

# A Complexity Measure for Ontology Based on UML\*

Dazhou Kang<sup>1</sup> Baowen Xu<sup>1,2,3</sup> Jianjiang Lu<sup>1,2</sup> William C.Chu<sup>4</sup>

<sup>1</sup>Department of Computer Science and Engineering, Southeast University, Nanjing, 210096, China

<sup>2</sup>Jiangsu Institute of Software Quality, Nanjing, 210096, China

<sup>3</sup>State Key Laboratory of software, Wuhan University, Wuhan 430072, China

<sup>4</sup>Department of Computer Science and Information Engineering, TungHai University, Taiwan

## Abstract

*UML is a good tool to represent ontologies. When using UML for Ontology development, one of the principal goals is to assure the quality of ontologies. UML class diagrams provide a static modeling capability that is well suited for representing ontologies, so the structural complexity of a UML class diagram is one of the most important measures to evaluate the quality of the ontologies. This paper uses weighted class dependence graphs to represent a given class diagrams, and then presents a structure complexity measure for the UML class diagrams based on entropy distance. It considers complexity of both classes and relationships between the classes, and presents rules for transforming complexity value of classes and different kinds of relations into a weighted class dependence graphs. This method can measure the structure complexity of class diagrams objectively.*

**Keywords:** Ontology; UML; class diagrams; complexity measure

## 1. Introduction

The Semantic Web has been more and more popular nowadays. The most important technologies for developing the Semantic Web are XML, RDF and Ontology. Ontology is an explicit specification of a conceptualization and is becoming increasingly important because it provides a critical semantic foundation for many rapidly expanding technologies such as software agents, e-commerce and knowledge management.

Current Ontology research and practice have their origins in declarative AI knowledge representations. Such as OIL, DAML, OWL, KIF, KL-ONE and so on. OIL, DAML and OWL are built on top of RDF Schema by adding modeling constructs from DL and most of others

are also based on DL. They are focused on formal logic-based and Web-based knowledge representation. This style of language has a well-understood semantic basis but lacks both a wide user community outside AI research laboratories and a standard graphical presentation, which is an important consideration for aiding the human comprehension of ontologies [1].

The Unified Modeling Language (UML) was originally designed for human-to-human communication of models for building systems in object-oriented programming languages. Now UML has been widely adopted by the software engineering community and its scope is broadening to include more diverse modeling tasks. One of them is Ontology modeling.

UML can be used directly as an Ontology representation either be used as a graphical front-end for another Ontology representation language. UML class diagrams provide a static modeling capability that is well suited for representing ontologies. UML object diagrams can be interpreted as declarative representations of knowledge[2].

There are a number of advantages when using UML for Ontology development. UML is a graphical notation based on many years of experience in software analysis and design by a variety of companies in a wide spectrum of industries and domains. It is widely adopted in industry and taught in many university courses and is supported by widely adopted CASE tools. Real world industrial agent-based systems need to interact with legacy enterprise systems, which often have existing UML models[3].

However, there is one significant current shortcoming of UML that it lacks a formal definition. Researchers are developing formal semantics for UML and the MOF. These efforts will remove one of the most commonly stated criticisms of the suitability of UML for representing formal models such as ontologies[1].

One of the principal goals in Ontology modeling is to assure the quality of ontology. Notice that object diagrams

\* This work was supported in part by the Young Scientist's Fund of NSFC (60373066, 60303024), National Grand Fundamental Research 973 Program of China (2002CB312000), National Research Foundation for the Doctoral Program of Higher Education of China.

Correspondence to: Baowen Xu, Dept. of Computer Science and Engineering, Southeast University, Nanjing 210096, China.. E-mail: bwxu@seu.edu.cn.

are really a kind of class diagrams; UML class diagrams constitute a key artifact in the modeling phase either as a directly representation or as a front-end. Ontology models are graphically represented by class diagrams form the basis of domain knowledge and lay the foundation for all later work, such as formalizing and reasoning. Therefore, their quality can have a significant impact on the quality of the system. Improving the quality of class diagrams will be a major step forward in improving the quality of Ontology development.

Metrics provide a valuable insight into specific ways of enhancing system quality. They are used not only for understanding, controlling, and improving development but also for determining the best ways to help practitioners and researchers[4]. The structural complexity measure is one of the most important measures to evaluate the quality of a UML class diagram. It is often the integration of many other metrics, such as metrics for single classes and metrics for relations and can measure the whole class diagram characteristics in a single value.

This paper uses weighted class dependence graphs to represent a given class diagrams, and then presents a structure complexity measure for UML class diagrams based on entropy distance. It considers complexity of both classes and relationships between the classes, and presents rules for transforming complexity value of classes and different kinds of relations into a weighted class dependence graphs.

Section 3 discusses the complexity measure for classes and relations in UML class diagrams separately. Then section 4 introduces weighted class dependence graphs, and gives the transformation rules integrating complexity of classes and relations together. The keystone is different relations have different characteristics when measuring their complexity. Section 5 presents a structure complexity measure based on entropy distance. Finally, section 6 gives the conclusion.

## 2. Related work

UML class diagram is from the field of OO software design. Within the field of software measurement, a plethora of metrics have been proposed for measuring OO software products. But most of them are related to products obtained from design and implementation phases. Only a few metrics in relation to class diagrams at the analysis phase have been proposed.

Studies on measuring the structural complexity of class diagrams are quite few until now. Chidamber and Kemerer proposed a set of six OO design metrics include the depth inheritance tree and the number of children metrics, which are well known in the field of OO design metrics but have not been widely accepted. These metrics can be used in the design phase, and are defined at class level[5].

Lorenz and Kidd proposed a group of class size metrics,

class inheritance metrics and class internals called “design metrics”, which deal with the static characteristics of software design[6].

Brito, Abreu and Melo proposed the MOOD (Metrics for Object Oriented Design) set of metrics, which allow measurement of encapsulation, inheritance, polymorphism, and message passing. MOOD metrics were used in the design phase, and are defined at system level[7].

Marchesi proposed a set of metrics to measure UML class diagrams at the analysis phase, but did not take into account some UML measurable elements, such as associations, aggregations and dependencies[4].

Genero proposed new metrics to cover the necessity of measuring these relationships[8], [9].

Manso and Genero used 8 metrics for measuring the structural complexity of class diagrams due to the usage of UML relationships, and 3 metrics to measure their size, and then carried out three controlled experiments to ascertain if any correlation exists between the structural complexity and the size of UML class diagrams and their maintainability[10]. But they did not give a single complexity measuring integrate all these metrics.

## 3. Complexity measure for classes and relations

Classes and relations are the basic elements of class diagrams. It can be considered that other classifies equivalent to classes here, including templates, interfaces and so on. In UML, relations are not independent elements. Each ends of a relation must link to a certain class, and the semantic of the relation is also related to these classes as well as the complexity. The complexity of classes should be measured first.

### 3.1. Classes

A common class has attributes and operations, and takes part in the class hierarchy. The class structure and the class inheritance are the two main factors influencing the complexity of a class.

At first we deal with quantifying an individual class, i.e. the attributes and operations of the class. There are many simple class size metrics: number of public instance methods, number of instance methods and variables, number of class methods and variables, and so on[6].

Another important factor influencing the complexity of a class is the inheritance. A small sized class may be very complex because it inherits many attributes and operations from its ancestors. Class inheritance metrics may involve: number of methods overridden, inherited and added [6]; number of ancestor classes can potentially affect this class, number of immediate subclasses subordinated to a class in the class hierarchy [5]; method and attribute inheritance factor, polymorphism factor [7], [11]. Marchesi proposed a set of metrics for single classes including measuring the

weighted number of responsibilities of a class, inherited or not [4].

This paper is not going to present a new method for measuring the complexity of a class. We use suitable metrics and methods proposed by others. Which specific metrics to use is not important, we just require it satisfies some restriction: there should be just one metric for measuring each class and it has only a single value; the value is above zero and denote the complexity of the class; it should take both the class structure and the class inheritance into account.

For example, one possible simple complexity measure can be:  $1 * (\text{Number of public operations} + \text{Number of public attributes}) + 0.5 * (\text{Number of public operation and attributes inherited}) + 0.2 * (\text{Number of non-public attributes and operations}) + (\text{Number of ancestors}) + (\text{Number of direct subclasses}) + (\text{other factors})$ . The number of ancestors and subclasses factor is optional, because it can be considered when measuring the generalization relations.

The other measure of classes is the cohesion measure [12], [13], [14]. It sometimes can replace the complexity measure, or be combined with the complexity measure.

### 3.2. Relations

There are mainly three kinds of relations in UML class diagrams: associations, generalizations, and dependencies. Marchesi only considered the generalizations [4]. Genero proposed a set of metrics for these relations and carried out several experiments to ascertain if any correlation exists between the metrics [15]. They included number of associations, aggregations, dependencies, generalizations and so on. But they did not give a single complexity measure integrate all these metrics. They mainly used metrics like number of associations in a class diagram; it is not enough.

We point out two new problem when measuring relations: how to compare the complexity between different kinds of relations and how to compare the complexity between relations of the same kinds.

Complexity of different kinds of relations can be compared. The dependency between classes causes the complexity on understanding and maintaining the relations between the classes. For example, when a composition association relates two classes, there is stronger dependency between the two classes than an aggregation association relates them, and the complexity of composition is higher than aggregation. Different kinds of relations influence the dependency between classes in different degrees.

We fractionize relations in UML class graph into totally 10 kinds and their different influences on the dependency between classes can be weighted in a value, and it forms Table 1.

**Table 1. Dependency weight value of relations**

No.	Relation	Weight
1	Dependency	H1
2	Common association	H2
3	Qualified association	H3
4	Association class	H4
5	Aggregation association	H5
6	Composition association	H6
7	Generalization (parent class is concrete)	H7
8	Binding	H8
9	Generalization (parent class is abstract)	H9
10	Realize	H10

Dependency relations are the most common relations. A dependency relation just shows there is dependency between two class without any explanations and restrictions, so H1 should be the minimum. Common association relations denote the relationship between instances of classes; they cannot be weaker than dependency relations. Qualified associations and association classes add restrictions to associations, they may more complex and the dependency between classes with these relations may be stronger than between classes with common association. Aggregations are specific associations, and compositions are specific aggregations. For example, A is a composite of B; when A is destroyed, B should be destroyed or given to another object. Aggregations do not have this restriction, but are more restrict than dependency relations, as one object cannot aggregate itself directly or indirectly. In generalizations, subclasses inherited all the non-private characteristics of the parent classes, and composition classes can only access the public elements of the nested classes. When parent classes are concrete, subclass can add new element and override inherited operations. When parent classes are abstract, subclasses should implement the virtual operations of the parent classes or they cannot have any instances. Binding relations are between templates and classes. The class has all the attributes and operations of the template, including private elements. When realizing a class (usually interface), implement class must realize all the operations of the interface. So realize relations have the highest weight value. It has:

$$H1 \leq H2 \leq H3 \leq H4 \leq H6 \quad (1)$$

$$H1 < H5 < H6 < H7 < H8 < H9 < H10 \quad (2)$$

Each ends of a relation should link to a certain class. So relations of the same kind may have different complexity because they related to different classes. The complexity of the classes may influence the complexity of the relations. These influences can be more or less, and different kinds of relations must be dealt with separately. It will be discussed in the next section.

## 4. Weighted Class Dependence Graphs

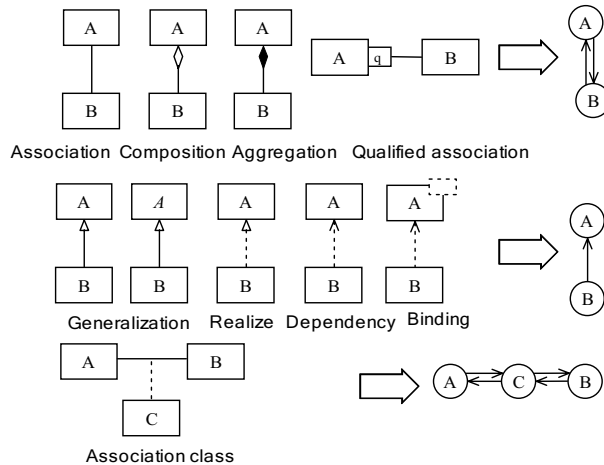
### 4.1. The definition of WCDG

Class diagrams have many specific symbols expressing different meanings. It will be exhausting to measure structural complexity on the class diagram directly. So we use Weighted Class Dependence Graphs (WCDG) to denote the given class diagram abstractly.

**Definition 1.**  $D$  denotes a given class diagram,  $T = \{t_i \mid 1 \leq i \leq 10\}$ ,  $V(D) = \{c \mid c \text{ is a class in } D\}$ ,  $R(D) = \{(n_1, n_2, w) \mid n_1, n_2 \in V(D) \wedge (\text{there are relations in } T \text{ from } n_1 \text{ to } n_2 \vee n_1 = n_2)\}$ , where  $(n_1 \neq n_2)$ ,  $w = \sum W_i$ ,  $W_i$  is the dependence weighted value of each relation from  $n_1$  to  $n_2$ ;  $(n_1 = n_2)$ ,  $w$  should be added the dependence weighted value of class  $n_1$  i.e.  $w = \sum W_i + W(n_1)$ . The dependence weighted values will be discussed later.

WCDG is defined as  $G(D) = (N, E)$ , where  $N = V(D)$ ,  $E = R(D)$ , i.e. the nodes and edges of WCDG.

Every node in the WCDG is corresponding to a class in the class diagram; relations are transformed to edges between the nodes. If there is a one-way relation between classes A and B in the class diagram, there should be an edge between nodes corresponding to A and B, which has the same direction with the relation. While the relation is bidirectional, there should be two edges with different directions to denote that relation. Their weight value should associate with the complexity measure of the corresponding relations.



**Figure 1. WCDG transform rules**

The transformation rules of different kind of relations have some different. Rectangles represent classes in the class diagrams; circles represent classes in the WCDG. The transform rules are showed in Figure 1.

Firstly we form the structure of the WCDG from the class diagram by transformation rules, and then calculate the weight value of the edges and nodes by the complexity

measure of classes and the dependence weight value of relations. We will discuss the dependence weight value of different kind of relations separately.

### 4.2. The dependence weighted values of relations

**Dependency.** Let the complexity measure of class A and B are  $C(A)$  and  $C(B)$ ; the weight value of dependencies is  $H1$ . When A becomes more complex, this dependency relation will be more difficult to deal with; but the complexity of B does not influence it. The complexity measure of this relation should be direct proportion of  $C(A)$ . A possible measure can be  $W = H1 * C(A)$ .

**Association.** Association means the relations of the instances of the classes. Associations between complex classes may not be more complex than associations between simple classes. Usually, associations are implemented by pointers. The complexity measure will be higher if the destination multiplicity becomes higher. It can be  $W = H2 * (2 - 1/n) * C(A)$ . Here  $n$  is the maximum destination multiplicity of A. When  $n$  is  $\infty$  (or any number above 0), let  $1/n = 0$ ,  $W = H2 * 2 * C(A)$ .

When the association is bidirectional, it will be transformed to two edges with different direction.  $W1 = H2 * (2 - 1/n) * C(A)$ ;  $W2 = H2 * (2 - 1/m) * C(B)$ , where  $m$  is the maximum destination multiplicity of B.

**Qualified association.** It is like the common ones:  $W = H3 * (2 - 1/n) * C(B) + b$ . Here  $b$  denotes the complexity of the qualifier; it is usually very low as qualifier usually only contains 1 or 2 simple expressions. It seems that qualifiers can reduce the complexity of associations as it usually makes  $n = 1$ .

If the relation is from B to A, it can be considered as the common association, i.e.  $W = H3 * (2 - 1/n) * C(A)$ .

**Association class.** Association classes can be transformed to associations between three classes, showed in Figure 1. The four edges can be seemed as common associations. The edges starting from C always has the destination multiplicity  $n = 1$  because an association class can only relate to one certain pair of instances of A and B.  $W1 = H4 * (2 - 1/n) * C(C)$ ;  $W2 = H4 * C(B)$ . When the relation is not bidirectional, such as only from A to B, the edge from B to C should be canceled.

**Aggregation and composition association.** According to the definition of composition, A has full control of B.  $W1 = H6 * (2 - 1/n) * C(B)$ . B can only take part in one composition, so it has  $W2 = H6 * C(A)$ .

Aggregations have the same WCDG form as compositions. But they are weaker than compositions, and one object can be part of many aggregates.  $W1 = H5 * (2 - 1/n) * C(B)$ ;  $W2 = H5 * (2 - 1/m) * C(A)$ .

Aggregations and compositions are transferable relations. That is if B is an aggregate or composite, it will increase the complexity of the relation. Suppose the aggregations and compositions take part in the complexity measure of

classes; perhaps W1 will be more accurate, while W2 may be confused as the calculation of  $C(A)$  already have included this relation as a parameter. Aggregations and compositions are not in a well-formed hierarchy like generalization hierarchy; sometimes they can form a recurrence in the class diagram (not in the object diagram). They are less strict than generalizations; their influence on class complexity is not exact.

**Generalization.** Generalizations are also transferable relations. generalization hierarchy is a well-formed structure, and inherited mechanism is strict and exact. So generalizations take part in the complexity measure of classes. Generalizations can not be bidirectional. The inherited complexity has been calculated in the complexity measure of classes. It is a redundancy to calculate it here. So simply  $W = H7 * C(A)$  when A is a concrete class; or  $W = H9 * C(A)$  when A is an abstract class.

**Binding.** A is a template, and will not exist in runtime. It is one kind of dependency.  $W = H8 * C(A)$ .

**Realize.** It is one kind of abstract dependency, and one kind of generalization without inheriting the implements. A is usually an interface.  $W = H10 * C(A)$ .

### 4.3. The edges and nodes in WCDG

After all the relations have been transformed to edges, some of the edges can be merged. For every pair of nodes  $(n_1, n_2)$ , not including  $n_1 = n_2$ , there should be only one edge exist, its weighted value  $W(n_1, n_2)$  is the sum of the weighted values of all edges from  $n_1$  to  $n_2$ . Notice that  $W(n_1, n_2)$  and  $W(n_2, n_1)$  do not influence each other.

$W(n, n)$  denotes the dependence weighted value of one single class. It includes the complexity of the class, but that is not all. For example, associations may from one class to the same class, denote the relations between objects of this class. They can be transformed to edges using the rules above.  $W(n, n)$  should be the sum of weighted values of all edges from  $n$  back to  $n$ , and be added the  $H1 * C(n)$ .  $C(n)$  is the complexity of the class  $n$  denotes; a single class is at least depend on itself, even if it

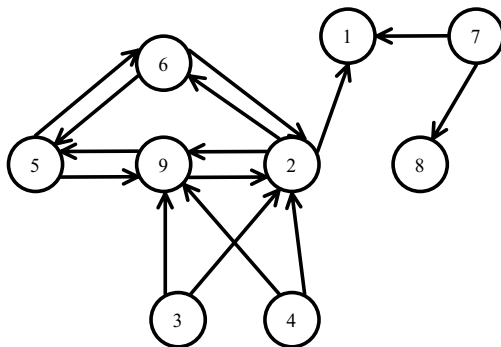


Figure 3. The sample WCDG

has no relations. It is obvious that  $W(n_1, n_2) = w$ , where  $(n_1, n_2, w) \in R(D)$ .

The WDCG can also be expressed in a matrix, in which  $W[i][j] = W(n_i, n_j)$ . It will predigest the calculation of complexity.

### 4.4. An example

Here is a simple class diagram in Figure 2 as an example.

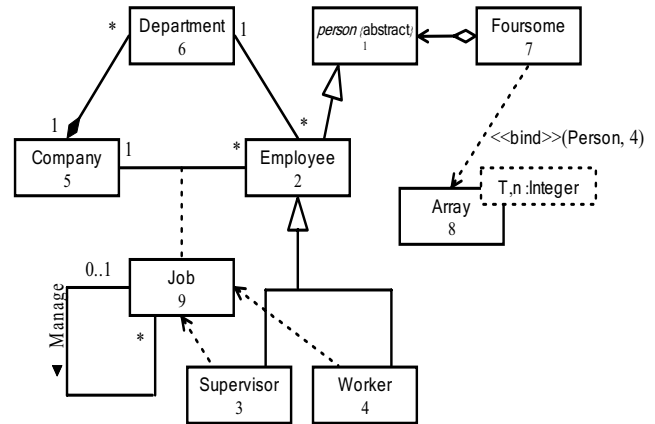


Figure 2. The sample class diagram

First give each classifier a number, and calculate their complexity values. Their complexity values are presented by  $C[1], C[2], \dots, C[9]$ . Here we are not going to discuss the complexity measure of the classes, just simply suppose that:

$$C[] = \{3.5, 5.8, 4.2, 2.6, 7.2, 2.4, 5.0, 5.0, 1.2\};$$

Give each relation a weight value satisfy Equation (1), (2):

$$H1 = 1, H2 = 2, \dots, H10 = 10.$$

Use transformation rules above to build the WCDG of this class diagram (Figure 3), calculate each  $W(n_1, n_2)$  and write it in  $W[n_1][n_2]$ ,  $(n_1, n_2 = 1, 2, 3, \dots, 9)$ , in Table 2.

For instance,  $W(9, 9) = H1 * C[9] + H2 * 2 * C[9] = 1 * 1.2 + 2 * 2 * 1.2 = 6.0$ .

Table 2: Dependence matrix of the sample

	1	2	3	4	5	6	7	8	9
1	3.5	0	0	0	0	0	0	0	0
2	31.5	5.8	0	0	0	2.4	0	0	4.8
3	0	40.6	4.2	0	0	0	0	0	1.2
4	0	40.6	0	2.6	0	0	0	0	1.2
5	0	0	0	0	7.2	28.8	0	0	9.6
6	0	23.2	0	0	43.2	2.4	0	0	0
7	30.6	0	0	0	0	0	5.0	0	0
8	0	0	0	0	0	0	0	5.0	0
9	0	23.2	0	0	28.8	0	0	0	6.0

## 5. A structure complexity measure based on entropy distance

X and Y are two discrete stochastic variables:

$$A_x = \{x_i \mid 1 \leq i \leq m\},$$

$$A_y = \{y_j \mid 1 \leq j \leq n\}.$$

The entropy of their joint distribution is:

$$H(X, Y) = - \sum_{x_i \in A_x \wedge y_j \in A_y} p(x_i, y_j) \log p(x_i, y_j) \quad (3)$$

The entropy of X when Y happened:

$$H(X|Y) = \sum_{x_i \in A_x \wedge y_j \in A_y} P(x_i, y_j) \log \frac{1}{P(x_i|y_j)} \quad (4)$$

The mutual-information of X and Y:

$$I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \quad (5)$$

The entropy distance is defined as:

$$DH(X, Y) = H(X, Y) - I(X, Y) \quad (6)$$

Let D be a given class diagram; G(D) is the WCDG corresponding to D; N(D) is the set of all nodes in the WCDG; W(n<sub>i</sub>, n<sub>j</sub>) denotes the total weight of edges from n<sub>i</sub> to n<sub>j</sub> (n<sub>i</sub>, n<sub>j</sub> both in N(D)).

We can use the entropy distance to measure the complexity of G(D). Use stochastic variables X and Y to denote the output and input edges weight of each node. Let  $A_x = A_y = N(D)$ , for each  $x_i \in A_x$ , each  $y_j \in A_y$ , we have

$$p(x_i) = \frac{\sum_{n2 \in N(D)} W(x_i, n2)}{\sum_{n1 \in N(D)} \sum_{n2 \in N(D)} W(n1, n2)} \quad (7)$$

$$p(y_j) = \frac{\sum_{n1 \in N(D)} W(n1, y_j)}{\sum_{n1 \in N(D)} \sum_{n2 \in N(D)} W(n1, n2)} \quad (8)$$

$$p_{X,Y}(x_i, y_j) = \frac{W(x_i, y_j)}{\sum_{n1 \in N(D)} \sum_{n2 \in N(D)} W(n1, n2)} \quad (9)$$

$$p_{X,Y}(x_i|y_j) = \frac{W(x_i, y_j)}{\sum_{n1 \in N(D)} W(n1, y_j)} \quad (10)$$

**Definition 2.** The complexity of D is defined to be Complexity(D) = DH(X, Y) = H(X, Y) - I(X, Y).

Specially, when D =  $\emptyset$ , Complexity(D) = 0.

**Table 3: The joint distribution of X and Y,  $p(x_i, y_j)/\%$**

	1	2	3	4	5	6	7	8	9	$p(x_i)/\%$
1	1.00	0	0	0	0	0	0	0	0	1.00
2	8.96	1.65	0	0	0	0.68	0	0	1.37	12.66
3	0	11.55	1.20	0	0	0	0	0	0.34	13.09
4	0	11.55	0	0.74	0	0	0	0	0.34	12.64
5	0	0	0	0	2.05	8.20	0	0	2.73	12.98
6	0	6.60	0	0	12.29	0.68	0	0	0	19.58
7	8.71	0	0	0	0	0	1.42	0	0	10.13
8	0	0	0	0	0	0	0	1.42	0	1.42
9	0	6.60	0	0	8.20	0	0	0	1.71	16.51
$p(y_j)/\%$	18.67	37.96	1.20	0.74	22.54	9.56	1.42	1.42	6.49	

**Table 4:  $p(x_i|y_j)$**

	1	2	3	4	5	6	7	8	9
1	0.05	0	0	0	0	0	0	0	0
2	0.48	0.04	0	0	0	0.07	0	0	0.21
3	0	0.30	1	0	0	0	0	0	0.05
4	0	0.30	0	1	0	0	0	0	0.05
5	0	0	0	0	0.09	0.86	0	0	0.42
6	0	0.17	0	0	0.55	0.07	0	0	0
7	0.47	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	1	0
9	0	0.17	0	0	0.36	0	0	0	0.26

The complexity of the sample class diagram in Figure 2 is calculated as an example. First we calculate the joint

distribution of X and Y using Equation (7), (8), (9) based on data from Table 2 and generate Table 3. Then we can calculate the entropy of their joint distribution:

$$H(X, Y) = - \sum_{x_i \in A_x \wedge y_j \in A_y} p(x_i, y_j) \log p(x_i, y_j) = 2.691$$

Calculate  $p(x_i|y_j)$  using Equation (10) and generate Table 4. We can get

$$H(X|Y) = \sum_{x_i \in A_x \wedge y_j \in A_y} P(x_i, y_j) \log \frac{1}{P(x_i|y_j)} = 1.062$$

The mutual-information of X and Y is

$$I(X, Y) = H(X) - H(X|Y) = 0.947$$

The complexity therefore the entropy distance is

$$\text{Complexity}(D) = DH(X, Y) = H(X, Y) - I(X, Y) = 1.744$$

## 6. Conclusions

Comparing to the other ways of Ontology representation, using UML for Ontology development has its own advantages. One of the principal goals in ontology modeling is to assure the quality of ontology. UML class diagrams constitute a key artifact in the modeling phase. The structural complexity is one of the most important measures to evaluate the quality of a UML class diagram.

We use weighted class dependence graphs to represent a given class diagrams, and then present a structure complexity measure for UML class diagrams based on entropy distance. It considers complexity of both classes and relationships between the classes, and presents rules for transforming complexity value of classes and different kinds of relations into a weighted class dependence graphs. This method of measure has many good properties; therefore it can measure the structure complexity of class diagrams objectively.

The UML class diagrams can only represent static model, therefore ontologies of static knowledge. When dealing with dynamic knowledge, UML dynamic diagrams and state diagrams should be used. How to represent ontologies using dynamic and state diagrams, and how to measure their quality is problems in future work.

## References

- [1] S. Cranefield, "UML and the Semantic Web", *Proceedings of the International Semantic Web Working Symposium*, 2001, 113-130.
- [2] K. Baclawski, M.K. Kokar, P.A. Kogut, L. Hart, J. Smith, W.S. Holmes, J. Letkowski, M.L. Aronson, "Extending UML to Support Ontology Engineering for the Semantic Web", *Lecture Notes in Computer Science*, vol. 2185, 2001, 342-360.
- [3] P. Kogut, S. Cranefield, L. Hart, M. Dutra, K. Baclawski, M. Kokar, J. Smith, "UML for Ontology Development", *The Knowledge Engineering Review*, Cambridge University Press, December 2001.
- [4] M. Marchesi, "OOA metrics for the United Modeling Languages", *Proceedings of 2nd Euromicro Conference on Software Maintenance and Reengineering*, Palazzo degli Affari, Italy, 1998, 67-73.
- [5] S. Chidamber, C. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, 1994, 20(6), 476-493.
- [6] M. Lorenz, J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.
- [7] E. Brito, F. Abreu, W. Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality", *Proceedings of 3rd International Metric Symposium*, 1996, 90-99.
- [8] M. Genero, M.E. Manso, M. Piattini, et al, "Early metrics for object oriented information systems", *Proceedings of 6th International Conference on Object Oriented Information Systems*, London, UK, 2000, 414-425.
- [9] M. Genero, M. Piattini, "Empirical validation of measures for class diagram structural complexity through controlled experiments", *Proceedings of 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, Budapest, Hungary, 2001, 87-95.
- [10] M.E. Manso, M. Genero, M. Piattini, "No-Redundant Metrics for UML Class Diagram Structural Complexity", *CAiSE 2003 The 15th Conference On Advanced Information Systems Engineering*, LNCS 2681, 127-142.
- [11] E. Brito, F. Abreu, H. Zuse, H. Sahraoui, W. Melo, "Quantitative Approaches in Object-Oriented Software Engineering", *Object-Oriented technology: ECOOP'99 Workshop Reader, Lecture Notes in Computer Science 1743*, Springer-Verlag, 1999, 326-337.
- [12] M. Hitz, B. Montazeri, "Measuring coupling and cohesion in object-oriented systems", *Proceedings of International Symposium on Applied Corporate Computing*, Monterrey, Mexico, 1995, 25-27.
- [13] L.C. Briand, S. Morasca, V.R. Basili, "Defining and validating measures for object-based high-level design", *IEEE Transactions on Software Engineering*, 1999, 25(5): 722-743.
- [14] H.S. Chae, Y.R. Kwon, D.H. Bae, "A cohesion measure for object-oriented classes", *Software Practice & Experience*, 2000, 30(12), 1405-1431.
- [15] M. Genero, M.E. Manso, M. Piattini, F. Garcia, "Early metrics for object oriented information systems", *Proceedings of 6th International Conference on Object Oriented Information Systems*, Spring Press, London, UK 18-20, December 2000, 414-425.
- [16] M. Genero, M.E. Manso, M. Piattini, F. Garcia, "Assessing the Quality and the Complexity of OMT Models", *2nd European Software Measurement Conference-FESMA 99*, Amsterdam, the Netherlands, 1999, 99-109.
- [17] J. Rumbaugh, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading, MA, USA, 1999.