Managing Multiple and Distributed Ontologies on the Semantic Web

Zhenning Shangguan
Feb-03-2009
CSCI-6965 Class Presentation
Outline

• Introduction
• Related Work
• OI model
• Ontology Evolution
• Implementation
Introduction

• Three challenges
  – Representation model
  – Ontology reuse and interoperability
  – Ontology evolution

• Contributions
  – Conceptual modeling framework
  – Integrated evolving process
  – Scalable implementation within KAON²
Related work – multiple ontologies

• RDF
  – Freely refer to any resources, impossible to reason over the web

• Other languages (OIL, DAML+OIL, OWL)
  – Declarative importing
  – Tools simply read these files and create an integrated model
  – OilEd, Ontolingua, Protege-2000
Related Work - Evolution

• No commonly agreed methods and guidelines
• SHOE
  – Support versioning, but no change propagation
• CONCORDIA
  – Retired status, special links to track retired parents and children
  – Insufficient for managing changes on the SW
• Script-based evolution
  – Identify typical change sequences, tools understand the consequences of these changes
  – Users unable to control how to complete the overall modification and no suggestion about additional changes that are required or might improve the ontology
Related Work - Evolution

• Change detection
  – Some proposal by comparing ontologies (complicated and time-consuming)
  – We record all performed changes
  – Also able to determine causes and consequences of these changes
  – Coherency maintenance of dependent and replication consistency
OI Model (Ontology)

• Single Ontology

• Multiple Ontologies

• Distributed Ontologies
Single Ontology

• General Principle
  – Object-oriented modeling paradigm + practicality

• Practicality
  – Metamodelling
  – Domain and range
  – Cardinality
Mathematical Definition

• An OI model (ontology instance model) structure is a tuple $OIM := (E, INC)$ where:
  – $E$ is the set of entities of the OI model,
  – $INC$ is the set of included OI models.
Ontology Structure

- An ontology structure of an OI model is a structure $O(OIM) := (C, P, R, S, T, INV, HC, HP, \text{domain}, \text{range}, \text{mincard}, \text{maxcard})$ where:
  - $C \subseteq E$ is a set of concepts
  - $P \subseteq E$ is a set of properties
  - $R \subseteq P$ is a set of relational properties (properties from the set $A = P \setminus R$ are called attribute properties)
  - $S \subseteq R$ is a subset of symmetric properties
  - $T \subseteq R$ is a subset of transitive properties
  - $INV \subseteq R \times R$ is a symmetric relation that relates inverse relational properties; if $(p_1, p_2) \in INV$, then $p_1$ is an inverse relational property of $p_2$
Ontology Structure

- $H_C \subseteq C \times C$ is an acyclic relation called concept hierarchy; if $(c_1, c_2) \in H_C$, then $c_1$ is a subconcept of $c_2$ and $c_2$ is a superconcept of $c_1$

- $H_P \subseteq P \times P$ is an acyclic relation called property hierarchy; if $(p_1, p_2) \in H_P$, then $p_1$ is a subproperty of $p_2$ and $p_2$ is a superproperty of $p_1$

- function domain : $P \rightarrow 2^C$ gives the set of domain concepts for some property $p \in P$

- function range : $R \rightarrow 2^C$ gives the set of range concepts for some relational property $p \in R$

- function mincard : $C \times P \rightarrow N_0$ gives the minimum cardinality for each concept-property pair

- function maxcard : $C \times P \rightarrow (N_0 \cup \{\infty\})$ gives the maximum cardinality for each concept-property pair
Instance Pool Structure

• An instance pool associated with an OI model is a 4-tuple
  \( \text{IP}(\text{OIM}) := (I, L, \text{instconc}, \text{instprop}) \) where:
    – \( I \subseteq E \) is a set of instances
    – \( L \) is a set of literal values \( L \cap E = \emptyset \)
    – function \( \text{instconc} : C \rightarrow 2^I \) relates a concept with a set of its instances
    – function \( \text{instprop} : P \times I \rightarrow 2^{I \cup L} \) assigns to each property instance pair a set of instances related through given property
Root Concept

• Root OI model is defined as a particular, well-known OI model with structure \( \text{ROIM} := (\{\text{kaon:Root}\}, \emptyset) \). \text{kaon:Root} is the root concept; every other concept must subclass \text{kaon:Root} (it may do so indirectly)
Modularization Constraints

- If OI model OIM imports some other OI model OIM₁ (with elements marked with subscript 1), that is, if OIM₁ ∈ INC(OIM), then the following modularization constraints must be satisfied:
  - $E₁ \subseteq E$, $C₁ \subseteq C$, $P₁ \subseteq P$, $R₁ \subseteq R$, $T₁ \subseteq T$, $INV₁ \subseteq INV$, $HC₁ \subseteq HC$, $HP₁ \subseteq HP$
  - $\forall p \in P₁ \text{domain}₁(p) \subseteq \text{domain}(p)$
  - $\forall p \in P₁ \text{range}₁(p) \subseteq \text{range}(p)$
  - $\forall p \in P₁, \forall c \in C₁ \text{mincard}₁(c, p) = \text{mincard}(c, p)$
  - $\forall p \in P₁, \forall c \in C₁ \text{maxcard}₁(c, p) = \text{maxcard}(c, p)$
  - $I₁ \subseteq I$, $L₁ \subseteq L$
  - $\forall c \in C₁ \text{instconc}₁(c) \subseteq \text{instconc}(c)$
  - $\forall p \in P₁, i \in I₁ \text{instprop}₁(p, i) \subseteq \text{instprop}(p, i)$
(Denotational) Semantics

- An interpretation of an OI model OIM is a structure \( I = (\Delta^I, \Delta_D, E^I, L^I, C^I, P^I) \) where:
  - \( \Delta^I \) is the set of object interpretations
  - \( \Delta_D \) is the concrete domain for data types \( \Delta^I \cap \Delta_D = \emptyset \)
  - \( E^I : E \to \Delta^I \) is an entity interpretation function that maps each entity to a single element in a domain
  - \( L^I : L \to \Delta_D \) is a literal interpretation function that maps each literal to an element of the concrete domain
  - \( C^I : \Delta^I \to 2^{\Delta^I} \) is a concept interpretation function by treating concepts as subsets of the domain
  - \( P^I : \Delta^I \to 2^{\Delta^I \times (\Delta^I \cup \Delta_D)} \) is a property interpretation function by treating properties as relations on the domain
Semantics

• Almost the same with RDFS semantics

• Closed world: domain/range, cardinality as constraints

\[
\forall p, c, i \quad c \in \text{domain}(p) \land (\exists x \quad (E^I(i), x) \in P^I(E^I(p))) \land \\
E^I(i) \notin C^I(E^I(c)) \Rightarrow \text{ontology is inconsistent} \\
\forall p, c, i \quad c \in \text{range}(p) \land (\exists x \quad (x, E^I(i)) \in P^I(E^I(p))) \land \\
E^I(i) \notin C^I(E^I(c)) \Rightarrow \text{ontology is inconsistent} \\
\forall p, c, i \quad E^I(i) \in C^I(E^I(c)) \land \text{mincard}(c, p) > \\
|\{ y \mid (E^I(i), y) \in P^I(E^I(p))\}| \Rightarrow \text{ontology is inconsistent} \\
\forall p, c, i \quad E^I(i) \in C^I(E^I(c)) \land \text{maxcard}(c, p) < \\
|\{ y \mid (E^I(i), y) \in P^I(E^I(p))\}| \Rightarrow \text{ontology is inconsistent}
\]
Domain & range

• Usually, domain/range axioms specify sufficient conditions for class membership

\[
\forall p, c, i \quad c \in \text{domain}(p) \land (\exists x \quad (E^I(i), x) \in P^I(E^I(p))) \Rightarrow \\
E^I(i) \in C^I(E^I(c))
\]

\[
\forall p, c, i \quad c \in \text{range}(p) \land (\exists x \quad (x, E^I(i)) \in P^I(E^I(p))) \Rightarrow \\
E^I(i) \in C^I(E^I(c))
\]

• Domain (teaches) = professors
  (Peter_Fox teaches CSCI-6965)
\[\rightarrow\] Peter_Fox \in Professors
Metamodeling

• *In order to allow metaconcepts, the following constraint is stated: C ∩ I may, but does not need to, be Ø. Also, P ∩ I may, but does not need to, be Ø.*

• An element can be either a concept or an instance. E.g., A is an instance of A. (rdfs:Class rdf:type rdfs:Class)
Multiple Ontologies

• Open-closed reuse principle
  – each ontology should be a closed, consistent, and a self-contained entity, but open to extensions in other ontologies

• Ultimate goal
  – eliminate the copy-and-paste reuse, which may be problematic
Ontology Inclusion

• An ontology is able to include other ontologies, obtaining the union of the definitions from all included ones
Distributed Ontologies

• How to reuse ontologies in a distributed environment?

• Two possible solutions:
  – Central server
  – Replication

• Replication eliminates performance problems, but introduces evolution and consistency issues.

SUO – Sports Utility Ontology
BO – Bicycle Ontology
CO – Climbing Ontology
ICO – Integrated Catalog Ontology

original ontology
ontology replica
Ontology Evolution

• General process:
  – Change capturing
  – Change propagation
  – Change implementation
  – Change validation

• Differs according to different levels of ontology evolution
  – Single
  – Multiple
  – Distributed
Single Ontology Evolution

• Essential goal: maintain ontology consistency during evolution

• Elementary changes will not always ensure the consistency of the ontology
  – Induced additional changes are needed
  – E.g., Domain (teaches) = professors
    (Peter Fox teaches CSCI-6965)
    → Peter Fox ∈ Professors
Single Ontology Evolution

• For some change, various sets of induced changes can be generated, all leading to the consistent state

• Resolution points:
  – Answer by users to a list of questions about which action to take at some critical points during evolution
  – E.g., how to handle orphaned concepts? – delete them or re-connect them to some ancestor concept?
Multiple Ontology Evolution

• Dependent ontology consistency
  – A dependent ontology is consistent if the ontology itself and all its included ontologies, observed alone and independently of the ontologies in which they are reused, are single ontology consistent.
Evolution Process

• Two ways to propagate changes
  – **Push-based**: strict consistency requirement
  – Pull-based: less stringent consistency

• Dependent ontology evolution process
Evolution Process

• Ontology propagation order
  – Topological order according to inclusion relation

• Change filtering
  – Only induced changes are forwarded

• Change ordering
  – S→I→D, D should process changes generated by I before changes generated by S
Dependent Ontology Evolution Algorithm

Algorithm 1 Dependent Ontology Evolution Algorithm

\textsc{EvolveOntologies}(\mathcal{LC}, o)

\textbf{Require:} $\mathcal{LC}$ - list of changes, $o$ - ontology being changed

1: \textbf{for all} $c \in \mathcal{LC}$ \textbf{do}
2: \hspace{1em} \textsc{ProcessChange}(c, o)
3: \textbf{end for}

\textsc{ProcessChange}(c, o)

\textbf{Require:} $c$ - change to process, $o$ - ontology being changed

4: $\mathcal{T}S$ = topological sort of ontologies at the node
5: $es$ = evolution strategy for $o$
6: /*Semantics of Change*/
7: \textbf{while} generated change $gc$ by $es$ for $c$ in $o$ \textbf{do}
8: \hspace{1em} \textsc{ProcessChange}(gc, o)
9: \textbf{end while}
10: /*Change Filtering*/
11: \textbf{if} $c$ is generated in $o$ \textbf{then}
12: \hspace{1em} /*Ontology Propagation Order*/
13: \hspace{1em} \textbf{for all} ontology $d$ after $o$ in $\mathcal{T}S$ \textbf{do}
14: \hspace{2em} \textbf{if} ontology $d$ includes $o$ \textbf{then}
15: \hspace{3em} /*Change Ordering*/
16: \hspace{3em} \textsc{ProcessChange}(c, d)
17: \hspace{2em} \textbf{end if}
18: \hspace{1em} \textbf{end for}
19: \textbf{end if}
20: /*Change Implementation*/
21: change ontology $o$ according to $c$
Distributed Ontology Evolution

• Replication ontology consistency
  – An ontology is replication consistent if it is equivalent to its original and all its included ontologies (directly and indirectly) are replication consistent.
Evolution Process

• Two ways to propagate changes
  – Push-based: infeasible on the Web
  – **Pull-based**: sacrifice consistency for performance

• Distributed ontology evolution process
Identification of Change Origins

- If for some directly included replica the original has replication inconsistency, this step is aborted. Otherwise, a list of directly included replicas that have pending replication inconsistency (but whose original is replication consistent) is determined.
Distributed Ontology Evolution Algorithm

**Algorithm 2 Distributed Ontology Evolution Algorithm**

```plaintext
UPDATE DISTRIBUTED ONTOLOGY(o)

Require: o - ontology that have to be updated
1: /*Identification Of Changed Originals*/
2: inconsistentReplicas=identificationOfChangedOriginals(o)
3: for all inconsistentReplica in inconsistentReplicas do
4:   /*Extraction of Deltas*/
5:   evolutionLog=findEvolutionLog(inconsistentReplica)
6:   deltas=readEvolutionLog(evolutionLog)
7:   /*Merging Deltas*/
8:   changes=mergeDeltas(deltas)
9: end for
10: evolveOntologies(changes,o)
IDENTIFICATION OF CHANGED ORIGINALS(o)

Require: o - ontology that has to be updated
11: replicas=findFirstLevelReplicas(o)
12: for all replica in replicas do
13:   includedOntologies=findAllIncludedOntologies(replica)
14:   for all includedOntology in includedOntologies do
15:     if includedOntology is not replication consistent then
16:       generate exception("Included models are not updated yet!")
17:     end if
18:   end for
19: end for
```

2/3/2009
Implementation

• KAON
  – Open-source ontology management infrastructure
  – KAON API
Related Work – Classical Conceptual Modeling

• E-R model (conceptual) to relational model (logical)
  – Complex when evolving and implementing, have to convert from conceptual to logical first
  – **Our approach**: isolate the mapping into a separate step, making the user unaware of the logical model
Related work - Classical Conceptual Modeling

• Object-oriented modeling vs. ontology modeling
  – In OOP, responsibilities are stored in methods, which ontology doesn’t care about
  – Our approach: membership of an object in a class means some unary predicate is true for this object
    • Property change leads to re-classification
    • Multiple membership in some classes reflects different aspects are true for this object
Related Work – Frame-based logic

• F-logic
  – Frame-based knowledge modeling language
  – 1) Meta-statements about classes, 2) able to define horn-logic rules for inferring new info
  – **Our approach**: lightweight inference, which is easily implementable
Related Work - DL

• RDFS, OIL, DAML+OIL, OWL, etc
  – Modularization
  – Inconsistent management of lexical info
Future work

• Integrate and enrich our approach with similar approaches, esp. DL, w/o decreasing performance

• Ontology mapping and integration with evolution support

• Improved support for ontology evolution
  – Not only necessary changes, but also realizing secondary goals, e.g., min # of changes