©2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Signifying Ontology Complexity for Knowledge Sharing

P. Wongthongtham and B. Zadjabbari
DEBII, Curtin University, Australia
p.wongthongtham@cbs.curtin.edu.au; behrang.jabbari@gmail.com

Abstract

Ontologies are used in widespread application areas particularly to provide a shared semantically domain knowledge in a declarative formalism for intelligent reasoning. Even ontology enables knowledge sharing however complexity of knowledge being conceptualized in the ontology is critical to the success of knowledge sharing efforts. Other factor like trust in the source of knowledge can also affect knowledge transfer. In this paper we propose metrics to measure the complexity of ontology for knowledge sharing. We have chosen Software Engineering Ontology as our case study.

1. Introduction

Ontologies are used in widespread application areas e.g. semantic web, medical informatics, e-commerce, etc. Mainly ontologies are used to provide a shared semantically domain knowledge in a declarative formalism for intelligent reasoning. Even though ontology enables knowledge sharing, there are some other factors affect in effective knowledge transfer. We found two main factors related to knowledge sharing efforts i.e. trust and knowledge context. Two specific types of trust in the knowledge sharing process are benevolence-based trust and competence-based trust [1]. Besides, complexity of knowledge is critical to the success of knowledge sharing efforts. Assumingly the knowledge is conceptualized in declarative formalism i.e. Ontology having quality data, stability, and completeness. When the ontology is less complex we may not need a high value of competence-based trust. In contrast, if the ontology is rather complication, a high value of competence-based trust is required. Yet some knowledge is difficult to codify in ontology which is out of concern in this paper.

In this paper, we propose metrics to measure the complexity of ontology for knowledge sharing. We choose Software Engineering Ontology (SE Ontology) [2] as our case study. The rest of this paper is organized as follows. We describe relationship between ontology complexity and knowledge sharing in section 2. In section 3, we present related works about ontology complexity analysis. Our ontology complexity metrics are proposed in section 4. Complexity analysis of the SE Ontology is given in section 5. We conclude our work and outlook for future works in section 6.

2. Ontology complexity and Knowledge sharing

In this section we describe relationship between ontology complexity and knowledge sharing. Ontology complexity is related to the complexity of conceptualization of the domain of interest. It is measured to reflect how easy any ontology is to understand. Definition of ontology complexity is clarified in features that characterize complexity of ontology i.e. (*i*) usability and usefulness and (ii) maintainability.

For example, a more complicated ontology may indicate a more specified knowledge. However, it may be difficult to comprehend by user or software agent and would require a high value of competence-based trust. Usability and usefulness of the knowledge may be then decreasing which implies a major impact on knowledge sharing. Additionally complicated ontology may be hard to maintain.

Thus, the key factors for effective knowledge sharing are trust including benevolence based and competency based trust [1] and complexity of ontology including (*i*) usability and usefulness and (*ii*) maintainability.

3. Related works

There are existing metrics for analyzing ontology quality. Only few of them focus on complexity of ontology.

Burton-Jones et al. [3] measure elements of quality i.e. syntactic quality, semantic quality, pragmatic quality, and social quality using a number of attributes. Dazhou et al. [4] present complexity measurement for ontology based on UML. However UML cannot entirely represent semantic richness like ontology does. UML is not a suitable modeling language to represent an ontology, thus, the method cannot measure the structure complexity of ontology objectively. Chris Mungall [5] researched the increased complexity of Gene Ontology which is similar to Dalu et al. method [6]. Anthony et al. [7] also proposed a metric suite to measure the increased complexity of tourism ontologies throughout ontology evolution. However, the metrics in [5], [6], and [7] are evaluating ontology in ontology evolution. Idris [8] proposed conceptual coherence and conceptual complexity metrics based on graph theory. Orme et al. [9] examined coupling between ontologies. Nevertheless, in [5], [6], [7], [8], and [9], complexity is analyzed by the concept structure and does not consider the number of restrictions.

4. Ontology complexity

There is no unified metric so far to reflect complexity of ontotology. In this section we present our metrics: Total Number of Datatype Properties (TNoDP), Average Datatype Properties per Class (ADP/C), Total Number of Object Properties (TNoOP), Average Object Properties per Class (AOP/C), Total Number of Constraints (TNoC), Average Constraints per Object Property (AC/OP), Total Number of Hierarchical Paths (TNoHP), and Average Hierarchical Paths per Class (AHP/C). The metrics give impression of how well and how fine concepts are being defined. High value of metrics shows concepts being well presented within an ontology. We assume that the ontology being evaluated the complexity is written in Web Ontology Language (OWL).

4.1. Total number of datatype properties

Metric of Total Number of Datatype Properties (TNoDP) as shown in formula (1) presents how well concepts are being defined. TNoDP is the sum of the number of datatype properties (*dp*) in an ontology. In OWL, the datatype property is indicated as *owl:dataTypeProperty*.

$$\text{TNoDP} = \sum_{i=1}^{n} dp_i \;\ldots\ldots\ldots\ldots (1)$$

n: number of datatype properties
dp: datatype property

4.2. Average datatype properties per class

Metric of Average Datatype Properties per Class (ADP/C) as shown in formula (2) indicates an overall of how well individual concepts are being defined in the ontology. ADP/C is the total number of datatype properties divided by the sum of the number of classes.

$$\text{ADP/C} = \sum_{i=1}^{n} dp_i \,/ \sum_{j=1}^{m} c_j \;\ldots\ldots\ldots (2)$$

n: number of datatype properties
dp: datatype property
m: number of classes
c: class

4.3. Total number of object properties

Metric of Total Number of Object Properties (TNoOP) as shown in formula (3) shows how well spread of concepts within the ontology. TNoOP is the sum of the number of object properties of each class in an ontology. In OWL, the object property is indicated as *owl:objectProperty*.

$$\text{TNoOP} = \sum_{i=1}^{n} op_i \;\ldots\ldots\ldots (3)$$

n: number of object properties
op: object property

4.4. Average object properties per class

Metric of Average Object Properties per Class (AOP/C) as shown in formula (4) specifies an overall of how well spread of individual concepts within the ontology. AOP/C is the total number of object properties of each class divided by the sum of the number of classes.

$$\text{AOP/C} = \sum_{i=1}^{n} op_i \,/ \sum_{j=1}^{m} c_j \;\ldots\ldots\ldots (4)$$

n: number of object properties
op: object property
m: number of classes
c: class

4.5. Total number of constraints

Metric of Total Number of Constraints (TNoC) as shown in formula (5) illustrates how well relations being restricted in between classes. TNoC is the sum of the number of constraints in an ontology. In OWL, constraints are indicated as *owl:allValuesFrom, owl:someValueFrom, owl:hasValue, owl:cardinality, owl:minCardinality,* and *owl:maxCardinality.*

$$\text{TNoC} = \sum_{i=1}^{n} const_i \;\ldots\ldots\ldots\; (5)$$

n: number of constraints
const: constraint

4.6. Average constraints per object property

Metric of Average Constraints per Object Property (AC/OP) as shown in formula (6) demonstrates an overall of how well individual relations being restricted in between classes. AC/OP is the total number of constraints divided by the sum of the number of object properties.

$$\text{AC/OP} = \sum_{i=1}^{n} const_i \,/ \sum_{j=1}^{m} op_j \;\ldots\ldots\ldots\; (6)$$

n: number of constraints
const: constraint
m: number of object properties
op: object property

4.7. Total number of hierarchical paths

Metric of Total Number of Hierarchical Paths (TNoHP) as shown in formula (7) proves how fine concepts being presented. Hierarchical paths is also known as inheritance of concepts reflecting hierarchy of concepts (relations 'is-a', 'part-of', and 'compose-of'). TNoHP is the sum of the number of path of each concept starting from the root node to the leaf node. In OWL, the hierarchical path is represented as *owl:subClassOf.*

$$\text{TNoHP} = \sum_{i=1}^{n} p_i \;\ldots\ldots\ldots\; (7)$$

n: number of hierarchical paths
p: hierarchical path

4.8. Average hierarchical paths per class

Metric of Average Hierarchical Paths per Class (AHP/C) as shown in formula (8) presents an overall of how fine individual concept being presented. AHP/C is the total number of path of each concept divided by the sum of the number of classes.

$$\text{AHP/C} = \sum_{i=1}^{n} p_i \,/ \sum_{j=1}^{m} c_j \;\ldots\ldots\ldots\; (8)$$

n: number of hierarchical paths
p: hierarchical path
m: number of classes
c: class

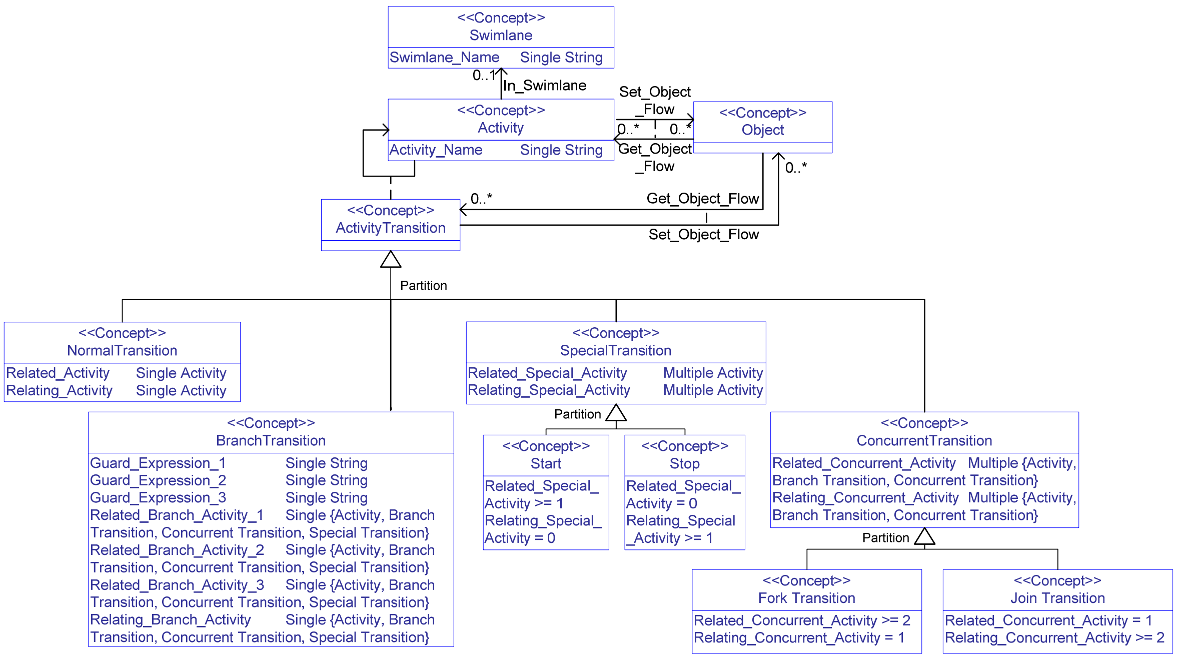
5. Software Engineering Ontology complexity

In this section, we present the complexity metrics for the existing SE Ontology as an example. Figure 1 shows an ontology model of UML activity diagrams.

In the ontology model shown in Figure 1, there are 12 classes i.e. *Swimlane*, *Activity*, *Activity Transition*, *Normal Transition*, *Branch Transition*, *Special Transition*, *Stop Transition*, *Start Transition*, *Concurrent Transition*, *Fork Transition*, and *Join Transition*. Class *Swimlane* has 1 datatype property and no object property. Class *Activity* has 1 datatype property and 2 object properties. Class *Object* has no datatype property and 2 object properties. Class *Activity Transition* has no datatype property and 1 object property. Class *Normal Transition* has no datatype property and 2 object properties. Class *Branch Transition* has 3 datatype properties and 4 object properties. Class *Special Transition* has no datatype property and 2 object properties. Class *Start* has no datatype properties and 2 inherited object properties, same as class *Stop*. Class *Concurrent Transition* has no datatype property and 2 object properties. Class *Fork Transition* has no datatype property and 2 inherited object properties, same as class *Join Transition*. Thus total number of datatype properties is 5 and average datatype properties per class is 0.42 (5/12). Total number of object properties is 22 and average object properties per class is 1.83 (22/12).

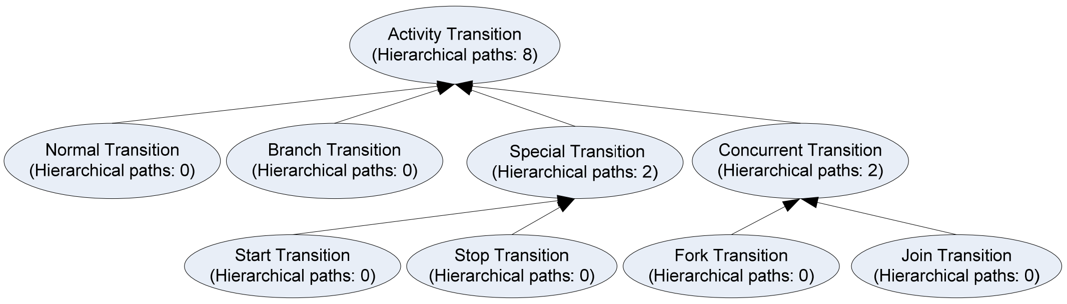
There are total 8 constraints (2 constraints in Class *Start* i.e. in object properties *Related_Special_Activity* and *Relating_ Special_Activity*, 2 constraints in Class *Stop* i.e. in object properties *Related_Special_Activity* and *Relating_ Special_Activity*, 2 constraints in Class *Fork Transition* i.e. in object properties *Related_Concurrent_Activity* and *Relating_ Concurrent_Activity*, 2 constraints in Class *Join Transition* i.e. in object properties *Related_Concurrent_Activity* and *Relating_ Concurrent _Activity*). Thus, the average of constraints per object property is 0.36 (8/22).

From the ontology model shown in Figure 1, we show total number of hierarchical paths in Figure 2. The total number of hierarchical paths is 12 (8+2+2) thus average paths per class is 1 (12/12).



<<Concept>> Swimlane
Swimlane_Name Single String
0..1
In_Swimlane
<<Concept>> Activity
Activity_Name Single String
Set_Object_Flow
0..*
0..*
Get_Object_Flow
<<Concept>> Object
0..*
<<Concept>> ActivityTransition
0..*
Get_Object_Flow
Set_Object_Flow
Partition
<<Concept>> NormalTransition
Related_Activity Single Activity
Relating_Activity Single Activity
<<Concept>> SpecialTransition
Related_Special_Activity Multiple Activity
Relating_Special_Activity Multiple Activity
<<Concept>> BranchTransition
Guard_Expression_1 Single String
Guard_Expression_2 Single String
Guard_Expression_3 Single String
Related_Branch_Activity_1 Single {Activity, Branch Transition, Concurrent Transition, Special Transition}
Related_Branch_Activity_2 Single {Activity, Branch Transition, Concurrent Transition, Special Transition}
Related_Branch_Activity_3 Single {Activity, Branch Transition, Concurrent Transition, Special Transition}
Relating_Branch_Activity Single {Activity, Branch Transition, Concurrent Transition, Special Transition}
Partition
<<Concept>> Start
Related_Special_Activity >= 1
Relating_Special_Activity = 0
<<Concept>> Stop
Related_Special_Activity = 0
Relating_Special_Activity >= 1
<<Concept>> ConcurrentTransition
Related_Concurrent_Activity Multiple {Activity, Branch Transition, Concurrent Transition}
Relating_Concurrent_Activity Multiple {Activity, Branch Transition, Concurrent Transition}
Partition
<<Concept>> Fork Transition
Related_Concurrent_Activity >= 2
Relating_Concurrent_Activity = 1
<<Concept>> Join Transition
Related_Concurrent_Activity = 1
Relating_Concurrent_Activity >= 2

Figure 1. Ontology model of UML activity diagram [10]



Activity Transition (Hierarchical paths: 8)
Normal Transition (Hierarchical paths: 0)
Branch Transition (Hierarchical paths: 0)
Special Transition (Hierarchical paths: 2)
Concurrent Transition (Hierarchical paths: 2)
Start Transition (Hierarchical paths: 0)
Stop Transition (Hierarchical paths: 0)
Fork Transition (Hierarchical paths: 0)
Join Transition (Hierarchical paths: 0)

Figure 2. Hierarchical paths in Ontology model of UML activity diagram

6. Conclusion and future works

Effectiveness of knowledge sharing is very important. The complexity of knowledge being conceptualized in an ontology affects knowledge sharing efforts. We have proposed metrics in this paper to measure complexity of the ontology. In our future works we will apply fuzzy logic based model to compute a complexity value. The complexity value can be ranged between 0 and 1 which 0 means the ontology was not very complicated while 1 means the ontology was very complicated. Fuzzy inference systems can effectively handle the situations which cannot be characterized by a simple and well defined deterministic mathematical model. We will utilize some rules and a number of membership functions to derive the result.

7. References

[1] Levin, D.Z., et al., *Trust and knowledge sharing: A critical combination*. 2007, IBM Institute for knowledge-based organizations.

[2] Wongthongtham, P., et al., *Development of a Software Engineering Ontology for Multi-site Software Development.* IEEE Transactions on Knowledge and Data Engineering, 2008.

[3] Andrew, B.-J., et al., *A semiotic metrics suite for assessing the quality of ontologies.* Data Knowl. Eng., 2005. **55**(1): p. 84-102.

[4] Dazhou, K., et al. *A complexity measure for ontology based on UML*. in *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*. 2004.

[5] Mungall, C., *Increased complexity in the GO*. 2005, BDGP / GO Consortium.

[6] Zhang, D., C. Ye, and Z. Yang. *An evaluation method for ontology complexity analysis in ontology evolution*. in *Managing Knowledge in a World of Networks, 15th International Conference, EKAW 2006*. 2006. Podebrady, Czech Republic.

[7] Anthony, M.O., Y. Haining, and H.E. Letha, *Indicating ontology data quality, stability, and completeness throughout ontology evolution: Research Articles.* J. Softw. Maint. Evol., 2007. **19**(1): p. 49-75.

[8] His, I., *Analyzing the Conceptual Coherence of Computing Applications through Ontological Excavation*. 2004.

[9] Anthony, M.O., Y. Haining, and H.E. Letha, *Coupling Metrics for Ontology-Based Systems.* IEEE Softw., 2006. **23**(2): p. 102-108.

[10] Wongthongtham, P., *A methodology for multi-site distributed software development*, in *School of Information Systems*. 2006, Curtin University of Technology: Perth.