



---

## D16.1v02 BNF grammar for WSML user language

WSMO Working Draft – 18th April 2004

Author: Eyal Oren, Michael Kifer, Jos de Bruijn, Holger Lausen, Dumitru Roman, Michael Felderer  
Editor : Eyal Oren

This version:

<http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418>

Latest version:

<http://www.wsmo.org/2004/d16/d16.1/v0.2>

Previous version:

## Contents

<b>1 Introduction</b>	<b>2</b>
<b>2 Example definition</b>	<b>3</b>
<b>3 BNF grammar for WSML-U</b>	<b>5</b>
<b>4 Acknowledgment</b>	<b>12</b>

## 1 Introduction

This document presents the grammar for the Web Services Modelling Language for Users (WSML-U). This language is meant for representing the concepts introduced in the Web Service Modelling Ontology (WSMO) in a human-readable way. WSMO is a meta-ontology for describing various aspects related to Semantic Web Services. The four main elements of WSMO are: ontologies, web services, mediators and goals. These (and other) concepts and their relations form the meta-ontology WSMO, which is described in [5].

Using WSMO one can describe different web services, that each have a certain capability, which are expressed in a certain ontology, that use certain mediators to overcome heterogeneity; using WSMO one can also describe goals that one may have, expressed in a certain ontology, which could be achieved using one or more capabilities which a web service offers. One of the languages to express this all is WSML-U. All concepts from WSMO can be expressed in a certain way using WSML-U: language constructs exist to express ontologies, web services, goals, etc.

This document specifies unambiguously the syntax for WSML. The semantics of WSML-U will be described in other documents, yet to be written. The language to write down this syntax is a variant of Extended Backus Nauer Form; a variant readable by ANTLR [4]. ANTLR is a tool that can construct recognisers, parsers and compilers from a grammar specification, it's functionality is (among others) similar to the well-known lex/yacc tools. Visit [www.antlr.org](http://www.antlr.org) for more information.

As this is a draft version, certain issues have not yet been resolved. Some of these issues come from the fact that WSMO is also work in progress, so not all concepts have been defined completely yet; some of these issues are not related to WSMO as such, but are modelling decisions regarding only the language syntax. For completeness, we will name these outstanding issues explicitly:

- this version only describes WSML-Standard, as complement to WSMO-Standard. A version for WSML-Lite and WSML-Full will have to be defined also.
- this version does not describe the concepts of choreography and orchestration, as they are not defined in WSMO-Standard yet. A grounding of a Web Service to a URI is therefore not described yet, as this will be part of the choreography.
- the definition for the qualified names of identifiers has been modelled after the definition in [2]. However, it is not yet exactly the same. Future versions of this grammar will follow this definition completely.

## 2 Example definition

The complete definition of the grammar of WSML-U can be found in section 3. This grammar prescribes how a correct WSMO definition is written using WSML-U. Since the modelling decisions in the grammar (with respect to the resulting language) are not always easily readable, this section will introduce an example definition written in WSML-U. First we will show how to model F-Logic statements using WSML-U, using the same example as in [3]. Then we will model an example web service using an example ontology, from [1]. We model only a part of the example, as the modelling here serves only to explain the use of WSML-U; for an explanation of the use of WSMO (how to completely model this situation where users want to buy tickets and some web service offers that), please see [1].

Consider the following F-Logic statement, describing an object bob, with a name, an age, an affiliation in some department, of which he is the manager and where he has two assistants:

```
bob[
  name -> "Bob";
  age -> 40;
  affiliation -> cs1[
    dname -> "CS";
    mngr -> bob;
    assistants -> {john, sally}
  ]
]
```

This example fact is written in WSML-U as follows:

```
bob[
  name -> "Bob",
  age -> 40,
  affiliation -> cs1[
    dname -> "CS",
    mngr -> bob,
    assistants -> john AND sally
  ]
]
```

As can be seen, the WSML-U syntax is almost equal to Flora-2 syntax. Also note that whitespace and newline characters can be put in at will as they are ignored by the parser (because of the rule *WS* that ‘eats’ them).

Consider now a web service for buying airline tickets. This web service has one capability, namely to supply requesters with a airline ticket from some location to some other location. This web service uses a specific ontology, in which concepts like flight, location, carrier, etc. are defined. This web service has one interface on which to invoke its capability, which is linked to a WSDL definition. A partial WSML definition of this situation would look like this; modelled after [1]:

```
WEBSERVICE
  IDENTIFIER <wsnfp-01> END
  NONFUNCTIONALPROPERTIES
```

```

TITLE "Search, book and pay plane ticket" END
CREATOR "Sinuhe Arroyo" END
...
END
CAPABILITY
  IDENTIFIER <wsnfp-01-capability> END
  PRECONDITION
    origin => X AND destination => T AND departureDate => Y AND
    arrivalDate => Z AND departureTime => U AND arrivalTime => V
    AND carrier => W AND seat => R AND class => N AND
    origin => M AND destination => S
  END
  PRECONDITION
    validOrigin(X) AND validDestination(T) AND validTrip(X, T) AND
    validDepartureDate(Y) AND validArrivalDate(Z) AND
    validTripDates(Y, Z) AND validDepartureTime(U) AND
    validArrivalTime(V) AND validCarrier(W) AND
    validSeat(R) AND validTripCarrier(X, T, W)
  END
  POSTCONDITION
    travelTime => X AND date => Y AND price => Z AND carrier => U
    AND connections => V AND connectingDestinations => W AND
    connectingTime => R AND seat => N AND origin => M AND
    destination => S AND carrier => C
  END
  POSTCONDITION
    any(X) AND any(Y) AND any(Z) AND any(U) AND any(V) AND any(W) AND
    any(R) AND any(N) AND any(M) AND any(S)
  END
  ASSUMPTION
    arrivalTime => X AND connectingFlight.departureTime => Y AND
    destination => Z AND connectingFlight.origin => V AND
    timeConnectingFlight(X, Y) AND spaceConnectingFlight(Z, V)
    AND carrierConnectingFlight(Z, V, C)
  END
  EFFECT
    deliveredTicket
  END
INTERFACE
  IDENTIFIER <wsnfp-01-interfacel> END
  CHOR (placeholder choreography)
END
ONTOLOGY
  IDENTIFIER <wsnfp-01-ontology> END
  NONFUNCTIONALPROPERTIES
    TITLE "Plane Itinerary Ontology" END
    CREATOR "Sinuhe Arroyo" END

```

```

...
END
CONCEPT
  AXIOM flight END
  ATTRIBUTE
    AXIOM origin END
    RANGE AXIOM location END END
  ATTRIBUTE
  ATTRIBUTE
    AXIOM destination END
    RANGE AXIOM location END END
  ATTRIBUTE
  ...
END
CONCEPT
  AXIOM location END
  ATTRIBUTE
    AXIOM name END
    RANGE AXIOM string END END
  END
  ATTRIBUTE
    AXIOM airport END
    RANGE AXIOM boolean END END
  END
END
...
END

```

We can see that the WSML-U definition is clearly less concise than the equivalent F-Logic one. However, the assumption for defining WSML-U is that not all users are comfortable writing F-Logic statements, and would prefer a block-structured syntax even if it means losing conciseness. Since we cannot decide this question by personal favourites, discussion of this grammar is necessary to show whether this is a correct assumption.

### 3 BNF grammar for WSML-U

```

wsmoDefinition: (ontologyDefinition | webServiceDefinition |
  goalDefinition | mediatorDefinition)*
;

```

```

ontologyDefinition:
  ("REFONTOLOGY" Identifier "END") |
  ("ONTOLOGY"
  "IDENTIFIER" Identifier "END"
  (nonFunctionalDefinition)?
  ("USEDMEDIATOR" (ooMediatorDefinition)* "END"))?

```

```

    (axiomDefinition | conceptDefinition |
    relationDefinition | instanceDefinition)*
    "END" )
;

webServiceDefinition:
    ("REFWEBSERVICE" Identifier "END") |
    ("WEBSERVICE"
    "IDENTIFIER" Identifier "END"
    (nonFunctionalWSDefinition)?
    ("USEDMEDIATOR" (ooMediatorDefinition)* "END")?
    (capabilityDefinition)?
    (interfaceDefinition)*
    "END" )
;

goalDefinition:
    ("REFGOAL" Identifier "END") |
    ("GOAL"
    "IDENTIFIER" Identifier "END"
    (nonFunctionalDefinition)?
    ("USEDMEDIATOR" (ooMediatorDefinition | ggMediatorDefinition)* "END")?
    (postConditionDefinition)*
    (effectDefinition)*
    "END" )
;

mediatorDefinition:
    ooMediatorDefinition | ggMediatorDefinition | wgMediatorDefinition |
    wwMediatorDefinition
;

ooMediatorDefinition:
    ("REFOOMEDIATOR" Identifier "END") |
    ("OOMEDIATOR"
    "IDENTIFIER" Identifier "END"
    (nonFunctionalWSDefinition)?
    ("SOURCE" (ontologyDefinition | ooMediatorDefinition) "END")*
    ("TARGET" (ontologyDefinition | goalDefinition |
    webServiceDefinition | mediatorDefinition ) "END")*
    "MEDIATIONSERVICE" (goalDefinition | wwMediatorDefinition) "END"
    "END" )
;

ggMediatorDefinition:
    ("REFGGMEDIATOR" Identifier "END") |
    ("GGMEDIATOR"

```

```

    "IDENTIFIER" Identifier "END"
    (nonFunctionalWSDefinition)?
    "SOURCE" (goalDefinition | ggMediatorDefinition) "END"
    "TARGET" (goalDefinition | ggMediatorDefinition) "END"
    ("USEDMEDIATOR" (ooMediatorDefinition)* "END")?
    ("REDUCTION" axiomDefinition "END")?
"END" )
;

wgMediatorDefinition:
("REFWGMEDIATOR" Identifier "END") |
("WGMEDIATOR"
  "IDENTIFIER" Identifier "END"
  (nonFunctionalWSDefinition)?
  "SOURCE" (webServiceDefinition | wgMediatorDefinition) "END"
  "TARGET" (goalDefinition | wgMediatorDefinition) "END"
  ("USEDMEDIATOR" (ooMediatorDefinition)* "END")?
  ("REDUCTION" axiomDefinition "END")?
"END" )
;

wwMediatorDefinition:
("REFWWMEDIATOR" Identifier "END") |
("WWMEDIATOR"
  "IDENTIFIER" Identifier "END"
  (nonFunctionalWSDefinition)?
  "SOURCE" (webServiceDefinition | wwMediatorDefinition) "END"
  "TARGET" (webServiceDefinition | wwMediatorDefinition) "END"
  ("USEDMEDIATOR" (ooMediatorDefinition)* "END")?
"END" )
;

axiomDefinition:
("REFAXIOM" Identifier "END") |
(
  "IDENTIFIER" Identifier "END"
  (nonFunctionalDefinition)?
  logicalexpression
)
;

instanceDefinition:
"INSTANCE"
  axiomDefinition
  (instanceOfDefinition)*
  (attributeValueDefinition)*
"END"

```

```

;

instanceOfDefinition:  "INSTANCEOF"  conceptDefinition "END" ;

attributeValueDefinition:
  "ATTRIBUTEVALUE"
    axiomDefinition
    valueDefinition
  "END"
;

valueDefinition:  "VALUE"  axiomDefinition "END" ;

relationDefinition:
  "RELATION"
    axiomDefinition
    (parameterDefinition)*
  "END"
;

parameterDefinition:
  "PARAMETER"
    axiomDefinition
    domainDefinition
  "END"
;

domainDefinition:  "DOMAIN"  axiomDefinition "END" ;

conceptDefinition:
  "CONCEPT"
    axiomDefinition
    (superConceptDefinition)*
    (attributeDefinition)*
    (methodDefinition)*
  "END"
;

methodDefinition:
  "METHOD"
    axiomDefinition
    rangeDefinition
    (parameterDefinition)+
  "END"
;

attributeDefinition:

```

```

"ATTRIBUTE"
  axiomDefinition
  rangeDefinition
"END"
;

rangeDefinition: "RANGE" axiomDefinition "END" ;

superConceptDefinition: "SUPERCONCEPT" conceptDefinition "END" ;

capabilityDefinition:
("REFCAPABILITY" Identifier "END") |
("CAPABILITY"
  "IDENTIFIER" Identifier "END"
  (nonFunctionalDefinition)?
  ("USEDMEDIATOR" (ooMediatorDefinition | wgMediatorDefinition)* "END")?
  (preConditionDefinition | postConditionDefinition |
  assumptionDefinition | effectDefinition)*
"END" )
;

preConditionDefinition: "PRECONDITION" axiomDefinition "END" ;

postConditionDefinition: "POSTCONDITION" axiomDefinition "END" ;

assumptionDefinition: "ASSUMPTION" axiomDefinition "END" ;

effectDefinition: "EFFECT" axiomDefinition "END" ;

interfaceDefinition:
("REFINTERFACE" Identifier "END") |
("INTERFACE"
  "IDENTIFIER" Identifier "END"
  (nonFunctionalDefinition)?
  ("USEDMEDIATOR" (ooMediatorDefinition)* "END")?
  choreographyDefinition
  orchestrationDefinition
"END" )
;

choreographyDefinition: PLACEHOLDERCHOREOGRAPHY ;

orchestrationDefinition: PLACEHOLDERORCHESTRATION ;

nonFunctionalDefinition:
"NONFUNCTIONALPROPERTIES"
  nonFunctionalPropertiesList

```

```

"END"
;

nonFunctionalPropertiesList:
("TITLE" String "END")?
("CREATOR" Identifier "END")?
("SUBJECT" Identifier "END")*
("DESCRIPTION" Identifier "END")?
("PUBLISHER" Identifier "END")?
("CONTRIBUTOR" Identifier "END")?
("DATE" Date "END")?
("TYPE" Identifier "END")?
("FORMAT" Identifier "END")?
("SOURCE" Identifier "END")?
("LANGUAGE" Identifier "END")?
("RELATION" Identifier "END")?
("COVERAGE" Identifier "END")?
("RIGHTS" Identifier "END")?
("VERSION" Identifier "END")?
;

nonFunctionalWSDefinition:
"NONFUNCTIONALWSPROPERTIES"
  nonFunctionalPropertiesList
  ("PERFORMANCE" Identifier "END")?
  ("RELIABILITY" Identifier "END")?
  ("SECURITY" Identifier "END")?
  ("SCALABILITY" Identifier "END")?
  ("ROBUSTNESS" Identifier "END")?
  ("ACCURACY" Identifier "END")?
  ("TRANSACTIONAL" Identifier "END")?
  ("TRUST" Identifier "END")?
  ("FINANCIAL" Identifier "END")?
  ("NETWORKRELATEDQOS" Identifier "END")?
"END"
;

logicalexpression: rule | fact POINT;
fact: molecule;
existsClause: "EXISTS" PARLEFT VARCONST (COMMA VARCONST)* PARRIGHT;
forallClause: "FORALL" PARLEFT VARCONST (COMMA VARCONST)* PARRIGHT;
rule: ((existsClause)? head)? Implication ((forallClause)? body)?;
head: expr;
body: expr;
expr: (VARCONST)* molecule ( (AND expr) | (OR expr) )*;

// replaced to avoid left recursion caused e.g. by 'expr: expr (AND expr)?'

```

```

// head: conjunct;
// body: conjunct;
// expr: molecule | conjunct | disjunct;
// conjunct: (VARCONST)* expr (AND expr)?;
// disjunct: (VARCONST)* expr (OR expr)?;

molecule: termOrID ((MEMBER | SUBCLASS) term)?
  (PARLEFT method (COMMA method)* PARRIGHT)? ;
method: term ARROW molecule |
  CURLYLEFT molecule (COMMA molecule)* CURLYRIGHT ;
termOrID: Identifier | term;
term: String | Integer | VARCONST | reifiedFormula |
  FUNCTOR (PARLEFT term (COMMA term)* PARRIGHT)?;
reifiedFormula: DOLLAR CURLYLEFT expr CURLYRIGHT;

WS: (' ' | '\n' | '\t' | '\r' {newline();} | '\r' {newline();})+
  {$setType(Token.SKIP);} ;
ARROW: "->" | "=>" | "*->" | "*=>" ;
MEMBER: ":";
SUBCLASS: "::";
AND: "AND";
OR: "OR";
VARCONST: ('_')? (UpperCaseLetter)* (Letter | Digit)*;
FUNCTOR: (LowerCaseLetter)* (Letter | Digit)*;
PARLEFT: '(' ;
PARRIGHT: ')';
IMPLICATION: ":-";
BRACKETLEFT: '[';
BRACKETRIGHT: ']';
CURLYLEFT: '{';
CURLYRIGHT: '}';
DOLLAR: '$';
COMMA: ',';
POINT: '.';

protected Prefix: '<' NCName '>';
protected LocalPart: NCName;
protected NCName: (Letter | '_' ) (NCNameChar)* ;
protected NCNameChar: Letter | Digit | StrangeCharacters | CombiningChar ;
protected Letter: LowerCaseLetter | UpperCaseLetter;
protected UpperCaseLetter: 'A'..'Z';
protected LowerCaseLetter: 'a'..'z';
protected Digit: '0'..'9';
protected CombiningChar: '/' | '&' | Colon | '?' ;
protected StrangeCharacters: '.' | '-' | '_' ;
protected Colon: ':' ;

```

PLACEHOLDERCHOREOGRAPHY: "CHOR";  
PLACEHOLDERORCHESTRATION: "ORCH";

Identifier: '<' (Prefix ':'?) LocalPart '>';  
String: '\\"' (Letter | Digit | '.')+ '\\\"';  
Date: Digit Digit Digit Digit '/' Digit Digit '/' Digit Digit;  
Integer: Digit (Digit)\*;

## References

- [1] S. Arroyo, M. Stollberg, and Y. Ding. WSMO Primer. DERI Working Draft v01, 2004.
- [2] P. V. Biron and A. Malhotra (Eds). "XML Schema Part 2: Datatypes". W3C Recommendation, May 2001.
- [3] J. de Bruijn and M. Kifer. F-logic/XML - An XML Syntax for F-logic. WSMO Working Draft v01, 2004.
- [4] T.J. Parr and R.W. Quong. ANTLR: A predicated-LL(k) parser generator. *Software—Practice and Experience*, 25(7):789–810, July 1995.
- [5] D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology Standard. WSMO Working Draft v02, 2004.

## 4 Acknowledgment

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate program.

The editors would like to thank all the members of the WSMO working group for their advice and input into this document.