

Simplified Common Logic (SCL)

Draft Metamodel

AT&T/Gentleware DSTC
IBM Sandpiper Software

3 February 2004

Motivation

- Ensure that the ODM meets requirements in the Usage Scenarios and Goals document
- Emphasis on standards, including emerging standards in knowledge representation with well defined semantics
- Emphasis on commonly used knowledge representations and on ease of use by ontologists, knowledge engineers, and subject matter experts
- Desire to support automatic selection, composition, and interoperation of web services as an early application / demonstration vehicle
- Requires representation of pre- and post conditions, actions that change the state of the world, planning, complex constraints and preferences, etc.
- Requires reasoning about action and change as well as understanding the consequences of some action within the context of a situation
- The declarative representation of such information requires first order logic

Evolution of KIF / CL / SCL

- The Knowledge Interchange Format (KIF) – developed in the late 1980s
- KIF Reference Manual, Version 3.0 – http://ksl.stanford.edu/KSL_Abstracts/KSL-92-86.html
- Several “flavors” of KIF have emerged since:
 - IDEF5 (1994) – <http://www.kbsi.com/technology/methods/sbont.htm>
 - Draft ANSI KIF (1998) – <http://logic.stanford.edu/kif/dpans.html>
 - FIPA KIF Content Specification Language (2000 / 2001) – <http://www.fipa.org/specs/fipa00010/XC00010C.html>
 - Draft ISO KIF Part 1: First Order KIF (2001) – <http://cl.tamu.edu/discuss/prop.html>
 - Draft IEEE SUO KIF (2002) – <http://suo.ieee.org/KIF/>
- Common Logic (CL) initiative brought to ISO JTF 1 / SC 32 / WG 2 on metadata standards by John Sowa (ANSI NCITS L8 – March 2002) – <http://cl.tamu.edu/>
- Languages initially proposed under the CL umbrella included KIF, Prolog, Conceptual Graphs, OCL, Z
- Intent was to be able to support vendor representations – Cyc-L, Ontos’, XSB, Haley Eclipse, NASA CLIPS
- SCL goal – to create a simple, powerful representation; WG includes Pat Hayes, Chris Menzel, other well known logicians (2003)

About SCL

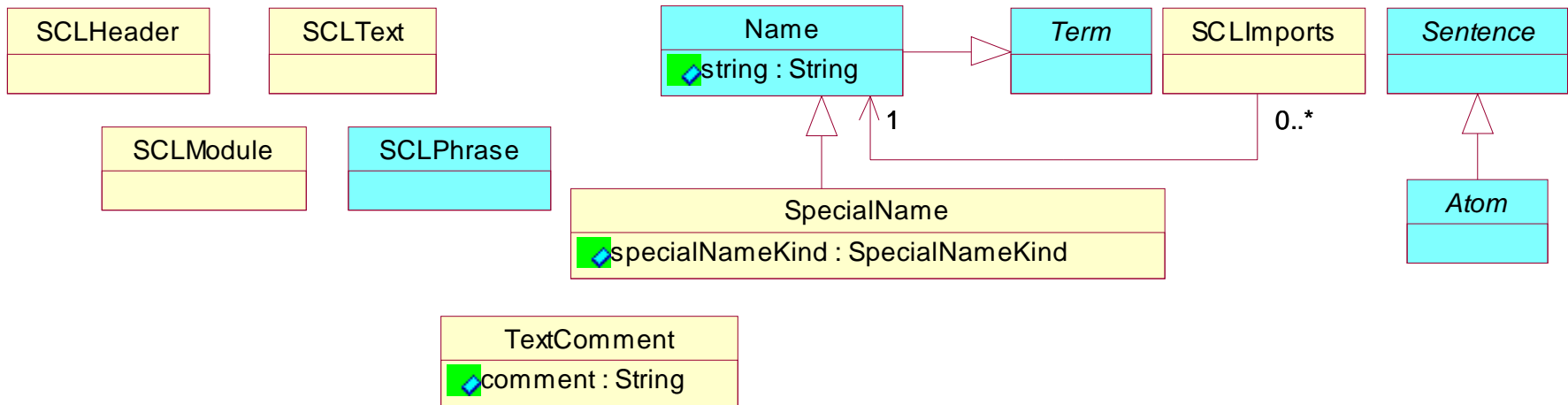
- SCL – FOL language for information exchange and transmission; it allows for a variety of different syntactic forms, expressible within a common XML-based syntax, sharing a single semantics
- Several features of SCL, *e.g.*, its use of sequence variables, are borrowed from KIF
- KIF – a common notation into which other languages could be translated without loss of meaning. SCL – for information interchange over a network without requiring translation; SCL provides a single common *semantic* framework, rather than a syntactically defined interlingua.
- KIF contains representative syntax for a variety of forms of expressions, *e.g.*, quantifier sorting, definition formats, a fully expressive meta-language. SCL is deliberately minimal – easier to state a precise semantics and to place exact bounds on the expressiveness of subsets of the language. It allows extended languages to be defined as encodings of axiomatic theories expressed in SCL, or by reduction to the SCL kernel.
- KIF was based explicitly on LISP. SCL is not LISP-based – motivated by languages for the semantic web, intended to be useable as an improved Lbase (<http://www.w3.org/TR/rdf-mt/>), *i.e.* as serving as a semantic reference language for other SW notations (*e.g.*, OWL).
- SCL specification will be presented to the ISO JTF 1 / SC 32 / WG 2 next month – see http://www.ihmc.us/users/phayes/SCL_december_3.html

SCL Metamodel Development Process

- Initiated development on a metamodel of ISO KIF, with intent to complete metamodels of both ISO KIF and draft ANSI KIF – 9/2003
- Solicited help from Dave Frankel and Deb McGuinness and completed initial draft ISO KIF metamodel – 10/2003
- Learned that the ISO KIF draft was superceded by SCL, solicited help from Pat Hayes to understand SCL and assist in modeling
- Continued SCL modeling effort, iteratively and synergistically with the SCL development team – 11/2003
- Goals included focus on completeness (multiplicities, role names, OCL invariants), accuracy in language representation, minimalistic and intuitive XMI/XML
- Completed draft metamodel with SCL team approval – 1/2004

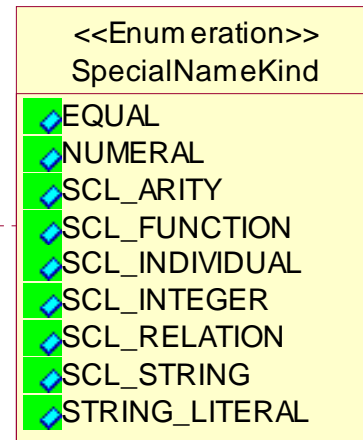
Expressions

reservedElement ::= 'and' | 'or' | 'iff' | 'implies' | 'forall' | 'exists' | 'not' | 'roleset:' | 'scl:imports' | 'scl:header' | 'scl:module'
name ::= specialname | ((alpha | other) wordchar*) - reservedElement
specialname ::= '=' | 'scl:imports' | 'scl:arity' | 'scl:Ind' | 'scl:Rel' | 'scl:header' | 'scl:Integer' | 'scl:String' | numeral | quotedstring



A STRING_LITERAL is what the SCL document calls a quotedstring. The name StringLiteral is less dependent on the surface syntax.

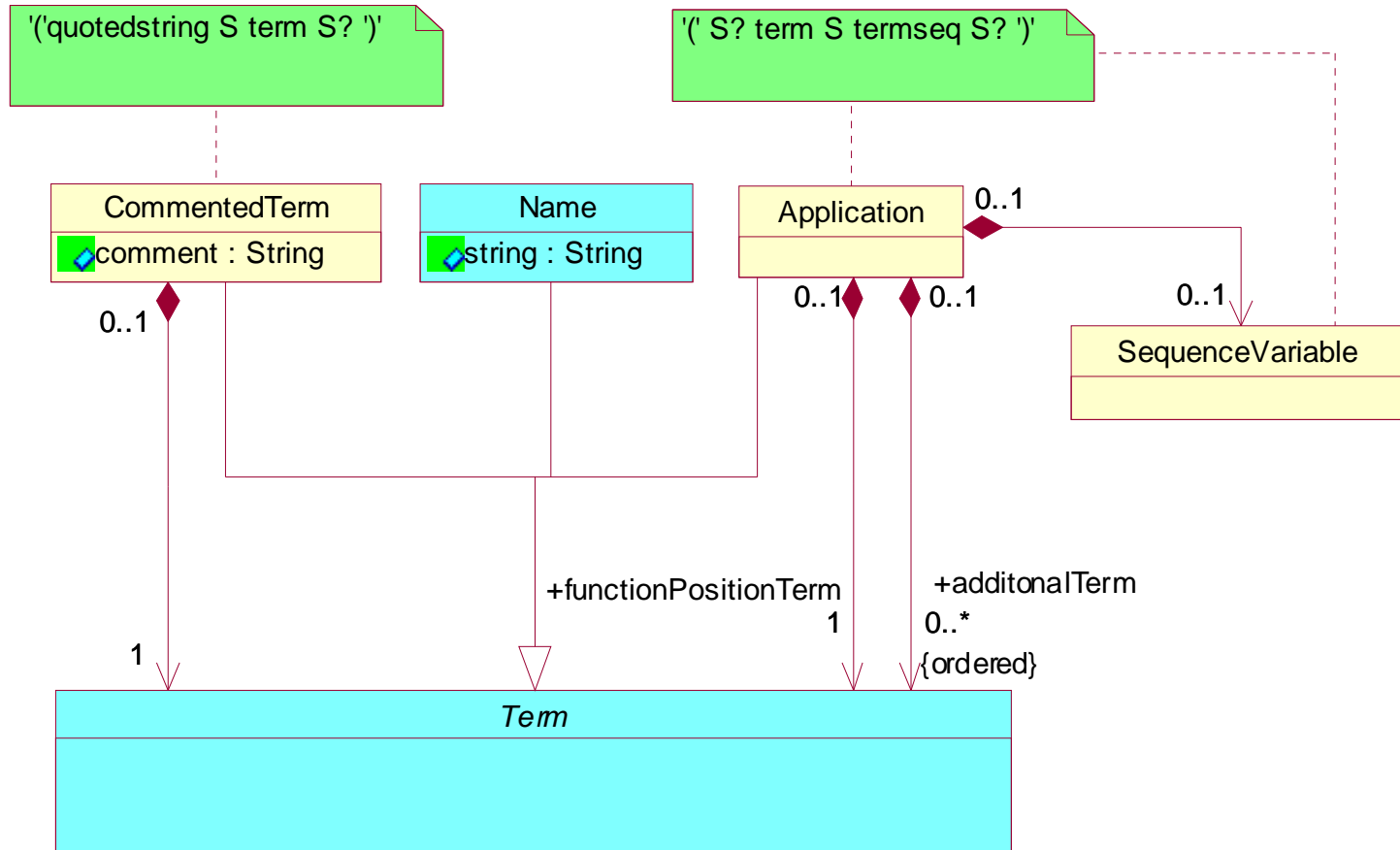
The SpecialNames enumerated here are those that represent semantic elements that are not explicitly referenced elsewhere in the metamodel and were thus grouped together in the enumeration for compactness.



Terms

$\text{termseq} ::= (\text{term } S?)^* \text{ seqvar?}$

$\text{term} ::= \text{name} \mid \text{'(} S? \text{ term } S \text{ termseq } S? \text{)' } \mid \text{'(quotedstring } S \text{ term } S? \text{)'}$



Names and Terms

- The lexical syntax section of the SCL document has rules for what can be a valid name.
- Some of these rules cannot be expressed in OCL. For example:
 - `name ::= ((alpha | other) wordchar*) - specialname`

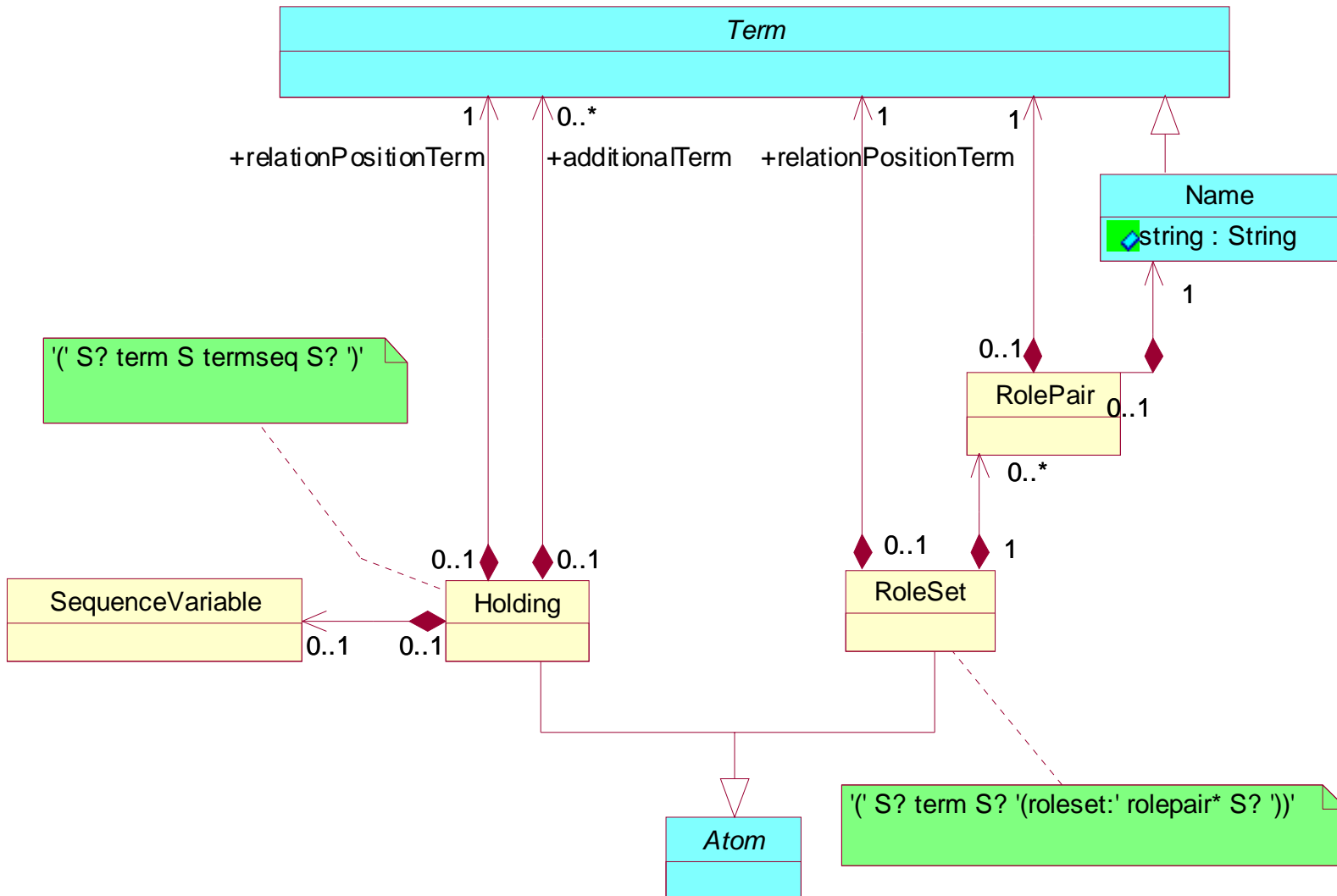
which depends on preceding definitions of alpha and other

- A name may not contain a quoted string
- We can use OCL to rule out special names but not to state rules about what kinds of ascii characters are allowed. Enforcing these rules has to be the responsibility of the parser.
- The Name/CommentedTerm/Application partition is disjoint
context Term inv DisjointPartion:
 - (self.ocIsKindOf(Name) xor self.ocIsKindOf(CommentedTerm)) and
 - (self.ocIsKindOf(Name) xor self.ocIsKindOf(Application)) and
 - (self.ocIsKindOf(CommentedTerm) xor self.ocIsKindOf(Application))
- Note lack of simple “disjoint” declaration in MOF

Atoms

rolepair ::= '(' S? name S term S? ')'

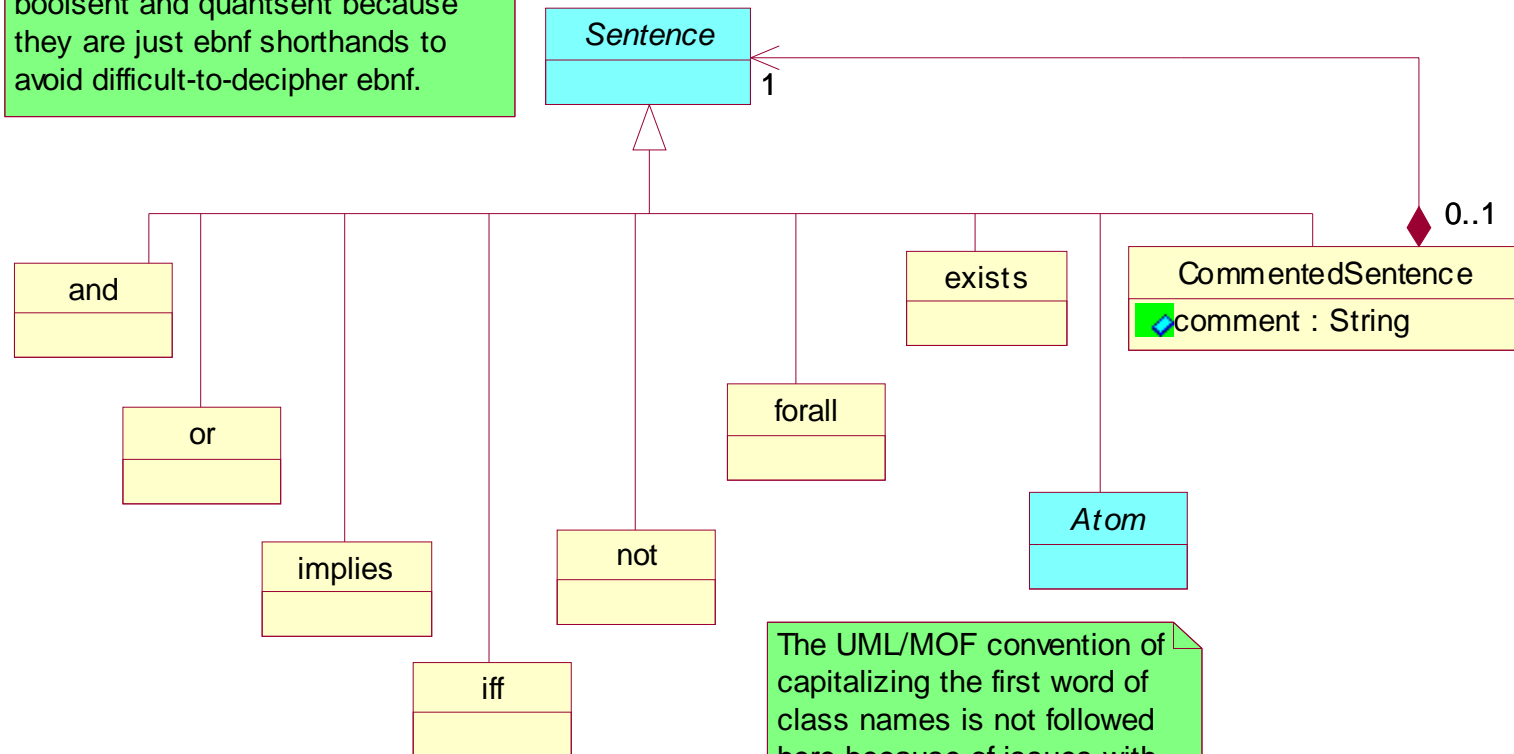
atom ::= '(' S? term S termseq S? ')' | '(' S? term S? '(roleset:' rolepair* S? ')'



Sentences

sentence ::= atom | boolsent | quantsent | '('quotedstring S? sentence ')'

We don't have metaclasses for boolsent and quantsent because they are just ebnf shorthands to avoid difficult-to-decipher ebnf.



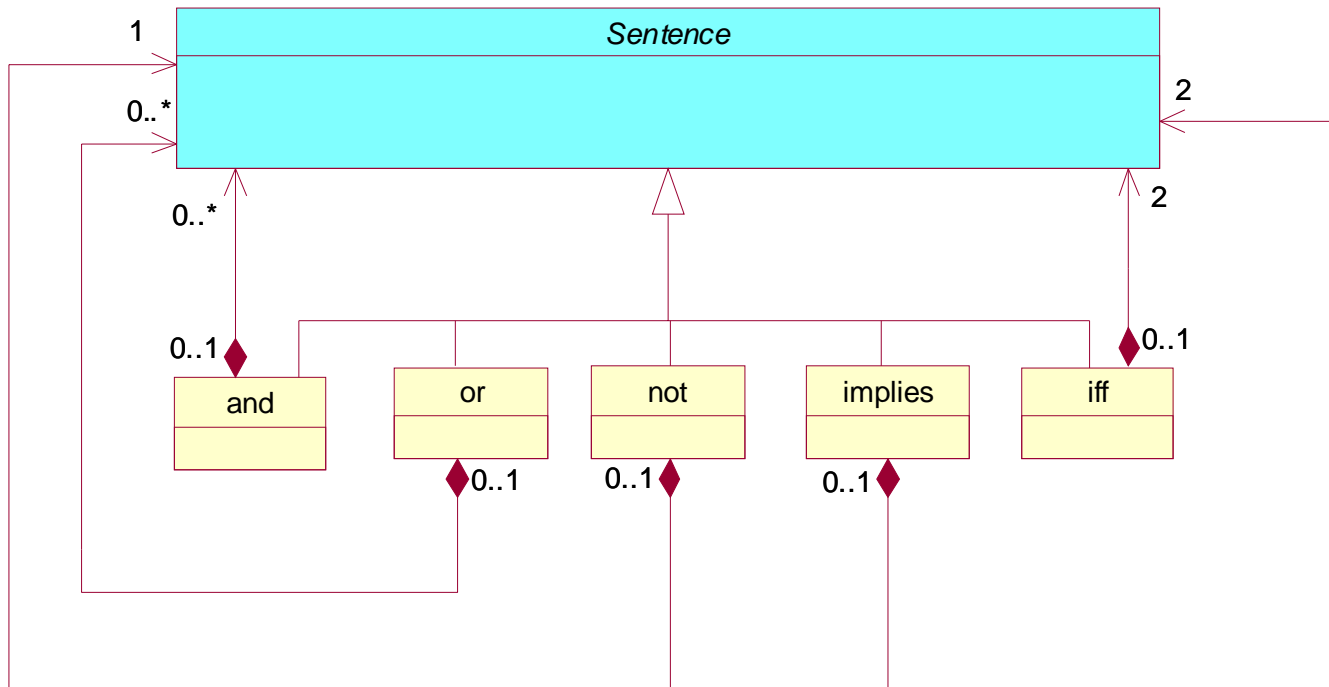
The UML/MOF convention of capitalizing the first word of class names is not followed here because of issues with normal usage in the ontology community

Atoms and Sentences

- Holding and RoleSet form a disjoint partition
context Atom inv DisjointPartition:
self.ocIsKindOf(Holding) xor self.ocIsKindOf(RoleSet)
- The partition formed by the subclasses of Sentence is disjoint
context Sentence inv DisjointPartition:
(self.ocIsKindOf(and) xor self.ocIsKindOf(or)) and
(self.ocIsKindOf(and) xor self.ocIsKindOf(not)) and
(self.ocIsKindOf(and) xor self.ocIsKindOf(implies)) and
(self.ocIsKindOf(and) xor self.ocIsKindOf(iff)) and
(self.ocIsKindOf(and) xor self.ocIsKindOf(forall)) and
(self.ocIsKindOf(and) xor self.ocIsKindOf(exists)) and
(self.ocIsKindOf(and) xor self.ocIsKindOf(Atom)) and
(self.ocIsKindOf(and) xor self.ocIsKindOf(CommentedSentence)) and
(self.ocIsKindOf(or) xor self.ocIsKindOf(not)) and
(self.ocIsKindOf(or) xor self.ocIsKindOf(implies)) and
(self.ocIsKindOf(or) xor self.ocIsKindOf(iff)) and
(self.ocIsKindOf(or) xor self.ocIsKindOf(forall)) and
...

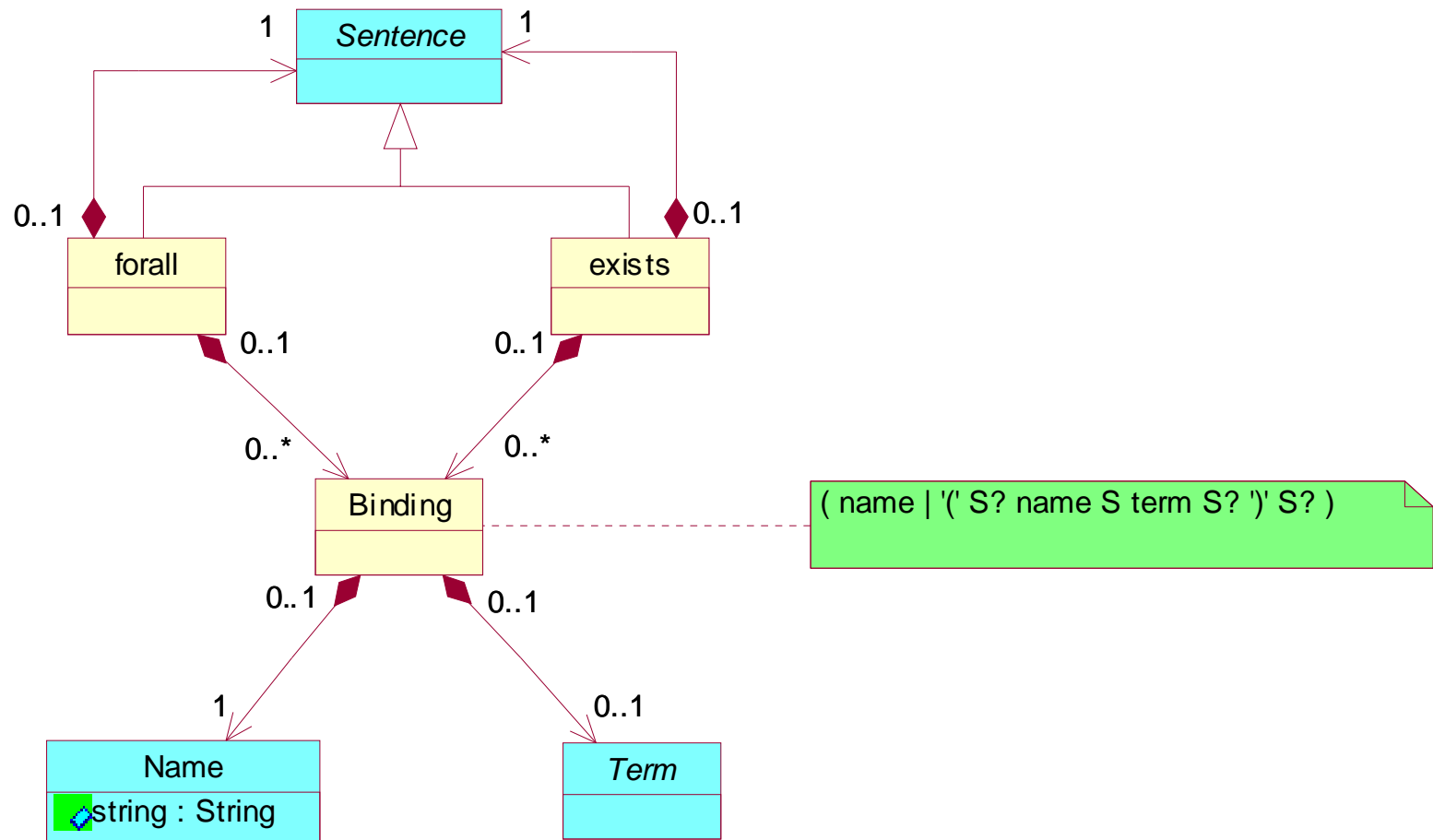
Boolean (Logical) Sentences

`boolsent ::= '(' (('and' | 'or') S) (sentence S?)* ')' | '(' ('implies' | 'iff') S? sentence S? sentence S? ')' | '(not' S? sentence S?)'`



Quantified Sentences

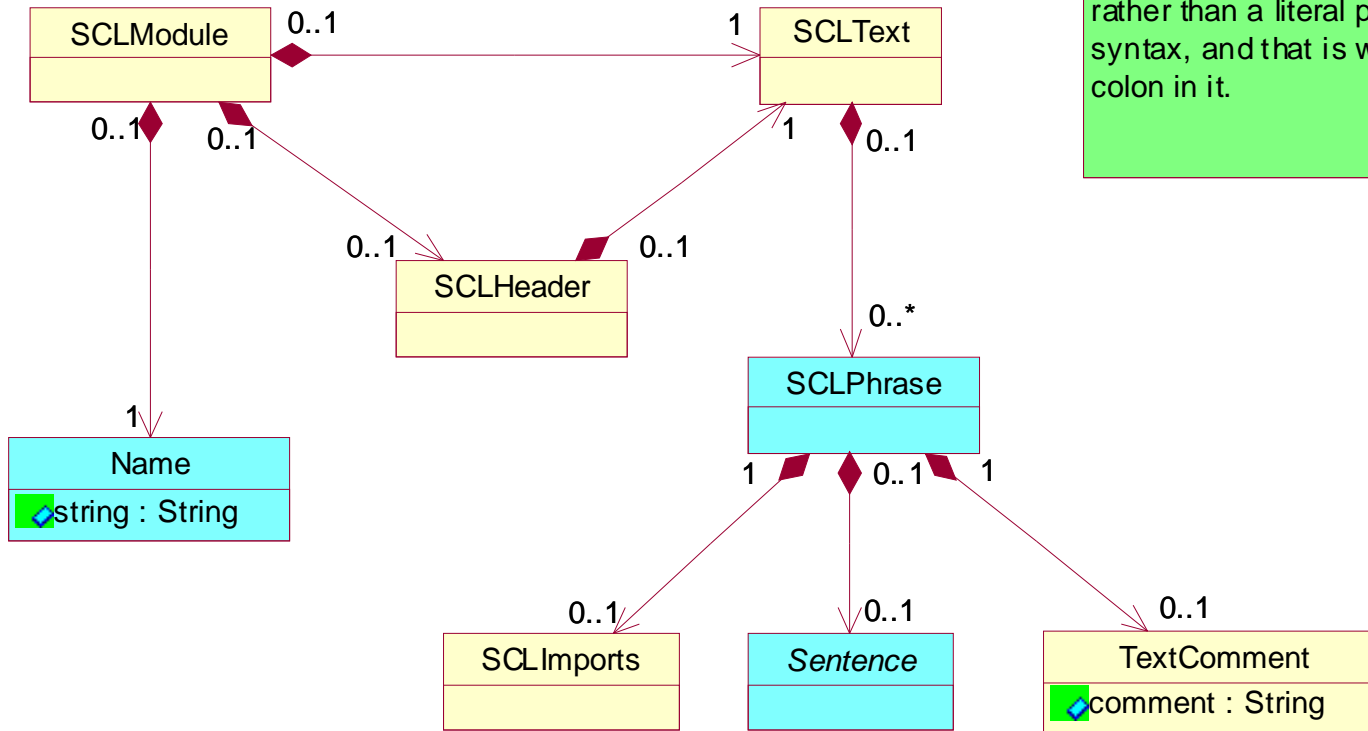
$\text{quantsent} ::= \text{'(S? ('forall' | 'exists' S?) ' S? (name | '(S? name S term S?)' S?)^* ' S? sentence S? '}$



Phrases

```
sclphrase ::= '(scl:imports' S name S? ')' | sentence | '(' quotedstring ')'  
scltext  ::= ( sclphrase S? )  
moduledefinition ::= '(scl:module' S? name S? '(' ( 'scl:header' scltext ')' )? S? scltext ')'
```

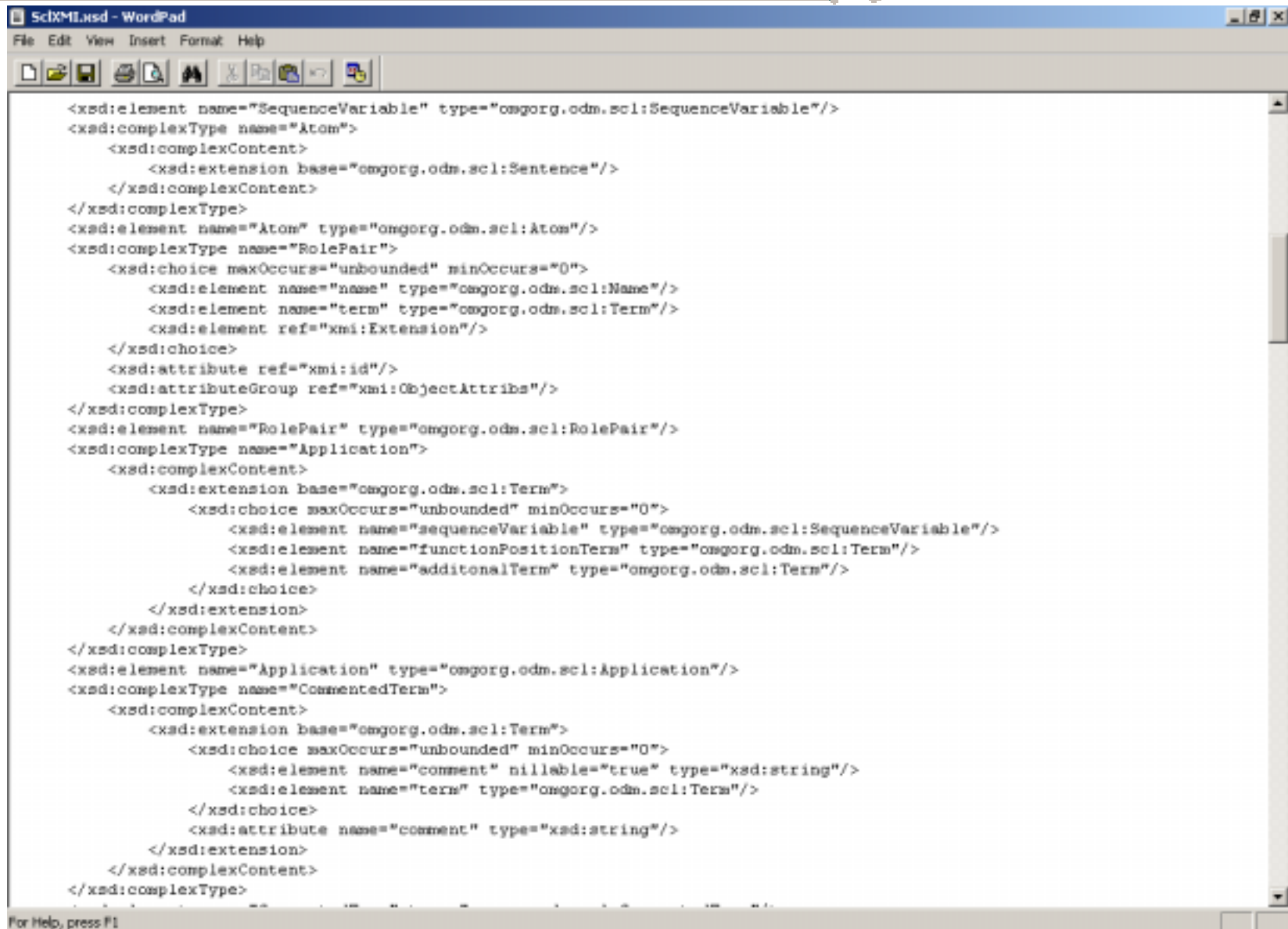
"sclphrase" is just an ebnf shorthand rather than a literal part of the surface syntax, and that is why there is no colon in it.



Phrases

- A phrase contains an Import, a Sentence, or a TextComment
context SCLPhrase inv XOR:
(self.import->notEmpty() xor self.sentence->notEmpty()) and
(self.import->notEmpty() xor self.textComment->notEmpty()) and
(self.sentence->notEmpty() xor self.textComment->notEmpty())

Draft XMI for SCL



```
<xsd:element name="SequenceVariable" type="omgorg.odm.scl:SequenceVariable"/>
<xsd:complexType name="Atom">
  <xsd:complexContent>
    <xsd:extension base="omgorg.odm.scl:Sentence"/>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Atom" type="omgorg.odm.scl:Atom"/>
<xsd:complexType name="RolePair">
  <xsd:choice maxOccurs="unbounded" minOccurs="0">
    <xsd:element name="name" type="omgorg.odm.scl:Name"/>
    <xsd:element name="term" type="omgorg.odm.scl:Term"/>
    <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
  <xsd:attribute ref="xmi:id"/>
  <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
</xsd:complexType>
<xsd:element name="RolePair" type="omgorg.odm.scl:RolePair"/>
<xsd:complexType name="Application">
  <xsd:complexContent>
    <xsd:extension base="omgorg.odm.scl:Term">
      <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="sequenceVariable" type="omgorg.odm.scl:SequenceVariable"/>
        <xsd:element name="functionPositionTerm" type="omgorg.odm.scl:Term"/>
        <xsd:element name="additionalTerm" type="omgorg.odm.scl:Term"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Application" type="omgorg.odm.scl:Application"/>
<xsd:complexType name="CommentedTerm">
  <xsd:complexContent>
    <xsd:extension base="omgorg.odm.scl:Term">
      <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="comment" nillable="true" type="xsd:string"/>
        <xsd:element name="term" type="omgorg.odm.scl:Term"/>
      </xsd:choice>
      <xsd:attribute name="comment" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

For Help, press F1

Next Steps

- Examples, where each example consists of
 - A sample SCL expression in SCL textual syntax
 - Corresponding abstract syntax tree as an instance of the metamodel, expressed as an object diagram
 - These kinds of examples were in the CWM specification and should be in all OMG metamodel specifications
- Validate OCL with OCL validator/parser
- Possible refinement of XMI via XMI's mapping parameters
 - XMI's schema production rules are parameterized
 - SCL XMI produced using defaults for all parameters
- Alignment of SCL with RDF/RDFS/OWL with assistance from Deb McGuinness and Pat Hayes
- Revised ODM Core based on the results of the alignment
- Review of RDF/RDFS/OWL metamodels with assistance from Pat Hayes, Deb McGuinness, and Peter Patel-Schneider in light of alignment and ODM Core revisions
- Narrative for the submission document
- Specification of compliance points