9. DOL abstract syntax

9.1. Abstract syntax categories

DOL provides abstract syntax categories for

- heterogeneous OMS (which can be basic OMS in some OMS language, or unions, translations, minimizations, combinations, approximations of OMS, among others)
- distributed OMS (items in distributed OMS are: OMS definitions, OMS mapping definitions, and qualifications choosing the logic, OMS language and/or serialization)
- identifiers
- annotations

Additionally, the categories of the abstract syntaxes of any conforming OMS languages (cf. clause 2.1) are also DOL abstract syntax categories.

The following subclauses, one per abstract syntax category, specify the abstract syntax of DOL in EBNF ISO/IEC 14977:1996. Note that ISO EBNF lacks an operator for "at least one repetition". This OMG Specification therefore adopts the following convention: Whenever some sequence `S` is repeated at least once, we give it a non-terminal identifier of its own (`RepeatedS = S { S } ;`), or group it as in `LongerExpression = Foo Bar ( S { S } ) ;`.

9.2. Distributed OMS

A distributed OMS consists of named (possibly heterogeneous) OMS, and mappings between its participating (heterogeneous) OMS. More specifically, a distributed OMS consists of a name, followed by a list of `DistOMSItem`s. A `DistOMSItem` is either an OMS definition (`OMSDefn`), or a mapping between OMS (`MappingDefn`), or a `Qualification` selecting a specific OMS language, logic and/or syntax that is used to interpret the subsequent `DistOMSItem`s. Alternatively, a distributed OMS can also be the verbatim inclusion of an OMS written in an OMS language that conforms with DOL (`OMSInConformingLanguage`; cf. 2.1).

```
DistOMS                    = [ PrefixMap ] , DistOMSDefn
                           | OMSInConformingLanguage ;
DistOMSDefn                = 'dist-oms-defn' , DistOMSName , { DistOMSItem } ;
OMSInConformingLanguage    = <language and serialization specific> ;
DistOMSItem                = OMSDefn | MappingDefn | Qualification ;
Qualification              = LanguageQual | LogicQual | SyntaxQual ;
LanguageQual               = 'lang-select' , LanguageRef ;
LogicQual                  = 'logic-select' , LogicRef ;
SyntaxQual                 = 'syntax-select' , SyntaxRef ;
DistOMSName                = IRI ;
```

[57] Note(57)

At the beginning of a distributed OMS, one can declare a `PrefixMap` for abbreviating long IRIs; see clause 9.5 for details.

9.3. Heterogeneous OMS

An OMS (`OMS`) can be one of the following:

- a basic OMS `BasicOMS` written inline, in a conforming serialization of a conforming OMS language[1],
- a translation of an OMS into a different signature or OMS language,
- a reduction of an OMS to a smaller signature and/or less expressive logic (that is, some non-logical symbols are hidden, but the semantic effect of sentences involving these is kept),
- an approximation of an OMS, normally in a sublogic, using a given approximation method (with the effect that sentences not expressible in the sublogic are weakened or removed),
- a union of OMS,
- an extension of an OMS by other ones, it can be optionally named and/or marked as conservative, monomorphic, definitional or implied,
- a module extracted from an OMS, using a restriction signature,
- a reference to an OMS existing on the Web,
- an OMS qualified with the OMS language that is used to express it,
- a combination of OMS (technically, this is a colimit, see [41]),
- a minimization of an OMS, forcing the subsequently declared non-logical symbols to be interpreted in a minimal way, while the non-logical symbols declared so far are fixed (alternatively, the non-logical symbols to be minimized and to be varied can be explicitly declared).

```
BasicOMS            = OMSInConformingLanguage ;
MinimizableOMS      = BasicOMS
                    | 'oms-ref' , OMSRef , [ ImportName ] ;
ExtendingOMS        = MinimizableOMS
                    | 'minimize' , MinimizableOMS ;
OMS                 = ExtendingOMS
                    | 'minimize-symbols' , OMS , CircMin , CircVars
```

[57] NOTE: FYI: Things changed from HetCASL:
- logic-select now mandatory (no default logic) and tree-scoped
- download-items (encourage linked data best practices instead)
- item-name-map (to be replaced by namespaces??)
- lib-version (to be replaced by metadata annotations, e.g. OMV)
- indirect-mapping (will always use full IRIs, and abbreviate them by syntactic namespaces)

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

```
                     | 'translation' , OMS , Translation
                     | 'reduction' , OMS , Reduction
                     | 'module-extract' , OMS , Extraction
                     | 'approximation' , OMS , Approximation
                     | 'union' , OMS , [ ConsStrength ] , OMS
                     | 'extension' , OMS , ExtensionOMS
                     | 'qual-oms' , { Qualification } , OMS
                     | 'bridge' , OMS, { Translation } , OMS
                     | 'combination' , Graph ;

CircMin              = Symbol , { Symbol } ;
CircVars             = { Symbol } ;

Translation          = 'renaming' , { LogicTranslation } , [ SymbolMapItems ] ;
LogicTranslation     = 'logic-translation' , OMSLangTrans ;

Reduction            = 'hidden' , { LogicReduction } , [ SymbolItems ]
                     | 'revealed' , [ SymbolMapItems ] ;
LogicReduction       = 'logic-reduction' , OMSLangTrans ;

SymbolItems          = 'symbol-items' , ( Symbol , { Symbol } ) ;
SymbolMapItems       = 'symbol-map-items' , ( SymbolOrMap , { SymbolOrMa   Note (58) [58]

Extraction           = 'extraction', ModuleProperties, [ InterfaceSignature ] ;

ModuleProperties     = Conservative | 'minimal' | 'safe' | 'depleting';

Approximation        = 'approximation' , InterfaceSignature , [ LogicRef ] ;

ExtensionOMS         = [ ConsStrength ] , [ ExtensionName ] , ExtendingOMS ;

ConsStrength         = Conservative | 'monomorphic'
                     | 'weak-definitional' | 'definitional' | 'implied' ;

Conservative         = 'consequence-conservative' | 'model-conservative' ;

InterfaceSignature   = 'interface-signature' , SymbolItems ;

ImportName           = IRI ;
ExtensionName        = IRI ;
```

An OMS definition `OMSDefn` names an OMS. It can be optionally marked as consistent, using `ConsStrength`.[2]. An `SymbolItems`, used in an OMS `Reduction`, is a list of non-logical symbols that are to be hidden. A `LogicReduction` denotes a logic reduction to a less expressive OMS language. A `SymbolMapItems`, used in OMS `Translations`, maps

[58] Note: TODO: say that this default may be overridden by specific logics, such as CASL

[2] More precisely, `'consequence-conservative'` here requires the OMS to have a non-trivial set of logical consequences, while `'model-conservative'` requires its satisfiability.

symbols to symbols[59] , or a logic translation. An OMS language translation `OMSLangTrans` or `ApproxMethod` can be either specified by its name (optionally qualified with source and target OMS language), or be inferred as the default translation or approximation method between a given source and target (where even the source may be omitted; it is then inferred as the OMS language of the current OMS). Note(59)

```
OMSDefn          = 'oms-defn' , OMSName , [ ConsStrength ] , OMS ;

Symbol           = IRI ;
SymbolMap        = 'symbol-map' , Symbol , Symbol ;
SymbolOrMap      = Symbol | SymbolMap ;
Term             = <an expression specific to a basic OMS language> ;

OMSName          = IRI ;

OMSRef           = IRI ;
OMSOrMappingorGraphRef  = IRI ;
ExtensionRef     = IRI ;

LoLaRef          = LanguageRef | LogicRef ;

LanguageRef      = IRI ;
LogicRef         = IRI ;
SyntaxRef        = IRI ;

OMSLangTrans     = 'named-trans' , OMSLangTransRef
                 | 'qual-trans' , OMSLangTransRef , LoLaRef , LoLaRef
                 | 'anonymous-trans' , LoLaRef , LoLaRef
                 | 'default-trans' , LoLaRef[60]  ;                 Note(60)

OMSLangTransRef  = IRI ;

ApproxMethod     = 'named-approx', ApproxMethodRef
                 | 'qual-approx' , ApproxMethodRef , LoLaRef
                 | 'default-approx' , LoLaRef[61]  ;                Note(61)
ApproxMethodRef  = IRI ;

ExtractionMethod = IRI ;
```

[59]Note: FYI: On 2012-07-18 we decided not to specify lambda-style symbol-to-term mappings for now. Would be convenient, but specifying its semantics in an OMS language independent way would require additional institution infrastructure – and the same effect can be achieved by auxiliary definitional extensions, cf. Colore (so promote this, informatively, as a "best practice"?)

[60]Note: TODO: need to figure out which of these we actually want to keep. named-trans and default-trans are sufficient, because the other ones contain redundant information that is only stated once more for clarity. Source and target logic of qual-trans are clear from inspecting the translation, and the source logic of anonymous-trans is clear from the OMS that is translated.

[61]Note: TODO: These alternatives are coherent with what we discussed about the approximation syntax with defaults, but they are different from OMSLangTrans. But see the comment for OMSLangTrans above.

9.4. OMS Mappings

A OMS mapping provides a connection between two OMS. A OMS mapping definition is the definition of either a named interpretation (`IntprDefn`), a named declaration of the relation between a module of an OMS and the whole OMS (`ModuleRelDefn`), or a named alignment (`AlignDefn`). The `SymbolMapItems` in an interpretation always must lead to a signature morphism; a proof obligation expressing that the (translated) source OMS logically follows from the target OMS is generated. In contrast to this, an alignment just provides a connection between two OMS without logical semantics, using a set of `Correspondences`. Each correspondence may map some OMS non-logical symbol to another one (possibly given by a term) and an optional confidence value. Moreover, the relation between the two non-logical symbols can be explicitly specified (like being equal, or only being subsumed). A `ModuleRelDefn` declares that a certain OMS actually is a module of some other OMS with respect to the `InterfaceSignature`.

```
MappingDefn          = IntprDefn | EquivDefn | GraphDefn | ModuleRelDefn | AlignDefn;

IntprDefn            = 'intpr-defn' , IntprName , [ Conservative ] , IntprType ,
                                 { LogicTranslation } , [ SymbolMapItems ] ;
IntprName            = IRI ;
IntprType            = 'intpr-type' , OMS , OMS ;

EquivDefn            = 'equiv-defn' , EquivName , EquivType , OMS ;
EquivName            = IRI ;
EquivType            = 'equiv-type' , OMS , OMS ;

GraphDefn            = 'graph-defn', GraphName, Graph ;
GraphName            = IRI ;
Graph                = 'graph', GraphElements , ExcludeExtensions
GraphElements        = 'graph-elements', { OMSOrMappingorGraphRef } ;
ExcludeExtensions    = 'exclude-imports' , { ExtensionRef } ;

ModuleRelDefn        = 'module-defn' , ModuleName , [ Conservative ] , ModuleType ,
                                  InterfaceSignature ;
ModuleName           = IRI ;
ModuleType           = 'module-type' , OMS , OMS ;

AlignDefn            = 'align-defn' , AlignName , [ AlignCard ] , AlignType[3]
                                 { Correspondence } ;
AlignName            = IRI ;
AlignCards           = AlignCardForward , AlignCardBackward[62]  ;          Note(62)
AlignCardForward     = 'align-card-forward' , AlignCard ;
AlignCardBackward    = 'align-card-backward' , AlignCard ;
AlignCard            = 'injective-and-total'
                     | 'injective'
```

[3]Note that this grammar uses "type" as in "the type of a function", whereas the Alignment API uses "type" for the totality/injectivity of the relation/function. For the latter, this grammar uses "cardinality".

[62]NOTE: TODO: mention that the default is twice "injective and total"

```
                       | 'total'
                       | 'neither-injective-nor-total' ;
AlignType              = 'align-type' , OMS , OMS ;

Correspondence         = CorrespondenceBlock
                       | SingleCorrespondence
                       | 'default-correspondence'[63] ;                  Note(63)
CorrespondenceBlock    = 'correspondence-block' , [ RelationRef ] , [ Confidence ][64]   Note(64)
                                                  { Correspondence } ;
SingleCorrespondence   = 'correspondence' , SymbolRef , [ RelationRef ] , [ Confidence ] ,
                                     TermOrSymbolRef , [ CorrespondenceID ][65]   Note(65)
 ;
CorrespondenceID       = IRI ;
SymbolRef              = IRI ;
TermOrSymbolRef        = Term | SymbolRef ;
RelationRef            = 'subsumes' | 'is-subsumed' | 'equivalent' | 'incompatible'
                       | 'has-instance' | 'instance-of' | 'default-relation'[66]   Note(66)
 | IRI ;
Confidence             = Double[67] ;                                     Note(67)
Double                 = ? a number ∈ [0,1] ? ;
```

[68] Note(68)

A symbol map in an interpretation is **required** to cover all non-logical symbols of the source OMS; the semantics specification in clause 10 makes this assumption[4]. Applications **shall** implicitly map those non-logical symbols of the source OMS, for which an explicit mapping is not given, to non-logical symbols of the same (local) name in the target OMS, wherever this is uniquely defined – in detail:

```
Require: $O_s, O_t$ are OMS
Require: $M \subseteq \Sigma(O_s) \times \Sigma(O_t)$ maps non-logical symbols (i.e. elements of the signature) of
  $O_s$ to non-logical symbols of $O_t$
  for all $e_s \in \Sigma(O_s)$ not covered by $M$ do
    $n_s \leftarrow$ localname($e_s$)
    $N_t \leftarrow \{\text{localname}(e) | e \in \Sigma(O_t)\}$
    if $N_t = \{e_t\}$ then {i.e. if there is a unique target}
      $M \leftarrow M \cup \{(e_s, e_t)\}$
    end if
```

[63]Note: TODO: add concrete syntax, plus explanation: applies current default correspondence to all non-logical symbols with the same local names, using the "same local name" algorithm presented elsewhere

[64]Note: TODO: How do we say that at least one of these should be given?

[65]Note: TODO: concrete syntax, e.g., a = x, b my:similarTo y %(correspond-b-to-y)%, c my:similarTo 0.75 z

[66]Note: TODO: say that, unless a different default is specified in a surrounding `CorrespondenceBlock`, the default is `'equivalent'`

[67]Note: TODO: check if Double really makes sense for *implementations*, maybe we'd like to compare confidence values for equality

[68]Note: TODO: cite Alignment API for `RelationRef`; recommend linked data for `RelationRef = IRI`, or recommend registry?

[4]Mapping a non-logical symbol twice is an error. Mapping two source non-logical symbols to the same target non-logical symbol is legal, this then is a non-injective OMS mapping.

```
  end for
Ensure: $M$ completely covers $\Sigma(O_s)$
```

The local name of a non-logical symbol is determined as follows[5]:

```
Require: $e$ is a non-logical symbol (identified by an IRI; cf. clause 9.5)
  if $e$ has a fragment $f$ then {production ifragment in IETF/RFC 3987:2005}
    return $f$
  else
    $n \leftarrow$ the longest suffix of $e$ that matches the Nmtoken production of XML W3C/TR REC-xml:2008
    return $n$
  end if
```

[69] Note(69)

9.5. Identifiers

This section specifies the abstract syntax of identifiers of DOL OMS and their elements.

9.5.1. IRIs

In accordance with best practices for publishing OMS on the Web, identifiers of OMS and their elements **should** not just serve as *names*, but also as *locators*, which, when dereferenced, give access to a concrete representation of an OMS or one of its elements. (For the specific case of RDFS and OWL OMS, these best practices are documented in [20]. The latter is a specialization of the linked data principles, which apply to any machine-processable data published on the Web [28].) It is recommended that publicly accessible DOL OMS be published as linked data.

[70]Therefore, in order to impose fewer conformance requirements on applications, DOL commits to using IRIs for identification IETF/RFC 3987:2005. It is **recommended** that distributed OMS use IRIs that translate to URLs when applying the algorithm for mapping IRIs to URIs specified in IETF/RFC 3987:2005, Section 3.1. DOL descriptions of any element of a distributed OMS that is identified by a certain IRI **should** be *located* at the corresponding URL, so that agents can locate them. As IRIs are specified with a concrete syntax in IETF/RFC 3987:2005, DOL adopts the latter into its abstract syntax as well as all of its concrete syntaxes (serializations)[71] . Note(70) Note(71)

In accordance with semantic web best practices such as the OWL Manchester Syntax [17], this OMG Specification does not allow relative IRIs, and does not offer a mechanism for defining a base IRI, against which relative IRIs could be resolved.

Concerning these languages, note that they allow arbitrary IRIs in principle, but in practice they strongly recommend using IRIs consisting of two components [20]:

[5]In practice, this can often have the effect of undoing an IRI abbreviation mechanism that was used when writing the respective OMS (cf. clause 9.5). In general, however, functions that turn abbreviations into IRIs are not invertible. For this reason, the implicit mapping of non-logical symbols is specified independently from IRI abbreviation mechanisms possibly employed in the OMS.

[69]NOTE: some text that was left over here, but I don't recall what we meant by it: recommendations for dealing with OMS language dialects

[70]NOTE: Q-AUT: Does this motivation/justification sound reasonable to you?

[71]NOTE: Q-ALL: I meant to say: for IRIs, the abstract syntax is the same as the concrete syntax.

namespace an IRI that identifies the complete OMS (a *basic OMS* in DOL terminology), usually ending with # or /

local name a name that identifies a non-logical symbol within an OMS

```
IRI     = 'full-iri' , FullIRI | 'curie' , CURIE[6] ;
FullIRI = ? as defined by the IRI production in IETF/RFC 3987:2005 ? ;
```

9.5.2. Abbreviating IRIs using CURIEs

As IRIs tend to be long, and as syntactic mechanisms for abbreviating them have been standardized, it is **recommended** that applications employ such mechanisms and support expanding abbreviative notations into full IRIs. For specifying the *semantics* of DOL, this OMG Specification assumes full IRIs everywhere, but the DOL abstract *syntax* adopts CURIEs (compact URI expressions) as an abbreviation mechanism, as it is the most flexible one that has been standardized to date.

The CURIE abbreviation mechanism works by binding prefixes to IRIs. A CURIE consists of a *prefix*, which may be empty, and a *reference*. If there is an in-scope binding for the prefix, the CURIE is valid and expands into a full IRI, which is created by concatenating the IRI bound to the prefix and the reference.

DOL adopts the CURIE specification of RDFa Core 1.1 W3C/TR REC-rdfa-core-20120607, Section 6 with the following changes:

- DOL does not support the declaration of a "default prefix" mapping [72] (covering CURIEs such as `:name`). Note(72)
- DOL does support the declaration of a "no prefix" mapping (covering CURIEs such as `name`).
- DOL does not make use of the `safe_curie` production.
- DOL does not allow binding a relative IRI to a prefix.
- Concrete syntaxes of DOL are encouraged but **not required** to support CURIEs.[7]

CURIEs can occur in any place where IRIs are allowed, as stated in clause 9.5.1. Informatively, we can restate the CURIE grammar supported by DOL as follows:

```
CURIE     = [ Prefix ] , Reference ;
Prefix    = NCName , ':' (* see "NCName" in W3C/TR REC-xml-names:2009, Section 3 *) ;
Reference = Path , [ Query ] , [ Fragment ] ;
Path      = ipath-absolute | ipath-rootless | ipath-empty
            (* as defined in IETF/RFC 3987 *) ;
Query     = '?' , iquery (* as defined in IETF/RFC 3987 *) ;
Fragment  = '#' , ifragment (* as defined in IETF/RFC 3987 *) ;
```

[6]specified below in clause 9.5.2

[72]NOTE: Q-AUT: Are such explanatory notes OK here?

[7]This is a concession to having an RDF-based concrete syntax among the normative concrete syntaxes. RDFa is the only standardized RDF serialization to support CURIEs so far. Other serializations, such as RDF/XML or Turtle, support a subset of the CURIE syntax, whereas some machine-oriented serializations, including N-Triples, only support full IRIs.

Prefix mappings can be defined at the beginning of a distributed OMS (specified in clause 9.2; these apply to all parts of the distributed OMS, including basic OMS as clarified in clause 9.5.3). Their syntax is:

```
PrefixMap         = 'prefix-map' , { PrefixBinding } ;
PrefixBinding     = 'prefix-binding' , BoundPrefix , IRIBoundToPrefix ;
BoundPrefix       = 'bound-prefix' , [ Prefix ] ;
IRIBoundToPrefix = 'full-iri' , FullIRI ;
```

Bindings in a prefix map are evaluated from left to right. Authors **should not** bind the same prefix twice, but if they do, the later binding wins.

9.5.3. Mapping identifiers in basic OMS to IRIs

While DOL uses IRIs as identifiers throughout, basic OMS languages do not necessarily do; for example:

- OWL W3C/TR REC-owl2-syntax:2009, Section 5.5 does use IRIs.
- Common Logic ISO/IEC 24707:2007 supports them but does not enforce their use.
- F-logic [26] does not use them at all.

However, DOL OMS mappings as well as [73] certain operations on OMS require making unambiguous references to non-logical symbols of basic OMS (`SymbolRef`). Therefore, DOL provides a function that maps global identifiers used within basic OMS to IRIs. This mapping affects all non-logical symbol identifiers (such as class names in an OWL ontology), but not locally-scoped identifiers such as bound variables in Common Logic ontologies. DOL reuses the CURIE mechanism for abbreviating IRIs for this purpose (cf. clause 9.5.2). Note(73)

CURIEs that have a prefix may not be acceptable identifiers in every serialization of a basic OMS language, as the standard CURIE separator character, the colon (`:`), may not be allowed in identifiers. [74] Therefore, the declaration of DOL-conformance of the respective serialization (cf. clause 2.2) **may** define an *alternative CURIE separator character*, or it **may** forbid the use of prefixed CURIEs altogether. Note(74)

The IRI of a non-logical symbol identifier in a basic OMS O is determined by the following function:

```
Require: $D$ is a distributed OMS
Require: $O$ is a basic OMS in serialization $S$
Require: $id$ is the identifier in question, identifying a symbol in $O$ according to the specification of $S$
Ensure: $i$ is an IRI
  if $id$ represents a full IRI according to the specification of $S$ then
    $i \leftarrow id$
  else
    {first construct a pattern $cp$ for CURIEs in $S$, then match $id$ against that pattern}
    if $S$ defines an alternative CURIE separator character $cs$ then
      $sep \leftarrow cs$
    else if $S$ forbids prefixed CURIEs then
```

[73] Note: TODO: maybe clarify which ones, by checking the grammar for all occurrences of SymbolRef

[74] Note: Q-ALL: I recall that in the 2012-04-18 teleconference we agreed on this – but does it really make sense? Are there any relevant OMS language serializations that do not allow : in identifiers (or that do allow it theoretically but discourage it in practice) but allow some other non-letter character?

```
    $sep \leftarrow$ undefined
  else
    $sep \leftarrow$ : {the standard CURIE separator character}
  end if
  {The following statements construct a modified EBNF grammar of CURIEs; see ISO/IEC
  14977:1996 for EBNF, and clause 9.5.2 for the original grammar of CURIEs.}
  if $sep$ is defined then
    $cp \leftarrow [NCName, sep], Reference$
  else
    $cp \leftarrow Reference$
  end if
  if $id$ matches the pattern $cp$, where $ref$ matches $Reference$ then
    if the match succeeded with a non-empty $NCName$ $pn$ then
      $p \leftarrow concat(pn, :)$
    else
      $p \leftarrow$ no prefix
    end if
    if $O$ binds $p$ to an IRI $pi$ according to the specification of $S$ then
      $nsi \leftarrow pi$
    else
      $P \leftarrow$ the innermost prefix map in $D$, starting from the place of $O$ inside $D$, and
      going up the abstract syntax tree towards the root of $D$
      while $P$ is defined do
        if $P$ binds $p$ to an IRI $pi$ then
          $nsi \leftarrow pi$
          break out of the while loop
        end if
        $P \leftarrow$ the next prefix map in $D$, starting from the place of the current $P$ inside
        $D$, and going up the abstract syntax tree towards the root of $D$
      end while
      return an error
    end if
    $i \leftarrow concat(nsi, ref)$
  else
    return an error
  end if
end if
return $i$
```

This mechanism applies to basic OMS given inline in a distributed OMS document (`BasicOMS`), not to OMS in external documents (`OMSInConformingLanguage`); the latter **shall** be self-contained.

While CURIEs used for identifying parts of a distributed OMS (cf. clause 9.5.2) are merely syntactic sugar, the prefix map for a basic OMS is essential to determining the semantics of the basic OMS within the distributed OMS. Therefore, any DOL serialization **shall** provide constructs for expressing such prefix maps, even if the serialization does not support prefix maps otherwise.

[75] Note(75)

9.6. DOL Serializations

Say how existing OMS in existing serializations have to be adapted/wrapped (or ideally: not adapted at all!) in order to become valid OMS in some DOL serialization.[76][77] Note(76) Note(77)

9.7. Annotations

[78] [79] Annotations always have a subject, which is identified by an IRI. Where the given OMS language does not provide a way of assigning IRIs to a desired subject of an annotation (e.g. if one wants to annotate an import in OWL), a distributed OMS may employ RDF annotations that use XPointer or IETF/RFC 5147 as means of non-destructively referencing pieces of XML or text by URI.[8] Note(78) Note(79)

[75]NOTE: TODO: somewhere we need to mention semantic annotations to embedded fragments in conforming OMS languages, e.g. %implied

[76]NOTE: TODO: Essential points are:– need to be able to say: "the file at URL U is in OWL 2 Manchester syntax"– maybe use packaging/wrapping format– compare MIME types, HTTP content negotiation (but don't go too deep into communication protocols)

[77]NOTE: Reply: Maybe we can implement something like the Linux command "file"?

[78]NOTE: this subclause will be moved to annex M

[79]NOTE: TODO: Properly integrate this text from our LaRC 2011 paper

[8]We intend to utilise the extensibility of the XPointer framework by developing additional XPointer schemes, e.g. for pointing to subterms of Common Logic sentences.