Semantic Web Services

The Web Service Modeling Language (WSML)

Where are we?

STI · INNSBRUCK

#	Title
1	Introduction
2	Web Science
3	Service Science
4	Web services
5	Web2.0 services
6	Semantic Web
7	Web Service Modeling Ontology (WSMO)
8	**Web Service Modeling Language (WSML)**
9	Web Service Execution Environment (WSMX)
10	OWL-S and other
11	Light-weight Annotations
12	Applications
13	Mobile Services

Outline

STI · INNSBRUCK

- Motivation
- Technical solution
  - Key Features of WSML
  - WSML Variants
  - Syntax Basics
  - Web Compliance
  - Conceptual Syntax
  - Logical Expressions Syntax
  - Restrictions imposed by WSML variants
  - WSML2Reasoner
- Illustration by a larger example
- Extensions
- Summary
- References

STI · INNSBRUCK

MOTIVATION

Motivation

- WSMO – a conceptual model for:
  - Ontologies
  - Web Services
  - Goals
  - Mediators
- However, WSMO does not provide a concrete syntax and semantic

⇒A unifying language (syntax and semantics) is needed for specifying WSMO Ontologies, Web Services, Goals, and Mediators

⇒The Web Service Modeling Language (WSML)

5

Motivation (cont')

- Existing semantic Web languages are not expressive enough to capture all aspects of WSMO
  - RDFS: only a very limited ontology language, and no support for Web services, Goals, or Mediators
  - OWL: a language for modeling ontologies based on the Description Logic paradigm, however does not offer direct means to describe Web Services, Goals and Mediators
  - OWL-S: an OWL ontology for the modeling Web services, but is not expressive enough for capturing all aspects of Web service; new languages such as SWRL or KIF have been embedded in OWL-S, however such combinations are often ambiguous and complicated

6

TECHNICAL SOLUTION

7

Key Features of WSML

- Three pillars of WSML, namely

(1) a *language independent conceptual model* for Ontologies, Web Services, Goals and Mediators, based on WSMO

(2) *Reuse* of several well-known logical language paradigms in *one* syntactical framework

(3) Web Compliance

8

Key Features of WSML (cont')

- One syntactic framework for a set of layered languages
  - Different Semantic Web and Semantic Web Service applications need languages of different expressiveness
  - No single language paradigm will be sufficient for all use cases
  - WSML investigates the use of Description Logics and Logic Programming for Semantic Web Services

9

Key Features of WSML (cont')

- Separation of conceptual and logical modeling
  - The conceptual syntax of WSML has been designed in such a way that it is independent of the underlying logical language
  - No or only limited knowledge of logical languages is required for the basic modeling of Ontologies, Web Services, Goals, and Mediators
  - The logical expression syntax allows expert users to refine definitions on the conceptual syntax using the full expressive power of the underlying logic, which depends on the particular language variant chosen by the user

10

Key Features of WSML (cont')

- Semantics based on well known formalisms
  - WSML captures well known logical formalisms such as Datalog and Description Logics in a unifying syntactical framework
    - WSML maintains the established computational properties of the original formalisms through proper syntactic layering
  - The variants allow the reuse of tools already developed for these formalisms
    - Efficient querying engines developed for Datalog
    - Efficient subsumption reasoners developed in the area of Description Logics
    - Inter-operation between the above two paradigms is achieved through a common subset based on Description Logic Programs

11

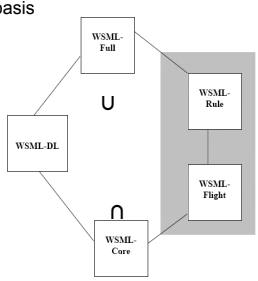
Key Features of WSML (cont')

- WWW Language
  - WSML adopts the IRI standard, the successor of URI, for the identification of resources, following the Web architecture
  - WSML adopts the namespace mechanism of XML and datatypes in WSML are compatible with datatypes in XML Schema and datatype functions and operators are based on the functions and operators of XQuery
  - WSML defines an XML syntax and an RDF syntax for exchange over the Web

12

WSML Variants

- WSML Variants - allow users to make the trade-off between the provided expressivity and the implied complexity on a per-application basis



13

WSML Variants (cont')

- WSML-Core - defined by the intersection of Description Logic and Horn Logic, based on Description Logic Programs
  - It has the least expressive power of all the languages of the WSML family and therefore has the most preferable computational characteristics
  - The main features of the language are the support for modeling classes, attributes, binary relations and instances
  - Supports class hierarchies, as well as relation hierarchies
  - Provides support for datatypes and datatype predicates

14

WSML Variants (cont')

- WSML-DL - an extension of WSML-Core which fully captures the Description Logic *SHIQ*(D), which captures a major part of the (DL species of the) Web Ontology Language OWL
- Differences between WSML-DL and OWL-DL:
  - No support for nominals (i.e. enumerated classes) as in OWL
  - Allows to write enumerations of individuals

    oneOf(Monday, Tuesday, Wednesday, Thursday, Friday,Saturday, Sunday)
  - Support for Qualified Cardinality Restrictions (QCR)
    - Used to constrain the number of values of a particular property
    - "The normal hand has exactly five fingers of which one is a thumb"
    - OWL does not support QCR: Class(NormalHand restriction (hasFinger cardinality (5)))
    - No possibility to model the fact that one finger is a thumb in OWL

15

WSML Variants (cont')

- WSML-Flight - an extension of WSML-Core with meta-modeling, constraints and nonmonotonic negation features
  - Based on a logic programming variant of F-Logic
  - Semantically equivalent to Datalog with inequality and (locally) stratified negation
  - Provides a powerful rule language
- WSML-Rule - an extension of WSML-Flight in the direction of Logic Programming
  - Captures several extensions such as the use of function symbols and unsafe rules, and does not require stratification of negation
  - The semantics for negation is based on the Stable Model Semantics

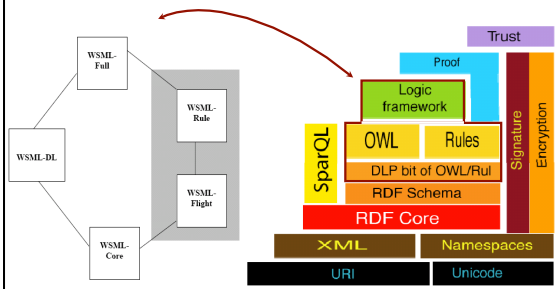
16

WSML Variants (cont')

- WSML-Full - unifies WSML-DL and WSML-Rule under a First-Order syntactic umbrella with extensions to support the nonmonotonic negation of WSML-Rule
  - Allows the full syntactic freedom of a First-Order logic and the full syntactic freedom of a Logic Programming language with default negation in a common semantic framework
- Compared to WSML-DL, WSML-Full adds full first-order modeling: n-ary predicates, function symbols and chaining variables over predicates
- Compared to WSML-Rule, WSML-Full adds disjunction, classical negation, multiple model semantics, and the equality operator

17

WSML – Relation to SW standards



18

Syntax Basics

- The WSML Full syntax consists of two major parts: the conceptual syntax and the logical expression syntax
  - The conceptual syntax is used for the modeling of ontologies, goals, web services and mediators; these are the elements of the WSMO conceptual model
  - Logical expressions are used to refine these definitions using a logical language
- The other language variants impose restrictions on the general syntax (WSML Full syntax)

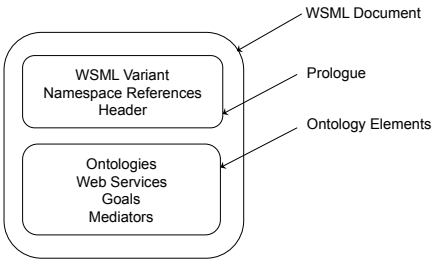
19

Syntax Basics – Main Parts

- A WSML specification is separated into two parts:
  - The first part provides a strictly ordered meta-information block about the specification:
    - WSML variant identification
    - Namespace references
    - Annotations
    - Import of ontologies
    - References to mediators used
    - The type of the specification
  - The second part of the specification, consisting of (unorderd) elements such as concepts, attributes, relations (in the case of an ontology specification), capability, interfaces (in the case of a goal or web service specification), etc.

20

WSML Document Structure



21

Namespaces and Identifiers

- WSML adopts the namespace mechanism of RDF; a namespace can be seen as part of an IRI
  - Namespaces can be used to syntactically distinguish elements of multiple WSML specifications and, more generally, resources on the Web
  - A namespace denotes a syntactical domain for naming resources
- An identifier in WSML is either a *data value*, an *IRI*, an *anonymous ID*, or a *variable*
  - The sets of identifiers of the following items are disjoint: ontology, goal, Web service, ooMediator, ggMediator, wgMediator, wwMediator, capability, interface, choreography, orchestration, state signature, grounding identifier, variant identifier, datatype wrapper identifier, built-in predicate identifier

22

Data Values

- WSML has direct support for different types of concrete data, namely, strings, integers and decimals, which correspond to the XML Schema primitive datatypes *string*, *integer* and *decimal*
- Concrete values can then be used to construct more complex datatypes, corresponding to other XML Schema primitive and derived datatypes, using datatype constructor functions
- Examples:

```
xsd#date(2005,3,12)
xsd#boolean("true")

age ofType xsd#integer
hasChildren ofType xsd#boolean
```

23

Internationalized Resource Identifiers

- The *IRI* (Internationalized Resource Identifier) mechanism provides a way to identify resources
  - IRIs may point to resources on the Web (in which case the IRI can start with 'http://'), but this is not necessary (e.g. books can be identified through IRIs starting with 'urn:isbn:')

```
_"http://example.org/PersonOntology#Human"
_"http://www.uibk.ac.at/"
```

- An IRI can be abbreviated to a Compact URI - it consists of two parts, namely, the namespace prefix and the local part
  - A Compact URI is written using a namespace prefix and a localname, separated by a hash ('#'): namespace_prefix#localname

```
dc#title (http://purl.org/dc/elements/1.1#title)
foaf#name (http://xmlns.com/foaf/0.1/name)
xsd#string (http://www.w3.org/2001/XMLSchema#string)
Person (http://example.org/#Person)
hasChild (http://example.org/#hasChild)
```

24

Anonymous Identifiers

STI · INNSBRUCK

- An anonymous identifier represents an IRI which is meant to be globally unique
  - Unnumbered anonymous IDs are denoted with '_#'. Each occurrence of '_#' denotes a new anonymous ID and different occurrences of '_#' are unrelated
  - Numbered anonymous IDs are denoted with '_#*n*' where *n* stands for an integer denoting the number of the anonymous ID

```
_#[a hasValue _#1] and _#1 memberOf b.
_#1[a hasValue _#] and _# memberOf _#.
```

- Anonymous identifiers are disallowed for the following elements:
  - the top-level elements *ontology*, *goal*, *webService*, *ooMediator*, *ggMediator*, *wgMediator* and *wwMediator*
  - *capability*, *interface*, *choreography* and *orchestration*

Variables

STI · INNSBRUCK

- Variable names start with an initial question mark "?"
- May only occur in place of concepts, attributes, instances, relation arguments or attribute values
  - A variable may not, for example, replace a WSML keyword however
- Variables may only be used:
  - Inside of logical expressions, and
  - As values in a *non-functional property* definition or in the *sharedVariables* block within capability definitions
- Examples: ?x, ?y1, ?myVariable

WSML Prologue

STI · INNSBRUCK

- WSML Prologue contains all those elements that are in common between all types of WSML specifications and all WSML variants:
  - WSML Variant e.g. the WSML variant reference for a WSML-Flight specification is:

```
wsmlVariant _http://www.wsmo.org/wsml/wsml-syntax/wsml-flight
```

  - Namespace References, e.g.

```
namespace {_"http://www.example.org/ontologies/example#",
          dc _"http://purl.org/dc/elements/1.1#",
          foaf _"http://xmlns.com/foaf/0.1#",
          wsml _"http://www.wsmo.org/wsml-syntax#",
          loc _"http://www.wsmo.org/ontologies/location#",
          oo _"http://example.org/ooMediator#"}
namespace _"http://www.example.org/ontologies/example#"
```

  - WSML header - consists of items that any WSML specification may have: *annotations*, *import ontologies* and *use mediators*

WSML header

STI · INNSBRUCK

- Annotations

```
annotations
  dc#title hasValue "WSML example ontology"
  dc#subject hasValue "family"
  dc#description hasValue "fragments of a family ontology to provide WSML examples"
  dc#contributor hasValue { _"http://homepage.uibk.ac.at/~c703240/foaf.rdf",
            _"http://homepage.uibk.ac.at/~csaa5569/",
            _"http://homepage.uibk.ac.at/~c703239/foaf.rdf",
            _"http://homepage.uibk.ac.at/homepage/~c703319/foaf.rdf" }
  dc#date hasValue xsd#date("2004-11-22")
  dc#format hasValue "text/html"
  dc#language hasValue "en-US"
  dc#rights hasValue _"http://www.deri.org/privacy.html"
  wsml#version hasValue "$Revision: 1.238 $"
endAnnotations
```

- Importing Ontologies

```
importsOntology {_"http://www.wsmo.org/ontologies/location", _"http://xmlns.com/foaf/0.1"}
```

- Using Mediators

```
usesMediator _"http://example.org/ooMediator"
```

Ontologies in WSML – Concepts

- WSML allows inheritance of attribute definitions, which means that a concept inherits all attribute definitions of its superconcepts

```
concept Human subConceptOf {Primate, LegalAgent}
  annotations
        dc#description hasValue "concept of a human being"
        dc#relation hasValue humanDefinition
  endAnnotations
  hasName ofType foaf#name
  hasRelative impliesType Human
  hasAncestor impliesType Human
  hasParent impliesType Human
  hasChild impliesType Human
  hasWeightInKG ofType xsd#float
  hasBirthdate ofType xsd#date
  dateOfDeath ofType xsd#date
  hasBirthplace ofType loc#location
  isMarriedTo impliesType Human
  hasCitizenship ofType oo#country
```

- Axioms can be used to define concepts, e.g.

```
axiom humanDefinition
  definedBy
        ?x memberOf Human equivalent ?x memberOf Primate and ?x memberOf LegalAgent.
```


Concepts with Attribute Definitions

- WSML allows two kinds of attribute definitions
  - Constraining definitions: An attribute definition of the form *A ofType D*, where *A* is an attribute identifier and *D* is a concept identifier, is a *constraint* on the values for attribute *A*; If the value for the attribute *A* is not known to be of type *D*, the constraint is violated and the attribute value is inconsistent with respect to the ontology
  - Inferring definitions: An attribute definition of the form *A impliesType D*, where *A* is an attribute identifier and *D* is a concept identifier, *implies* membership of the concept *D* for all values of the attribute *A*
- Attributes which do not have a datatype range can be specified as being reflexive, transitive, symmetric, or being the inverse of another attribute
- WSML allows the specification of cardinality and range constraints (defined like integrity constraints in databases)

Concepts with Attribute Definitions – Example

```
concept Human
  annotations
        dc#description hasValue "concept of a human being"
  endAnnotations
  hasName ofType foaf#name
  hasRelative symmetric impliesType Human
  hasAncestor transitive impliesType Human
  hasParent inverseOf(hasChild) subAttributeOf(hasAncestor) impliesType Human
  hasMother ofType FemaleHuman
  hasMother impliesType Mother
  hasChild subAttributeOf(hasRelative) impliesType Human
  hasWeightInKG ofType (1) xsd#float
  hasBirthdate ofType (1) xsd#date
  dateOfDeath ofType (0 1) xsd#date
  hasBirthplace ofType (1) loc#location
  isMarriedTo symmetric impliesType (0 1) Human
  hasCitizenship ofType oo#country
```


Ontologies in WSML – Instances

- The *memberOf* keyword identifies the concept to which the instance belongs, e.g.

```
instance Mary memberOf {Parent, Woman}
  annotations
        dc#description hasValue "Mary is parent of the twins Paul and Susan"
  endAnnotations
  hasName hasValue "Maria Smith"
  hasBirthdate hasValue xsd#date(1949,9,12)
  hasChild hasValue {Paul, Susan}
```

- WSML allows instances which are not members of a particular concept, e.g.

```
instance Mary
  hasName hasValue "Maria Smith"
```

Ontologies in WSML – Relations

- Relations in WSML can be used in order to model interdependencies between several concepts (respectively instances of these concepts)
- The parameters of a relation are strictly ordered and their domain can be optionally specified using the keyword *impliesType* or *ofType*
- Examples of relations and relation instances:

```
relation distanceInKm (ofType City, ofType City, impliesType _decimal) subRelationOf measurement
relation distanceInKm/3

relationInstance distance(Innsbruck, Munich, 234)
```

33

Ontologies in WSML – Axioms

- The usage of axioms in WSML allows for example to refine the definition already given in the conceptual syntax, e.g.

```
axiom humanDefinition
  definedBy
        ?x memberOf Human equivalent ?x memberOf Animal and ?x memberOf LegalAgent.
```

- WSML allows the specification of database-style constraints, e.g.

```
axiom humanBMIConstraint
  definedBy
        !- naf bodyMassIndex[bmi hasValue ?b, length hasValue ?l, weight hasValue ?w]
          and ?x memberOf Human and
          ?x[length hasValue ?l,
                weight hasValue ?w,
                bmi hasValue ?b].
```

34

Web Service Capabilities in WSML

- The desired and provided functionality of services are described in WSML in the form of capabilities
  - The desired capability is part of a goal and the provided capability is part of a Web service
- Core elements of capabilities:
  - Shared variables: the variables which are shared between the preconditions, postconditions, assumptions and effects
  - Preconditions: conditions on the inputs of the service
  - Postconditions: the relation between the input and the output of the service
  - Assumptions: what must hold (but cannot be checked beforehand) of the state of the world for the Web service to be able to execute successfully
  - Effects: the real-world effects of the execution of the Web service which are not reflected in the output
- A WSML goal or Web service may only have one capability

35

Web Service Capabilities in WSML – Example

```
capability
  sharedVariables ?child
  precondition
    annotations
          dc#description hasValue "The input has to be boy or a girl
                  with birthdate in the past and be born in Germany."
    endAnnotations
    definedBy
      ?child memberOf Child
          and ?child[hasBirthdate hasValue ?birthdate]
          and ?child[hasBirthplace hasValue ?location]
          and ?location[locatedIn hasValue oo#de]
          or (?child[hasParent hasValue ?parent]
                  and ?parent[hasCitizenship hasValue oo#de] ) .
  effect
    annotations
          dc#description hasValue "After the registration the child is a German citizen"
    endAnnotations
    definedBy
      ?child memberOf Child
      and ?child[hasCitizenship hasValue oo#de].
```

36

Web Service Interfaces in WSML

STI · INNSBRUCK

- An interface describes how the functionality of the Web service can be achieved by providing a twofold view on the operational competence of the Web service: Choreography decomposes a capability in terms of interaction with the Web service; and Orchestration decomposes a capability in terms of functionality required from other Web services
- Basic mechanism for representing choreographies:
  - A signature defines predicates and functions to be used in the description. Ground facts specify the underlying database states.
    - Signatures are defined using *ontologies*
    - The ground facts that populate database states are *instances* of concepts and relations defined by the ontologies
    - State changes are described in terms of creation of new instances or changes to attribute values of objects.
  - State changes are described using transition rules, which specify how the states change by falsifying (deleting) some previously true facts and inserting (making true) some other facts
    - **if** *Condition* **then** *Rules*
    - **forall** *Variables* **with** *Condition* **do** *Rules*
    - **choose** *Variables* **with** *Condition* **do** *Rules*

Interfaces in WSML – Examples

STI · INNSBRUCK

```
interface
  choreography _"http://example.org/mychoreography"
  orchestration _"http://example.org/myorchestration"

interface {_"http://example.org/mychoreography", _"http://example.org/mychoreography"}

interface buyInterface
  choreography buyChoreography
    annotations
              dc#title hasValue "Multimedia Shopping Service Choreography"
              dc#description hasValue "Describes the steps required for shopping
                 multimedia items over this web service"
    endAnnotations

    stateSignature

              importsOntology {
                _"http://example.org/ontologies/products/Products",
                _"http://example.org/ontologies/tasks/ShoppingTasks",
                _"http://example.org/ontologies/shopping/Shopping",
                _"http://example.org/ontologies/products/MediaProducts",
                _"http://example.org/ontologies/Media" }

              in
                 shoptasks#SearchCatalog withGrounding
              _"http://example.org/webservices/shopping/mediashoppingservice#wsdl.interfaceMessageReference(MediaShoppingServicePortType/SearchCatalog/In)"
              out
                 mediaproduct#MediaProduct withGrounding
              _"http://example.org/webservices/shopping/mediashoppingservice#wsdl.interfaceMessageReference(MediaShoppingServicePortType/SearchCatalog/Out)"
```


Interfaces in WSML – Example (cont')

STI · INNSBRUCK

```
transitionRules
  forall ?search
    with      (?search[
                byTitle hasValue ?title,
                byArtist hasValue ?artist,
                byMinPrice hasValue ?minPrice,
                byMaxPrice hasValue ?maxPrice,
                byMinRating hasValue ?minRating,
                byMaxRating hasValue ?maxRating
              ] memberOf shoptasks#SearchCatalog
              and ?artist memberOf media#Artist
              and exists?item(
                ?item memberOf mediaproduct#MediaProduct and(
                ?item[hasContributor hasValue ?artist] or
                ?item[hasTitle hasValue ?title] or
                (?item[hasPrice hasValue ?price] and
                  ?price >= ?minPrice and
                  ?price =< ?maxPrice) or
                ( ?item[hasRating hasValue ?rating] and
                  ?rating >= ?minRating and
                  ?rating =< ?maxRating ) ) ) )
    do
              add(?item[
                hasContributor hasValue ?artist,
                hasTitle hasValue ?title,
                hasPrice hasValue ?price,
                hasRating hasValue ?rating ] memberOf mediaproduct#MediaProduct )
              add(?artist memberOf media#Artist)
              delete(?search memberOf shoptasks#SearchCatalog)
    endForall
```


Non-functional Properties in WSML

STI · INNSBRUCK

- Properties which strictly belong to a Web service, goal, capability, interface or mediator and which are not functional and behavioral
- A WSML Web service, goal, capability, interface or mediator may specify multiple non-functional properties
- Example:

```
nonFunctionalProperty
  po#Price hasValue ?price
    annotations
              dc#description hasValue "If the client is older than 60 or younger than
                        10 years old the invocation price is lower than 10 euro"
    endAnnotations
    definedBy
      ?client[age hasValue ?age] memberOf hu#human and ?age[amount hasValue ?years, units hasValue hu#YearsDuration]
      memberOf hu#age and (greaterEqual(?years, 60) or lessEqual(?years,10))
      implies ?price[hasAmount hasValue ?amount, hasCurrency hasValue cur#Euro] memberOf
      po#AbsoultePrice and lessEqual(?amount, 10).
```

Goals, Web services, and Mediators in WSML

STI · INNSBRUCK

- A WSML goal specification is identified by the **goal** keyword optionally followed by an IRI which serves as the identifier of the goal, e.g.

  goal _"http://example.org/Germany/GetCitizenShip"
- A WSML Web service specification is identified by the webService keyword optionally followed by an IRI which serves as the identifier of the Web service, e.g.

  webService _"http://example.org/Germany/BirthRegistration"
- WSML allows for the specification of four kinds of mediators, namely ontology mediators, mediators between Web services, mediators between goals and mediators between Web services and goals, e.g.

  ooMediator _"http://example.org/ooMediator"

41

Logical Expressions in WSML

STI · INNSBRUCK

- Logical expressions occur within axioms and the capabilities which are specified in the descriptions of goals and Semantic Web services
- We give a syntax specification for general logical expressions in WSML; the general logical expression syntax encompasses all WSML variants and is thus equivalent to the WSML-Full logical expression syntax
  - Vocabulary
  - Terms
  - Set of Fromulae
- We specify for each of the WSML variants the restrictions the variant imposes on the logical expression syntax

42

Logical Expressions – Vocabulary

STI · INNSBRUCK

Definition. A vocabulary V of a WSML language $L(V)$ consists of the following:

- A set of identifiers V_{ID}.
- A set of object constructors $V_O \subseteq V_{ID}$.
- A set of function symbols $V_F \subseteq V_O$.
- A set of datatype wrappers $V_D \subseteq V_O$.
- A set of data values $V_{DV} \subseteq V_O$ which encompasses all string, integer and decimal values.
- A set of anonymous identifiers $V_A \subseteq V_O$ of the form _#, _#1, _#2, etc....
- A set of relation identifiers $V_R \subseteq V_{ID}$.
- A set of variable identifiers $V_V \subseteq V_{ID}$ of the form ?*alphanum*+.

- WSML allows
  - logical connectives: and, or, implies, impliedBy, equivalent, neg, naf, forall and exists and the following auxiliary symbols: '(', ')', '[', ']', ',', '=', '!=', memberOf, hasValue, subConceptOf, ofType, and impliesType
  - use of the symbol ':-' for Logic Programming rules and the use of the symbol '!-' for database-style constraints
  - use of true and false as predicates with the standard pre-defined meaning

43

Logical Expressions – Terms

STI · INNSBRUCK

Definition. Given a vocabulary V, the set of terms $Term(V)$ in WSML is defined as follows:

- Any $f \in V_O$ is a term.
- Any $v \in V_V$ is a term.
- If $f \in V_F$ and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.
- If $f \in V_D$ and $dv_1, \ldots, dv_n$ are in $V_{DV} \cup V_V$, then $f(dv_1, \ldots, dv_n)$ is a term.

Definition. Given a set of terms $Term(V)$, the set of atomic formulae in $L(V)$ is defined by:

- If $\alpha, \beta \in Term(V)$ and $\gamma \in Term(V)$ or γ is of the form $\{\gamma_1, \ldots \gamma_n\}$, with $\gamma_1, \ldots \gamma_n \in Term(V)$, then:
  - α **subConceptOf** γ is an atomic formula in $L(V)$. Here, α and γ both identify concepts.
  - α **memberOf** γ is an atomic formula in $L(V)$. Here, α identifies an instance and gamma identifies (a) concept(s).
  - $\alpha[\beta$ **ofType** $\gamma]$ is an atomic formula in $L(V)$. Here, α identifies a concept, β identifies an attribute and γ identifies (a) concept(s).
  - $\alpha[\beta$ **impliesType** $\gamma]$ is an atomic formula in $L(V)$. Here, α identifies a concept, β identifies an attribute and γ identifies (a) concept(s).
  - $\alpha[\beta$ **hasValue** $\gamma]$ is an atomic formula in $L(V)$. Here, α identifies an instance, β identifies an attribute and γ identifies (an) instance(s).

  These atomic formulae are also called *molecules*. Notice that an identifier can play a different role (instance, attribute, concept), depending on the use in a molecule.
- If $r \in V_R$ and $t_1, \ldots, t_n$ are terms, then $r(t_1, \ldots, t_n)$ is an atomic formula in $L(V)$.
- If $\alpha, \beta \in Term(V)$ then α **=** β and α **!=** β are atomic formulae in $L(V)$.

44

Logical Expressions in WSML – Set of Fromulae

STI · INNSBRUCK

Definition. The set of formulae in $L(V)$ is defined by:

- Every atomic formula in $L(V)$ is a formula in $L(V)$.
- Let α, β be formulae which do not contain the symbols '**:-**' and '**!-**', and let $?x_1,\ldots,?x_n$ be variables, then:
  - α **and** β is a formula in $L(V)$.
  - α **or** β is a formula in $L(V)$.
  - **neg** α is a formula in $L(V)$.
  - **naf** α is a formula in $L(V)$.
  - **forall** $?x_1,\ldots,?x_n$ (α) is a formula in $L(V)$.
  - **exists** $?x_1,\ldots,?x_n$ (α) is a formula in $L(V)$.
  - α **implies** β is a formula in $L(V)$.
  - α **impliedBy** β is a formula in $L(V)$.
  - α **equivalent** β is a formula in $L(V)$.
  - α **:-** β is a formula in $L(V)$. This formula is called an *LP (Logic Programming) rule*. α is called the *head* and β is called the *body* of the rule.
  - **!-** α is a formula in $L(V)$. This formula is called a *constraint*. We say α is a constraint of the knowledge base.

Examples of WSML Logical Expressions

STI · INNSBRUCK

- No human can be both male and female:

  !- ?x[gender **hasValue** {?y, ?z}] **memberOf** Human **and** ?y = Male **and** ?z = Female.

- A human who is not a man is a woman:

  ?x[gender **hasValue** Woman] **impliedBy neg** ?x[gender **hasValue** Man].

- The brother of a parent is an uncle:

  ?x[uncle **hasValue** ?z] **impliedBy** ?x[parent **hasValue** ?y] **and** ?y[brother **hasValue** ?z].

- Do not trust strangers:

  ?x[distrust **hasValue** ?y] :- naf ?x[knows **hasValue** ?y].

WSML-Core Logical Expressions Restrictions

STI · INNSBRUCK

- WSML-Core is based on the Logic Programming subset of Description Logics
- WSML-Core allows only a restricted form of logical expressions; there are two sources for these restrictions
  - The restriction of the language to a subset of Description Logics restricts the kind of formulas which can be written down to the *two-variable fragment of first-order logic*. Furthermore, it disallows the use of *function symbols* and restricts the arity of predicates to *unary* and *binary* and chaining variables over predicates
  - The restriction of the language to a subset of Datalog (without equality) disallows the use of the *equality symbol*, *disjunction* in the head of a rule and *existentially quantified variables* in the head of the rule

WSML-Core – Other Restrictions

STI · INNSBRUCK

- WSML-Core does not allow for the specification of the attribute features **reflexive**, **transitive**, **symmetric**, **inverseOf** and **subAttributeOf**
- Cardinality constraints are not allowed and thus it is not possible to specify functional properties
- Allowed attribute values are restricted to strings, numbers, ids, or a lists of such values; constructed data values are not allowed
- Does not allow for the specification of relations
- Does not allow the specification of relation instances
- For axioms, WSML-Core only allows the use of a restricted form of the WSML logical expression syntax
- Goals and Web services: the logical expressions in the 'assumptions', 'preconditions', 'effects' and 'postconditions' of a capability and 'definition' of a non-functional property are limited to WSML-Core logical expressions

WSML-Core Logical Expressions Examples

- The attribute 'hasAncestor' is transitive:

  ?x[hasAncestor **hasValue** ?z] **impliedBy**
  ?x[hasAncestor **hasValue** ?y] **and** ?y[hasAncestor **hasValue** ?z]

- A female person is a woman:

  ?x **memberOf** Woman **impliedBy** ?x **memberOf** Person **and** ?x **memberOf** Female

- A student is a person:

  Student **subConceptOf** Person

49

WSML-DL Logical Expressions Restrictions

- WSML-DL is an extension of WSML-Core to a full-fledged description logic with an expressiveness similar to OWL DL
- WSML-DL is both syntactically and semantically completely layered on top of WSML-Core
  - Every valid WSML-Core specification is also a valid WSML-DL specification
  - All consequences inferred from a WSML-Core specification are also valid consequences of the same specification in WSML-DL
- The difference between WSML-Core and WSML-DL lies in the logical expression syntax
  - The logical expression syntax of WSML-DL is less restrictive than the logical expression syntax of WSML-Core
- Restrictions: it disallows the use of *function symbols*, restricts the arity of predicates to *unary* and *binary* and prohibits *chaining variables over predicates*

50

WSML-DL Logical Expressions Examples

- The concept Human is defined as the disjunction between Man and Woman:

  ?x **memberOf** Human **equivalent** ?x **memberOf** Woman **or** ?x **memberOf** Man

- The concepts Man and Woman are disjoint:

  ?x **memberOf** Man **implies neg** ?x **memberOf** Woman

- Every Human has a father, which is a Human and every father is a human:

  ?x **memberOf** Human **implies exists** ?y (?x[father **hasValue** ?y] **and** ?y **memberOf** Human) **and forall** ?y (?x[father **hasValue** ?y] **implies** ?y **memberOf** Human)

51

WSML-Flight

- WSML-Flight is both syntactically and semantically completely layered on top of WSML-Core
- WSML-Flight adds the following features to WSML-Core:
  - N-ary relations with arbitrary parameters
  - Constraining attribute definitions for the abstract domain
  - Cardinality constraints
  - (Locally Stratified) default negation in logical expressions (in the body of the rule)
  - Expressive logical expressions, namely, the full Datalog subset of F-Logic, extended with inequality (in the body) and locally stratified negation
  - Meta-modeling. WSML-Flight no longer requires a separation of vocabulary (wrt. concepts, instances, relations)

52

WSML-Flight Logical Expressions Examples

- No human can be both male and female:

!- ?x[gender **hasValue** {?y, ?z}] **memberOf** Human **and** ?y = Male **and** ?z = Female

- The brother of a parent is an uncle:

?x[uncle **hasValue** ?z] **impliedBy** ?x[parent **hasValue** ?y] **and** ?y[brother **hasValue** ?z]

- Do not trust strangers:

?x[distrust **hasValue** ?y] :- naf ?x[knows **hasValue** ?y] and ?x memberOf Human and ?y memberOf Human

53

WSML-Rule

- WSML-Rule is an extension of WSML-Flight in the direction of Logic Programming
- WSML-Rule no longer requires safety of rules and allows the use of function symbols
  - The only differences between WSML-Rule and WSML-Flight are in the logical expression syntax
- WSML-Rule is both syntactically and semantically layered on top of WSML-Flight
  - Each valid WSML-Flight specification is a valid WSML-Rule specification

54

WSML-Rule Logical Expressions Examples

- Both the father and the mother are parents:

?x[parent **hasValue** ?y] **:-** ?x[father **hasValue** ?y] or ?x[mother **hasValue** ?y]

- Every person has a father:

?x[father **hasValue** f(?x)] **:-** ?x **memberOf** Person

- There may only be one distance between two locations, and the distance between locations *A* and *B* is the same as the distance between *B* and *A*:

!- distance(?location1,?location2,?distance1) **and** distance(?location1,?location2,?distance2) **and** ?distance1 != distance2
distance(?B,?A,?distance) **:-** distance(?A,?B,?distance)

55

WSML-Full

- WSML-Full unifies the Description Logic and Logic Programming variants of WSML, namely, WSML-DL and WSML-Rule, in a principled way, under a common syntactic umbrella, inspired by First-Order Logic
- Differences between WSML-DL and WSML-Full:
  - WSML-Full adds full first-order modeling: n-ary predicates, function symbols and chaining variables over predicates
  - WSML-Full allows non-monotonic negation
- Differences between WSML-Rule and WSML-Full:
  - WSML-Full adds disjunction, classical negation, multiple model semantics, and the equality operator

56

WSML Variants and Feature Matrix – Summary

STI · INNSBRUCK

Feature	Core	DL	Flight	Rule	Full
Classical Negation (**neg**)	-	X	-	-	X
Existential Quantication	-	X	-	-	X
(Head) Disjunction	-	X	-	-	X
n-ary relations	-	-	X	X	X
Meta Modeling	-	-	X	X	X
Default Negation (**naf**)	-	-	X	X	X
LP implication	-	-	X	X	X
Integrity Constraints	-	-	X	X	X
Function Symbols	-	-	-	X	X
Unsafe Rules	-	-	-	X	X

57

WSML2Reasoner

STI · INNSBRUCK

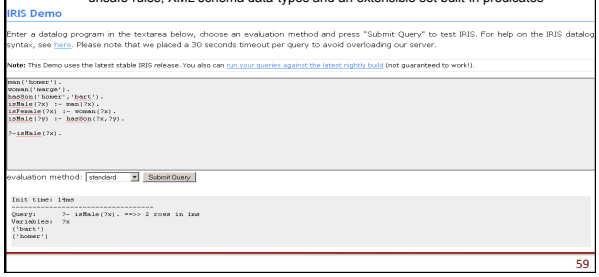
- WSML2Reasoner is a highly modular framework that combines various validation, normalization and transformation algorithms that enable the translation of ontology descriptions in WSML to the appropriate syntax of several underlying reasoning engines
- WSML2Reasoner has been implemented and tested using various reasoning engines that support all WSML variants with the exception of WSML-Full
- The WSML2Reasoner framework has a flexible architecture that allows the easy integration of existing reasoning components
  - It first maps WSML to either generic Datalog or OWL (Web Ontology Language), and then uses a plug-in layer that translates to the specific internal representation of a single reasoning engine

58

Reasoning engines integrated within the WSML2Reasoner framework

STI · INNSBRUCK

- Integrated Rule Inference System (IRIS) - http://iris-reasoner.org/
  - Supports WSML-Core, WSML-Flight, WSML-Rule
  - Is an extensible reasoning engine for expressive rule-based languages; supports Datalog extended with stratified and well-founded negation, function symbols, unsafe-rules, XML schema data-types and an extensible set built-in predicates



59

Reasoning engines integrated within the WSML2Reasoner framework (cont')

STI · INNSBRUCK

- KAON2
  - Supports WSML-Core, WSML-Flight, WSML-DL
  - Is an infrastructure for managing OWL-DL, SWRL, and F-Logic ontologies
  - Reasoning in KAON2 is implemented by novel algorithms which reduce a SHIQ(D) knowledge base to a disjunctive datalog program
- MINS (Mins Is Not Silri)
  - Supports WSML-Core, WSML-Flight, WSML-Rule
  - Is a reasoner for Datalog programs with negation and function symbols which supports well-founded semantics
- PELLET
  - Supports WSML-Core, WSML-DL
  - Is an OWL DL reasoner written in Java
  - Is based on the tableaux algorithms for expressive Description Logics (DL) and supports the full expressivity of OWL DL, including reasoning about nominals (enumerated classes)

60

ILLUSTRATION BY A LARGER EXAMPLE

Vehicle Ontology

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.sti-innsbruck.at/ontologies/vehicle#" }

ontology VehicleOntology
concept Vehicle
  hasTires ofType _integer
  hasMotor ofType _boolean
concept Automobile subConceptOf Vehicle
  hasLimitSpeed ofType _integer
concept Car subConceptOf Automobile
  hasDoors ofType _integer
concept TwoWheeler subConceptOf Vehicle
concept Motorcycle subConceptOf {Automobile, TwoWheeler}
concept Bicycle subConceptOf TwoWheeler

axiom definedBy
  // a vehicle with a motor is an automobile
  ?x memberOf Vehicle and ?x[hasMotor hasValue _boolean("true")] implies ?x memberOf Automobile.
  // an automobile with 4 tires is a car
  ?x memberOf Automobile and ?x[hasTires hasValue 4] implies ?x memberOf Car.
  // a vehicle with 2 tires is a TwoWheeler
  ?x memberOf Vehicle and ?x[hasTires hasValue 2] implies ?x memberOf TwoWheeler.
  // a two-wheeler with motor is a motorcycle
  ?x memberOf TwoWheeler and ?x[hasMotor hasValue _boolean("true")]
      implies ?x memberOf Motorcycle.
  // a two-wheeler without motor is a bicycle.
  ?x memberOf TwoWheeler and ?x[hasMotor hasValue _boolean("false")] implies ?x memberOf Bicycle.
```


Find An Automobile Goal

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.sti-innsbruck.at/goals#",
vehicle _"http://www.sti-innsbruck.at/ontologies/vehicle#",
discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#"}

goal FindAnAutomobile

importsOntology {vehicle#VehicleOntology}

capability FindAnAutomobileCapability

nonFunctionalProperties
  discovery#discoveryStrategy hasValue discovery#HeavyweightDiscovery
  discovery#discoveryStrategy hasValue discovery#NoPreFilter
endNonFunctionalProperties

sharedVariables ?x

precondition findAnAutomobilePre
definedBy
?x memberOf vehicle#Vehicle and ?x[vehicle#hasMotor hasValue _boolean("true")].

postcondition findAnAutomobilePost
definedBy
   ?x memberOf vehicle#Automobile.
```


Automobile Vendor Web Service

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.sti-innsbruck.at/services#",
vehicle _"http://www.sti-innsbruck.at/ontologies/vehicle#",
discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#" }

webService AutomobileVendor

importsOntology {vehicle#VehicleOntology}

capability AutomobileCapability

nonFunctionalProperties
  discovery#discoveryStrategy hasValue discovery#HeavyweightDiscovery
  discovery#discoveryStrategy hasValue discovery#NoPreFilter
endNonFunctionalProperties

sharedVariables ?x

precondition automobilePre
definedBy
?x memberOf vehicle#Vehicle and ?x[vehicle#hasMotor hasValue _boolean("true")].

postcondition automobilePost
definedBy
   ?x memberOf vehicle#Automobile.
```

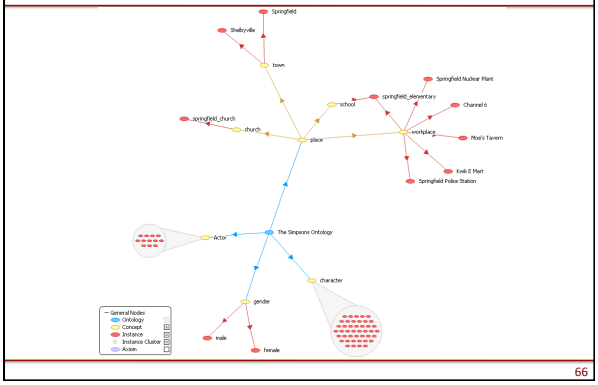
Car Vendor Web Service

STI · INNSBRUCK

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.sti-innsbruck.at/services#",
vehicle _"http://www.sti-innsbruck.at/ontologies/vehicle#",
discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#" }

webService CarVendor

importsOntology {vehicle#VehicleOntology}

capability CarCapability

nonFunctionalProperties
  discovery#discoveryStrategy hasValue discovery#HeavyweightDiscovery
  discovery#discoveryStrategy hasValue discovery#NoPreFilter
endNonFunctionalProperties

sharedVariables ?x

precondition carPre
definedBy
?x memberOf vehicle#Automobile.

postcondition carPost
definedBy
  ?x memberOf vehicle#Automobile and ?x[vehicle#hasTires hasValue 4].
```


Another Example – The Simpsons Ontology (visualized using WSMT)

STI · INNSBRUCK



The Simpsons Ontology – Preamble and Axioms

STI · INNSBRUCK

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://ontologies.sti2.at/",
    wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
    dc _"http://purl.org/dc/elements/1.1/" }

ontology simpsons
    nonFunctionalProperties
        dc#creator hasValue "Mick Kerrigan"
        dc#type hasValue _"http://www.wsmo.org/2004/d2#ontologies"
        dc#description hasValue "An ontology of the characters, actors and places in the Simpsons"
        dc#identifier hasValue simpsons
        dc#publisher hasValue "STI Innsbruck"
        dc#subject hasValue "The Simpsons"
        wsml#version hasValue "1.0"
        dc#language hasValue "en-UK"
        dc#title hasValue "The Simpsons Ontology"
        dc#date hasValue "2006-05-03"
    endNonFunctionalProperties

axiom priniciples_work_too
    definedBy
        ?x[principleOf hasValue ?y] memberOf character
                implies ?x[hasWorkingPlace hasValue ?y].

axiom spouses_are_in_love
    definedBy
        ?x[hasSpouse hasValue ?y] memberOf character
                implies ?x[inLoveWith hasValue ?y].
```


The Simpsons Ontology - Concepts

STI · INNSBRUCK

```
concept gender

concept actor
    nonFunctionalProperties
        dc#title hasValue "Actor"
    endNonFunctionalProperties
    hasName ofType _string
    hasGender ofType gender

concept place
    hasName ofType _string

concept town subConceptOf place
    hasMayor ofType character
    hasPoliceChief ofType character

concept workplace subConceptOf place
    hasOwner ofType character
    hasLocation ofType town

concept school subConceptOf place
    hasPrinciple ofType character
    hasLocation ofType town

concept church subConceptOf place
    hasReverant ofType character
    hasLocation ofType town

concept character
    hasName ofType _string
    hasGender ofType gender
    hasActor ofType actor
    hasSpouse ofType character
    hasChild ofType character
    hasParent ofType character
    hasSibling ofType character
    hasFriend ofType character
    hasNeighbour ofType character
    hasCatchPhrase ofType _string
    inLoveWith ofType character
    isCustomerOf ofType workplace
    hasWorkingPlace ofType place
    worshipsAt ofType church
    principleOf ofType school
    attends ofType school
    owns ofType workplace
    mayorOf ofType town
    policeChiefOf ofType town
    reverantOf ofType church
```

The Simpsons Ontology – (some) Instances

STI · INNSBRUCK

```
instance male memberOf gender

instance female memberOf gender

instance dan_castellanata memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Dan Castellanata"
    endNonFunctionalProperties
    hasName hasValue "Dan Castellaneta"
    hasGender hasValue male

instance julie_kavner memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Julie Kavner"
    endNonFunctionalProperties
    hasName hasValue "Julie Kavner"
    hasGender hasValue female

instance yeardley_smith memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Yeardley Smith"
    endNonFunctionalProperties
    hasName hasValue "Yeardley Smith"
    hasGender hasValue female

instance nancy_cartwright memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Nancy Cartwright"
    endNonFunctionalProperties
    hasName hasValue "Nancy Cartwright"
    hasGender hasValue female

instance hank_azaria memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Hank Azaria"
    endNonFunctionalProperties
    hasName hasValue "Hank Azaria"
    hasGender hasValue male

instance harry_shearer memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Harry Shearer"
    endNonFunctionalProperties
    hasName hasValue "Harry Shearer"
    hasGender hasValue male

instance marcia_wallace memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Marcia Wallace"
    endNonFunctionalProperties
    hasName hasValue "Marcia Wallace"
    hasGender hasValue female

instance pamela_hayden memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Pamela Hayden"
    endNonFunctionalProperties
    hasName hasValue "Pamela Hayden"
    hasGender hasValue female

instance tress_macneille memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Tress MacNeille"
    endNonFunctionalProperties
    hasName hasValue "Tress MacNeille"
    hasGender hasValue female

instance maggie_roswell memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Maggie Roswell"
    endNonFunctionalProperties
    hasName hasValue "Maggie Roswell"
    hasGender hasValue female

instance doris_grau memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Doris Grau"
    endNonFunctionalProperties
    hasName hasValue "Doris Grau"
    hasGender hasValue female
instance kelsey_grammar memberOf actor
    nonFunctionalProperties
        dc#title hasValue "Kelsey Grammar"
    endNonFunctionalProperties
    hasName hasValue "Kelsey Grammar"
    hasGender hasValue male

instance springfield memberOf town
    nonFunctionalProperties
        dc#title hasValue "Springfield"
    endNonFunctionalProperties
    hasName hasValue "Springfield"
    hasMayor hasValue joe_quimby
    hasPoliceChief hasValue chief_clancy_wiggum

instance shelbyville memberOf town
    nonFunctionalProperties
        dc#title hasValue "Shelbyville"
    endNonFunctionalProperties
    hasName hasValue "Shelbyville"
```


The Simpsons Ontology – Queries

STI · INNSBRUCK

Query 1: To find all the students and the principle of the school they attend:

?child[attends hasValue ?school] and ?principle[principleOf hasValue ?school]

Query 2: To find all the people that work at springfield_elementary:

?employee[hasWorkingPlace hasValue springfield_elementary]

Query 3: To find out who is the reverant of the church that each character attends:

?worshiper[worshipsAt hasValue ?church] and ?reverant[reverantOf hasValue ?church]

Query 4: To find all the characters that Dan Castellanata does the voices for on the simpsons:

?character[hasActor hasValue dan_castellanata]

Query 5: To find out who is in love with who in the simpsons world:

?character1[inLoveWith hasValue ?charachter2]

Query 6: To find all actors who play characters of the opposite sex:

?actor[hasGender hasValue ?actorgender] memberOf actor and ?character[hasGender hasValue ?charactergender] memberOf character and ?character[hasActor hasValue ?actor] and ?actorgender != ?charactergender

Demo: Use the IRIS Reasoner in WSMT for answers to these queries!

STI · INNSBRUCK

EXTENSIONS

WSML – Recent development

STI · INNSBRUCK

- 1 new variant and 4 updated versions of WSML developed in SOA4All project:
  - WSML-Quark
  - WSML-Core 2.0
  - WSML-DL 2.0
  - WSML-Flight 2.0
  - WSML-Rule 2.0

WSML Updates

- Basic Idea: Update WSML to align it with new research
- WSML-Core 2.0:
  - Still based on DLP ~ Semantically stronger version
  - Extended with individivual equivalence
  - Requires slightly more powerful rules engine
- WSML-DL 2.0:
  - Based on ELP (decidable fragment of the Semantic Web Rule Language (SWRL))
  - Contains OWL 2 RL, EL
  - Allows restricted form of rules to be included
- WSML-Flight/Rule 2.0:
  - Updated for layering
  - Ensure compatibility with RIF (Rule Interchange Format)
  - Technically contains same expressivity as OWL 2 QL

WSML Quark – New Variant

- New language variant
- Specifically for very light-weight knowledge representation
  - classification systems, ultra-light-weight
  - hierarchical schema (sub-concept)
- Reasoning is reduced to graph reachability
  - is-subconcept-of(C1,C2)
  - get-sub-concepts(C)
  - get-super-concepts(C)
- Hybrid algorithms
  - trade off between query time, indexing time and index size
  - tune for specific knowledge bases

SUMMARY

WSML – Summary and Conclusions

- Provides a formal syntax and semantics for representing WSMO Ontologies, Web services, Goals, and Mediators
- Defines several language variants (WSML-Core, WSML-DL, WSML-Flight, WSML-Rule, WSML-Full) and thus allows users to make a trade-off between the provided expressivity and the implied complexity
- Designed based on the principles of the Semantic Web
- Brings together different logical language paradigms and unifies them in one syntactical framework, enabling the reuse of proven reasoning techniques and tools
- WSML2Reasoner is a reasoning framework for WSML
- WSML is evolving

STI · INNSBRUCK

REFERENCES

References

STI · INNSBRUCK

- Mandatory reading:
  - http://www.wsmo.org/TR/d16/d16.1/v1.0/
- Further reading:
  - J. de Bruijn, D. Fensel, U. Keller, M. Kerrigan, H. Lausen, and J. Scicluna. *Modeling Semantic Web Services - The Web Service Modeling Language. Springer, 2008*
  - J. de Bruijn, H. Lausen, A. Polleres, D. Fensel: The Web Service Modeling Language WSML: An Overview. ESWC 2006: 590-604
  - J. de Bruijn, S. Heymans: A Semantic Framework for Language Layering in WSML. RR 2007: 103-117
  - D. Roman, J. de Bruijn, A. Mocan, H. Lausen, J. Domingue, C. Bussler, D. Fensel: WWW: WSMO, WSML, and WSMX in a Nutshell. ASWC 2006: 516-522
  - http://www.wsmo.org/wsml
  - http://www.wsmo.org/TR/d16/
  - http://www.wsmo.org/TR/d16/d16.3/v1.0/
  - http://www.wsmo.org/TR/d32/v1.0/
  - http://www.wsmo.org/TR/d36/v1.0/
  - http://www.wsmo.org/TR/d37/v0.2/
  - http://tools.sti-innsbruck.at/wsml2reasoner/
  - http://sourceforge.net/projects/wsml2reasoner/
  - http://wsmt.sourceforge.net
- Wikipedia links:
  - http://en.wikipedia.org/wiki/WSMO

Next Lecture

STI · INNSBRUCK

#	Title
1	Introduction
2	Web Science
3	Service Science
4	Web services
5	Web2.0 services
6	Semantic Web
7	Web Service Modeling Ontology (WSMO)
8	Web Service Modeling Language (WSML)
9	**Web Service Execution Environment (WSMX)**
10	OWL-S and other
11	Light-weight Annotations
12	Applications
13	Mobile Services

Questions?

STI · INNSBRUCK