— 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]. Note that a basic OMS used inside a DOL document may not use any of the DOL keywords (see clause 9.8.1); otherwise, it needs to be enclosed in curly braces[17];
— 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. 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] This restriction applies to DOL documents only, not to native documents.

9.5.2 Concrete Syntax

While in most cases the translation from concrete to abstract syntax is obvious (the structure is largely the same),

— both `%satisfiable`, `%cons` and `%mcons` are translated to `model-conservative`,

— both `%consistent` and `%ccons` are translated to `consequence-conservative`,

— both `%unsatisfiable` and `%notmcons` are translated to `not-model-conservative`,

— both `%inconsistent` and `%notccons` are translated to `not-consequence-conservative`,

— moreover, both `closed-world` and `minimize` are translated to `minimize`.

Note that the MOF abstract syntax subsumes all these elements except from those in the last line under the enumeration class `ConservativityStrength`. Not all elements of the enumeration can be used at any position; the corresponding restrictions are expressed as OCL constraints. By contrast, the concrete syntax features a more fine-grained structure of non-terminals (`Conservative`, `ConservativityStrength` and `ExtConservativityStrength`) in order to express the same constraints via the EBNF grammar.

```
BasicOMS            ::= <language and serialization specific>
ClosableOMS         ::= BasicOMS |  '{' BasicOMS '}' |  OMSRef [ImportName]
ExtendingOMS        ::= ClosableOMS | RelativeClosureOMS
RelativeClosureOMS  ::= ClosureType '{' ClosableOMS '}'
OMS                 ::= ExtendingOMS
                      | OMS Closure
                      | OMS OMSTranslation
                      | OMS Reduction
                      | OMS Extraction
                      | OMS Approximation
                      | OMS Filtering
                      | OMS 'and' [ConservativityStrength] OMS
                      | OMS 'then' ExtensionOMS
                      | Qualification* ':' GroupOMS
                      | 'combine' NetworkElements [ExcludeExtensions]
                      | GroupOMS
Closure             ::= ClosureType CircMin [CircVars]
ClosureType         ::= 'minimize'
                      | 'closed-world'
                      | 'maximize'
                      | 'free'
                      | 'cofree'
CircMin             ::= Symbol Symbol*
CircVars            ::= 'vars' Symbol Symbol*
GroupOMS            ::= '{' OMS '}' | OMSRef
OMSTranslation      ::= 'with' LanguageTranslation* SymbolMap
                      | 'with' LanguageTranslation+
LanguageTranslation ::= 'translation' OMSLanguageTranslation
Reduction           ::= 'hide' LogicReduction* SymbolList
                      | 'hide' LogicReduction+
                      | 'reveal' SymbolList
LogicReduction      ::= 'along' OMSLanguageTranslation
SymbolList          ::= Symbol  ',' Symbol *
SymbolMap           ::= GeneralSymbolMapItem  ',' GeneralSymbolMapItem *
Extraction          ::= 'extract' InterfaceSignature
                      | 'remove' InterfaceSignature
Approximation       ::= 'forget' InterfaceSignature ['keep' LogicRef]
                      | 'keep' InterfaceSignature ['keep' LogicRef]
                      | 'keep' LogicRef
Filtering           ::= RemovalKind BasicOMSOrSymbolList
RemovalKind         ::= 'reject' | 'select'
BasicOMSOrSymbolList ::= '{' BasicOMS '}' | SymbolList
```

9.8.1 Keywords and signs

The lexical symbols of the DOL text serialization include various ~~key words~~ keywords and signs that occur as terminal symbols in the context-free grammar in annex K. ~~Key words~~ Keywords and signs that represent mathematical signs are displayed as such, when possible, and those signs that are available in the Unicode character set may also be used for input.

9.8.1.1 Keywords

~~Key words~~ Keywords are always written lowercase. The following ~~key words~~ keywords are reserved, and are not available for use as variables or as CURIEs with no prefix[19], although they can be used as parts of tokens.

```
alignment
along
assuming
and
closed-world
cofree
combine
cons-ext
end
entails
entailment
equivalence
excluding
extract
free
hide
import
in
for
forget
interpretation
keep
language
library
logic
maximize
model
minimize
network
ni
of
oms
onto
ontology
refined
refinement
reject
relation
remove
result
reveal
select
separators
serialization
spec
```

[19] In such a case, one can still rename affected variables, or declare a prefix binding for affected CURIEs, or use absolute IRIs instead. These rewritings do not change the semantics.