- a `ReductionOMS`, applying a reduction (given by an `Reduction`) to an OMS (see use cases 7.9, 7.10 and 7.11 and Appendix M.9 and M.10 for examples);
- a `ExtractionOMS`, applying a module extraction operator (given by an `Extraction`) to an OMS (see use case 7.4 for an example);
- a `QualifiedOMS`, which is an OMS qualified with the OMS language that is used to express it.

Moreover, annex L informatively introduces `Applications`, which apply a substitution to an OMS.

A `ConservativityStrength` specifies additional relations that may hold between an OMS and its extension (or union with other OMS), like conservative or definitional extension. The rationale is that the extension should not have impact on the original OMS that is being extended.

An OMS definition `OMSDefinition` names an OMS. It can be optionally marked as inconsistent, consistent, monomorphic or having a unique model using `ConservativityStrength`. More precisely, `'consequence-conservative'` here requires the OMS to have only tautologies as signature-free logical consequences, while `'notconsequence-conservative'` expresses that this is not the case. `'model-conservative'` requires satisfiability of the OMS, `'not-model-conservative'` its unsatisfiability. `'definitional'` expresses that the OMS has a unique model (see Appendix M.5 for an example); this may be interesting for characterizing OMS (e.g. returned by model finders) that are used to describe single models.

The DOL metamodel for extension OMS is shown in Fig. 9.6. `ExtendingOMS` is a subclass of `OMS`, containing those OMS that may be used to extend a given OMS within an `ExtensionOMS`. An `ExtendingOMS` can be one of the following:

- a basic OMS `BasicOMS` written inline, in a conforming serialization of a conforming OMS language (which is defined outside this standard; practically every example uses basic OMS)[16];
- a reference (through an IRI) to an OMS (`OMSReference`, many examples illustrate this); or
- a `RelativeClosureOMS`, applying a closure operator to a basic OMS or OMS reference (these two are hence joined into `ClosableOMS`). A closure forces the subsequently declared non-logical symbols to be interpreted in a minimal or maximal way, while the non-logical symbols declared in the local environment are fixed.[17] Variants of closure are minimization, maximization, freeness (minimizing also data sets and equalities on these, which enables the inductive definition of relations and datatypes), and cofreeness (enabling the coinductive definition of relations and datatypes). See Annex M.6 for examples of the former two, and Annex M.11 for examples of the latter two.

Recall that the local environment is the OMS built from all previously-declared symbols and axioms.

Using `ExtendingOMS`, extensions of an OMS with an `ExtendingOMS` can be built. The latter can optionally be named and/or marked as conservative, monomorphic, definitional, weakly definitional or implied (using a `ConservativityStrength`, see clause 4.3 for details). Note that an `ExtendingOMS` used in an extension must not be an `OMSReference`.

Furthermore, OMS can be constructed using

- closures of an OMS with a `Closure`. This is similar to a `RelativeClosureOMS`, but the non-logical symbols to be minimized/maximized and to be varied are explicitly declared here (while a `RelativeClosureOMS` takes the local environment to be fixed, i.e. not varied);
- a translation `OMSTranslation` of an OMS into a different signature or OMS language. The former is done using a `SymbolMap`, specifying a map of symbols to symbols. The latter is done using an OMS language translation `OMSLanguageTranslation` can be either specified by its name, or be inferred as the default translation to a given target (the source will be inferred as the OMS language of the current OMS);
- a `Reduction` of an OMS to a smaller signature and/or less expressive logic (that is, some non-logical symbols and/or some parts of the model structure are hidden, but the semantic effect of sentences involving these is kept). The former is done using a `SymbolList`, which is a list of non-logical symbols that are to be hidden. The latter uses an `OMSLanguageTranslation` denoting a logic projection that is used as logic reduction to a less expressive OMS language.

---

[16] In this place, any OMS in a conforming serialization of a conforming OMS language is permitted. However, DOL's module sublanguage should be used instead of the module sublanguage of the respective conforming OMS language; e.g. DOL's OMS reference and extension construct should be preferred over OWL's import construct.

[17] Note that if applied to algebraic signatures (sorts and operation symbols), minimization can be used to express reachability (i.e. term-generatedness) of algebraic (first-order) models.

```
spec Bag =
  sort Elem
  then free {
     sort Bag
     ops mt:Bag;
         __union__:Bag*Bag->Bag, assoc, comm, unit mt
           }
end


spec Bag_variant =
 sort Elem
then minimize { %% select term generated models
 sort Bag
 ops mt:Bag;
   __union__:Bag*Bag->Bag, assoc, comm, unit mt
    }
then
 pred __elem__ : Elem * Bag
 forall x:Elem; b1,b2:Bag
 . not x elem mt
 . x elem (b1 union b2) <=> (x elem b1 / x elem b2)
 . b1=b2 <=> forall y:Elem . (y elem b1 <=> y elem b2) %(extensionality)%
    %% term generatedness and extensionality together
    %% select the standard bag model
end

equivalence e : Bag <-> Bag_variant = {}
end


spec Stream =
  sort Elem
  then cofree {
     sort Stream
     ops head:Stream->Elem;
         tail:Stream->Stream
           }
end

spec Finite =
  sort Elem
  free type Nat ::= 0 | suc(Nat)
  pred __<__ : Nat * Nat
  forall m,n:Nat
  . 0 < suc(n)
  . not n < 0
  . suc(m) < suc(n) <=> m < n
  op f: Nat ->? Elem
  . forall x:Elem . exists n:Nat . f(n)=x           %(f_surjective)%
  . exists n:Nat . forall m:Nat . def f(m) => m<n   %(f_bounded)%
  reveal Elem
end
```

M.12 Queries

```
%prefix( lang:  <http://purl.net/DOL/languages/> )%
library MyQuery
language lang:CASL
spec Person =
  sort s
  pred Person:s
```