClass» PeriodOfTime
«rdfsClass» PossibleWorld
«rdfsSubClassOf»
«rdfsSubClassOf»
«rdfsClass» EventParticipant
«rdfsClass» ParticularPeriod
«rdfsClass» ArbitraryPeriod

Finding the right level to extend from

In the examples presented thus far, it has been evident where to make an extension from. However, there are times where this is less obvious. Here are some questions you can ask about your new concept to help find that right level.

What kind of IES thing is it?	Does it have spatiotemporal extent? i.e. an Element. Or is it a set/class. Is it more a representation, identifier or measure? If it's an element, which of the high-level types of element is it? Is it a single entity or does it involve multiple entities i.e. an event? If your concept doesn't apply to any of the above, then a relationship or attribute could be considered.* However, these are last resort options and should only really be considered if there is an existing relationship or attribute that is very close to the extension you seek.
Which IES pattern is it most like?	The IES standard presents the model through a series of diagrams which represent major, reusable patterns. Seek the titles of patterns that have some relevance to the concept you want to add.
Which names and definitions encapsulate my concept?	Use the definitions of IES classes, relationships and attributes to guide you. The definition should really cover as close to 100% of what your concept talks about, even if it's covered in a vague sense. If there is something in the definition that clearly doesn't apply to your new concept, then it is most likely not suitable. Discounting what it's not is a useful approach. Sometimes you might also want to consider things from a set theory point-of-view. Consider your concept as a set and ask; will all the members of my new set also be found within the proposed super set?
Do the relationships and attributes also apply to my concept?	For class extensions, check if the relationships and attributes that hang off the proposed super class also apply to your new class.

You might find that you ask these questions of your concept in different orders, and you might have to repeatedly ask such questions as you work your way across the model and up (or down) potential hierarchies.

** Relationships (apart from type, subtype, isPartOf) and attributes in IES are normally shortcuts for more elaborate 4D structures. Where you see these in the model, a pragmatic choice has been made to provide a shortcut for such elaborate structures. As a result, they are **last resort** options when considering a level to extend from.*

Finding the right level: an example

Let’s take an example of adding a less obvious concept like a *Peaceful Protest*. Let’s use the questions from the previous slide to help us:

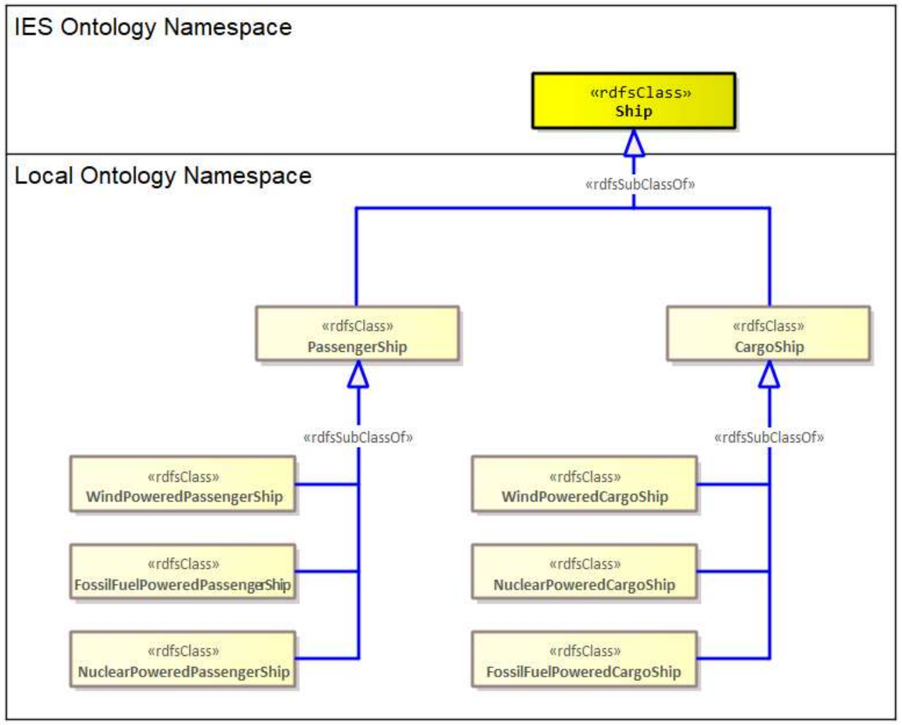
What kind of IES thing is it?	*Peaceful Protest have spatiotemporal extent and involves more than one entity; so, it’s a type of event.*
Which IES pattern is it most like?	*The “Disagreement and War” pattern sounds somewhat applicable, so let’s have look at that pattern.*
Which names and definitions encapsulate my concept?	*This pattern includes the events ies:War and ies:OperationalEvent. The former, from its name alone seems like a bit of a stretch for something peaceful and the definition of the latter implies military or national security actors. ies:Disagreement on the other hand seems more like it. However, when analysing our data, it suggests that a subset of our protests either aim to raise awareness about a cause, advocate for positive changes or show solidarity with a particular group. Therefore, not all members of our Peaceful Protest set fit nicely within the ies:Disagreement set.*
Which IES pattern is it most like?	*Nothing else in the “Disagreement and War” pattern seems to work for our Peaceful Protest. Let’s seek another, how about “Attendance”? A protest seems vaguely like some sort of attendance of people.*
Which names and definitions encapsulate my concept?	*The pattern includes an event called ies:Meeting, at first glance, this seems appropriate. But the definition suggests that participants in this event communicate with one and other. That is probably true for most protests but what if there is a silent protest? This line of thinking might cause me to go up a level to the superclass of Meeting which is ies:CoLocation. Now, in my mind, Peaceful Protest fits completely within the set of CoLocations.*
Do the relationships and attributes also apply to my concept?	*Let’s double check the relationships and attributes associated to CoLocation also apply to Peaceful Protest. CoLocation has an associated event participant called ies:Presence which works as the participant states of persons involved in the protest. However, when looking at the attributes I notice the ies:Meeting which I had discounted, has an attribute called ies:hasTheme. This might in fact be quite useful as I have a requirement to call out the reason behind the protest. As a result, I might tone down my pedantry re. silent protests in favor of the benefits ies:Meeting provides.*

As you can see, it's quite common to oscillate between various perspectives in search of the right level. Sometimes a balance needs to be struck between being opinionated and being pragmatic.

Complex extensions: The wrong way

Earlier on in this pack, we went through a simple extension example where we added two subtypes of ies:Ship to our local ontology: PassengerShip and CargoShip. Imagine now, we have an additional requirement to include type information about how they are powered which apply to both subtypes.

One solution is to develop a class hierarchy that is "nested" (shown to the right). However, this introduces a lot of duplication. The compounding of the types in the subclasses will cause headaches when you want to query for specific facets* of a ship e.g., we only want fossil fueled powered ships returned by our query.


IES Ontology Namespace
«rdfsClass» Ship
Local Ontology Namespace
«rdfsSubClassOf»
«rdfsClass» PassengerShip
«rdfsClass» CargoShip
«rdfsSubClassOf»
«rdfsSubClassOf»
«rdfsClass» WindPoweredPassengerShip
«rdfsClass» FossilFuelPoweredPassengerShip
«rdfsClass» NuclearPoweredPassengerShip
«rdfsClass» WindPoweredCargoShip
«rdfsClass» NuclearPoweredCargoShip
«rdfsClass» FossilFuelPoweredCargoShip

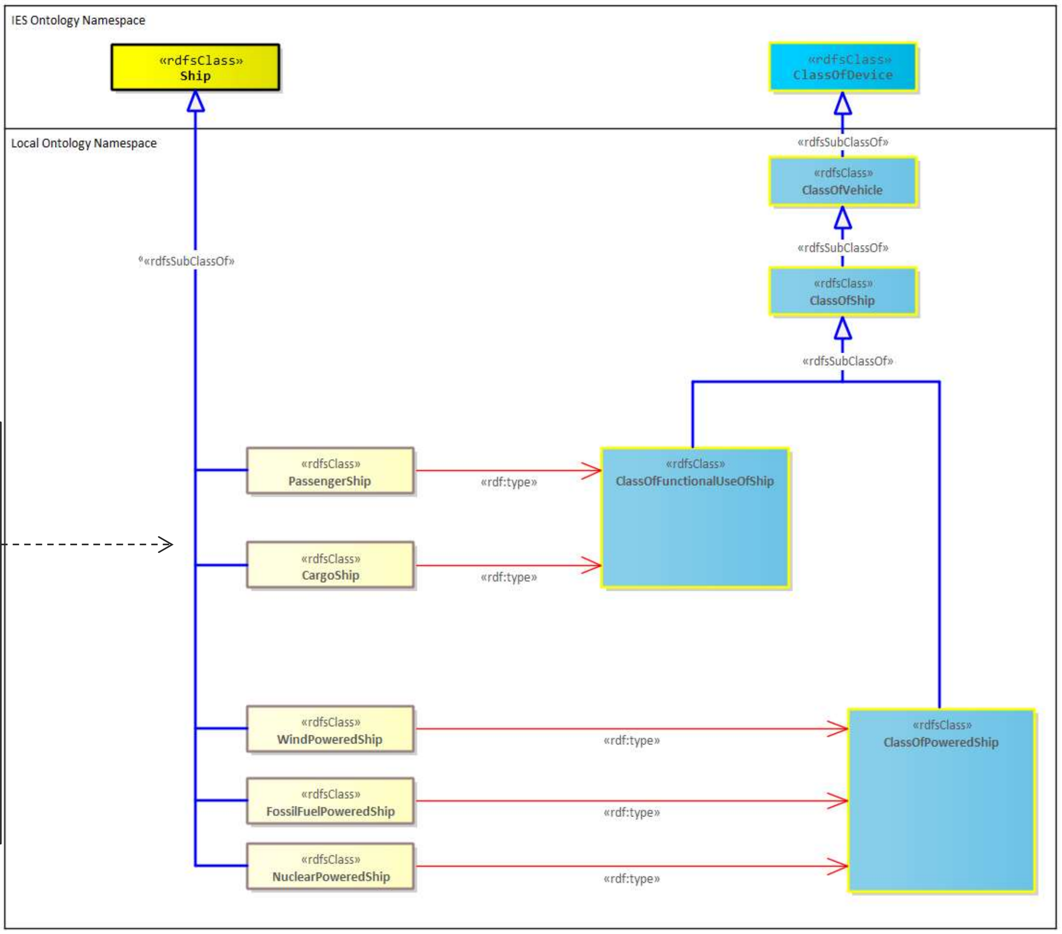
*A faceted classification system uses a set of semantically cohesive categories that are combined as needed to create an expression of a concept.

Complex extensions: The right way

A faceted approach can be developed in IES using powersets. We create classes in an atomic way and then group them using a class one level up aka. the powerset. Here our powersets are found in a hierarchy that extends from ies:ClassOfDevice.

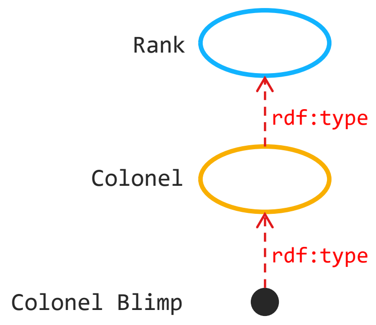
This approach helps create a flatter, easier to maintain structure that is easier to query. Moreover, this removes the need for compounding types.

```
ont:ClassOfVehicle              rdfs:subClassOf         ies:ClassOfDevice .
ont:ClassOfShip                 rdfs:subClassOf         ont:ClassOfVehicle .
ont:ClassOfFunctionalUseOfShip  rdfs:subClassOf         ont:ClassOfShip
ont:ClassOfPoweredShip          rdfs:subClassOf         ont:ClassOfShip .

ont:PassengerShip               rdfs:subClassOf         ies:Ship .
ont:PassengerShip               a                       ont:ClassOfFunctionalUseOfShip .

ont:CargoShip                   rdfs:subClassOf         ies:Ship .
ont:CargoShip                   a                       ont:ClassOfFunctionalUseOfShip .

ont:WindPoweredShip             rdfs:subClassOf         ies:Ship .
ont:WindPoweredShip             a                       ont:ClassOfPoweredShip .

ont:FossilFuelPoweredShip       rdfs:subClassOf         ies:Ship .
ont:FossilFuelPoweredShip       a                       ont:ClassOfPoweredShip .

ont:NuclearPoweredShip          rdfs:subClassOf         ies:Ship .
ont:NuclearPoweredShip          a                       ont:ClassOfPoweredShip .
```


IES Ontology Namespace
«rdfsClass» Ship
«rdfsClass» ClassOfDevice
Local Ontology Namespace
«rdfsSubClassOf»
«rdfsClass» ClassOfVehicle
«rdfsSubClassOf»
«rdfsSubClassOf»
«rdfsClass» ClassOfShip
«rdfsSubClassOf»
«rdfsClass» PassengerShip
«rdf:type»
«rdfsClass» ClassOfFunctionalUseOfShip
«rdfsClass» CargoShip
«rdf:type»
«rdfsClass» WindPoweredShip
«rdf:type»
«rdfsClass» ClassOfPoweredShip
«rdfsClass» FossilFuelPoweredShip
«rdf:type»
«rdfsClass» NuclearPoweredShip
«rdf:type»

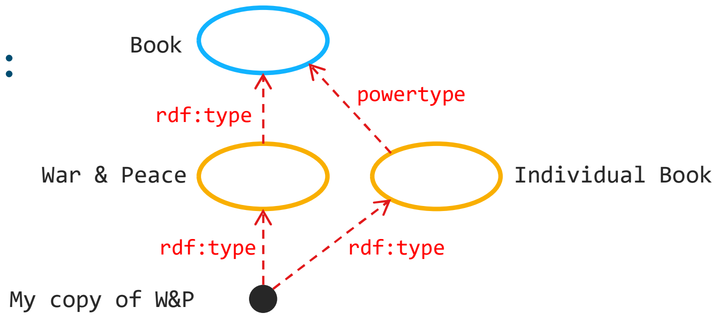
A reminder: powertypes

To be able to talk about classes that are themselves members of other classes, we need to be able to "push" up to the next type level. BORO ontologies such as IES have no limit to the number of layers you can go up. Each layer is connected to the previous by a rdf:type or a ies:powertype relationship. Consider the following examples:



In this example, Colonel Blimp is an instance of (rdf:type) the class Colonel. Colonel is an instance of the class Rank. Colonel Blimp is *not* an instance of Rank though. rdf:type is therefore not transitive.

We do the same thing for documents:



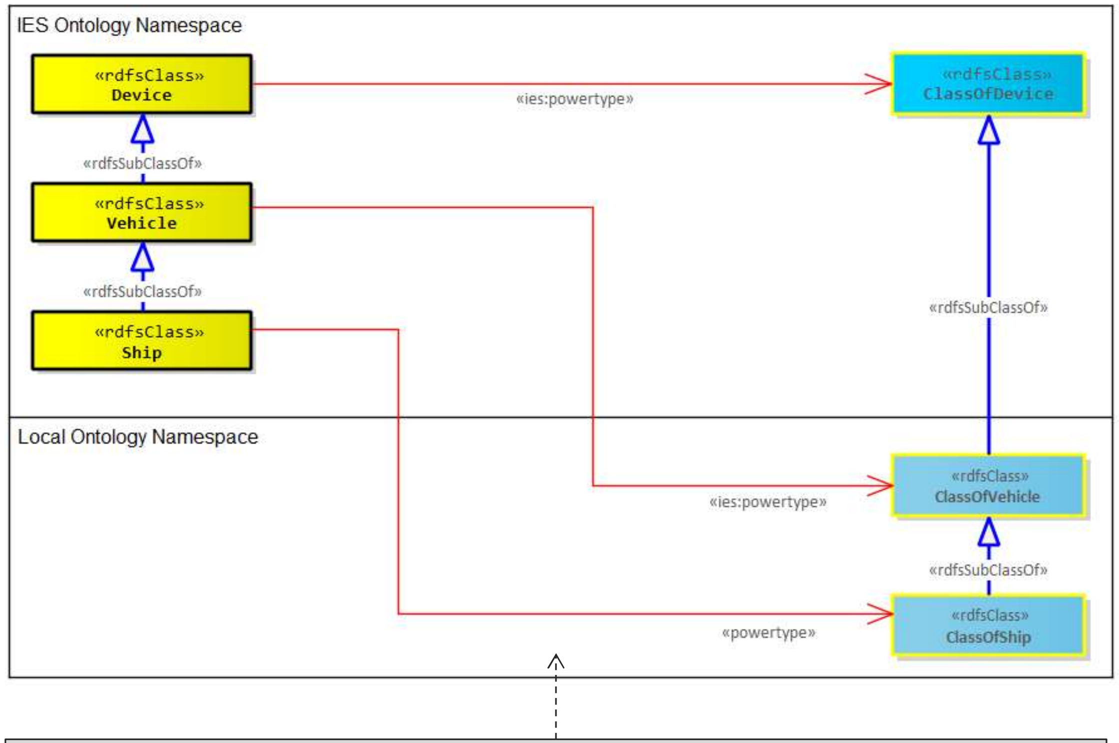
The mechanism used for stepping up the type levels in IES is the ies:powertype relationship. It relates a Class to another class whose members are all possible subtypes of that Class. This is a bit of logical plumbing in IES and may not be of interest to all users. If you are geek enough to be interested in this stuff, start by looking up "powerset" on Wikipedia and then look at Cantor's theorem

Complex extension: Additional “plumbing”

Introducing new powersets as we did with the hierarchy extending from ies:ClassOfDevice, sometimes requires us do some additional ontology pluming. We need to explicitly declare that our new ont:ClassOfVehicle and ont:ClassOfShip are powertypes of the equivalent elements currently in IES. Aka:

- ALL subtypes of ies:Vehicle are members of ont:ClassOfVehicle and;
- ALL subtypes of ies:Ship are members of ont:ClassOfShip.

Note, we don’t need this for the other powersets we introduced - ont:ClassOfFunctionalUseOfShip and ont:ClassOfPoweredShip. This is because we have not introduced any extensions that have subtypes which are all members of these powersets.

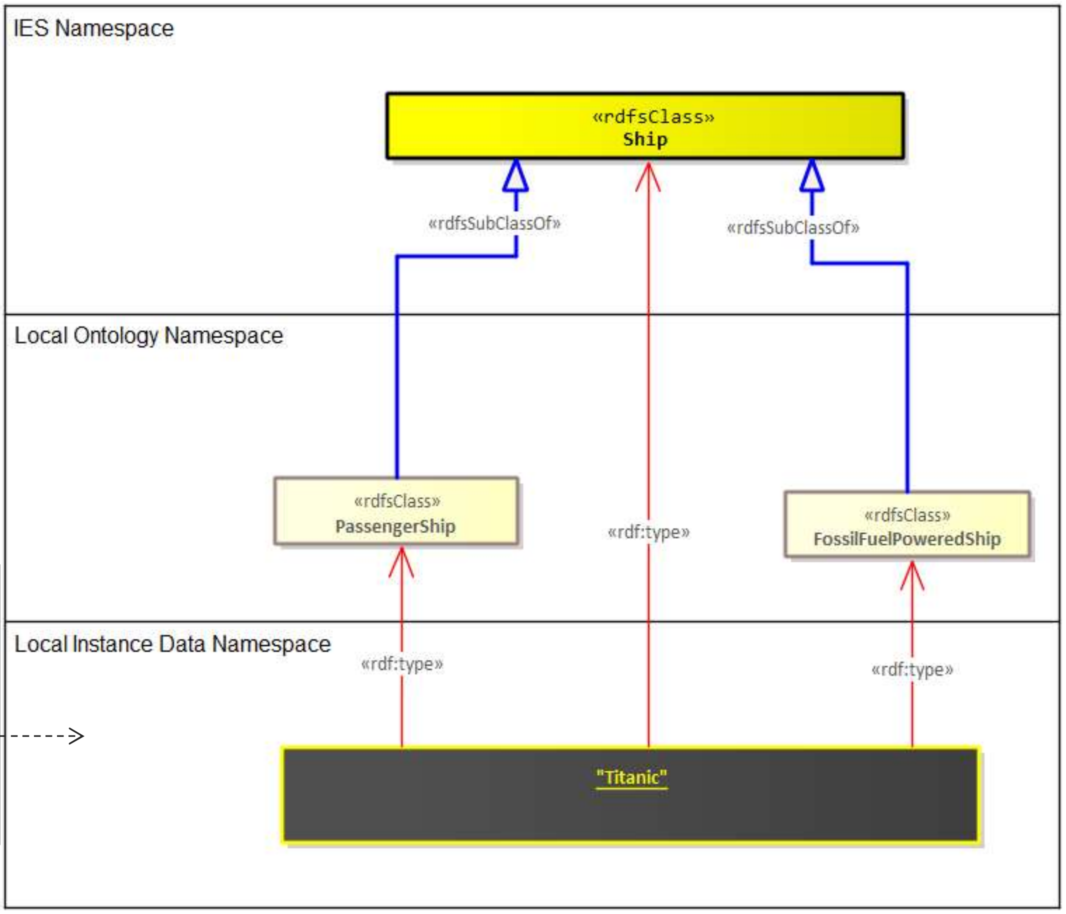


```
ies:Vehicle          ies:powertype          ont:ClassOfVehicle .
ies:Ship             ies:powertype          ont:ClassOfShip .
```

Complex extensions: Using new local classes

Because of the faceted approach we have taken, our ship instances will now have two types as shown here with the Titanic. And as is required to cater for those without access to our ontology extensions, a third type is needed to state the nearest IES equivalent.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ies:  <http://ies.data.gov.uk/ontology/ies4#>.
@prefix ont:  <http://example.com/local-ontology#> .
@prefix data: <http://example.com/local-data#> .


data:Titanic  a  ont:PassengerShip .
data:Titanic  a  ont:FossilFuelPoweredShip .
data:Titanic  a  ies:Ship .
```



Specific guidance for extending Entities

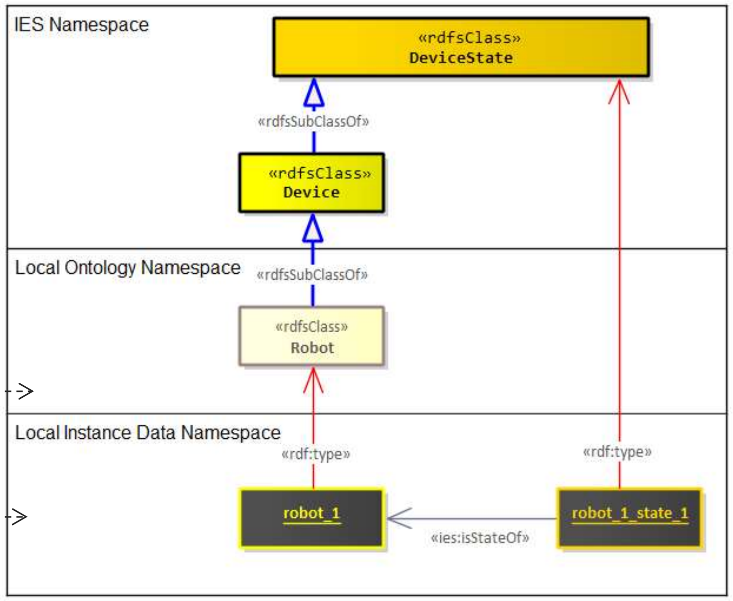
There is a bit of a “chicken-and-egg” situation that arises when considering extensions to an Entity. Concrete types of Entity (such as Person, Organisation, Vehicle) are effectively special types of their respective states (PersonState, OrganisationState, VehicleState). An Entity can be viewed as a maximal state, covering the entire duration of the existence of an Entity.

The chicken-and-egg situation arises from questions about which should you extend first, an appropriate State or Entity. For example, if you wanted to add a new subclass of ies:Device (say ont:Robot), do you create an extension of ies:DeviceState first (say ont:RobotState) and then make the new entity a subclass of the new state? i.e., *Robot subClassOf RobotState*.

The rule is, if you are adding a new Entity where its states would have no additional special relationships and/or attributes compared to its superclass’s states, then there is no need to create a specific state for this entity at all. Just extend directly from the appropriate Entity as per the illustrated example and utilise the superclass’s associated state. If there are special relationships / attributes, then extend the state first. As mentioned earlier in the document, adding new relationships and attributes is a last-resort move, therefore the latter form of extension, is a rare case.

```
ont:Robot                  rdfs:subClassOf     ies:Device .
```

```
data:robot_1               a                   ont:Robot .
data:robot_1_state_1       a                   ies:DeviceState .
data:robot_1_state_1       ies:isStateOf       data:robot_1.
```



Extension naming convention

Below is a set of mandatory (MUST and MUST NOT) rules and recommended (SHOULD and SHOULD NOT) rules for naming your extensions. These are based on recommendations made by Matthew West in his book “Developing High Quality Data Models”.

All Extensions	• MUST be based on the real-world thing it is representing. • MUST NOT be based on the data record which is itself based on the real-world thing. • MUST NOT use special characters or whitespace. • SHOULD use names(s) that suggest the definition. • SHOULD avoid using the same word(s) as used elsewhere in IES or your local ontology. If that is hard to avoid, reduce confusion by adding a qualifying term to make it unambiguous. • SHOULD avoid abbreviations: e.g., GPAppointment should be GeneralPractitonerAppointment. • SHOULD use a similar naming pattern and/or structure as the superclass or superProperty.
Extensions to Elements	• MUST use PascalCase. • SHOULD be a noun.
Extensions to ClassOfElements	• MUST use PascalCase. • SHOULD be a noun. • SHOULD keep to the convention of being prefixed with “ClassOf”.
Extensions to relationships or attributes	• MUST use camelCase. • SHOULD either: • provide some connective text that reads as a sentence between the domain and range for a relationship or the domain and the literal for an attribute, or; • named in terms of the role it plays in respect to the domain; Either of these might appear more natural in a particular circumstance.