Building Domain Specific Dictionaries of Verb-Object Relation from Source Code

Yasuhiro Hayase
Faculty of Information Sciences and Arts
Toyo University
Kawagoe, Saitama, Japan
Email: hayase@toyo.jp

Yu Kashima Yuki Manabe Katsuro Inoue
Graduate School of Information Science and Technology
Osaka University
Suita, Osaka, Japan
Email: {y-kasima,y-manabe,inoue}@ist.osaka-u.ac.jp

***Abstract*—An identifier is an important key in mapping program elements onto domain knowledge for the purpose of program comprehension. Therefore, if identifiers in a program have inappropriate names, developers can waste a lot of time trying to understand the program. This paper proposes a method for extracting and gathering verb-object (V-O) relations, as good examples of naming, from source code written in an object-oriented programming language. For each of several application domains, dictionaries containing the V-O relations are built and evaluated by software developers. The evaluation results confirm that the relations in the dictionaries are adequate in many cases.**

I. Introduction

Program comprehension consumes at least half the time allocated to the software maintenance process [1], [2]. Identifiers in the source code are very important for program comprehension. Software developers generally try to understand a program by guessing the roles of the program elements from their identifiers [3], [4].

In general, an identifier consists of several words, and multiple identifiers are used to represent the behavior of a program. As a result, various relations exist between the words in identifiers. For example, the name of a method has a verb-clause representing its behavior together with optional object-clauses in most cases. The name of the method relates to the name of the class and formal parameters when declaring the method, or receiver object and actual parameters when calling the method. These related identifiers sometimes assume the role of the object-clause for the verb in the method name.

Accordingly, if identifiers in a program have inappropriate names, it is difficult for developers to guess the role of the program elements and map the elements onto knowledge of the application domain. Lawrie *et al.* [5] revealed that identifiers that are acronyms or meaningless serial numbers cause developers to waste much more time in program comprehension, compared to when identifiers are spelled out fully without any abbreviations. Therefore, software developers should give identifiers names that accurately represent the role of the program elements when creating or changing source code.

Unfortunately, not all developers are able to give appropriate names to identifiers, since a broader knowledge and a great deal of experience are necessary to define accurate names. Developers need to learn the rules of various words and their combinations (e.g., naming rules) in different domains, such as the programming language, development organization or application domain. The only way to learn these rules is through examples, since the rules are not documented in many cases.

For the purpose of naming identifiers, our research group is working on building dictionaries containing good examples of identifier names. The dictionary has a network structure. In a previous work, we proposed a method for building a dictionary of the super-sub (abstract-concrete) relation of nouns used in identifiers [6]. We believe that the dictionary is useful for naming classes and variables.

This paper proposes a method for building a dictionary of verb-object (**V-O**) relations extracted from source code. In particular, verbs are extracted from a method name using existing natural language processing and the pattern matching system we developed, and then objects are extracted from the method name, names of formal parameters of the method, and the name of the class to which the method belongs. This process is applied to a set of source files categorized by application domain. Relations that appear frequently in a domain are included in the dictionary for that domain.

The proposed method was applied to Java source files belonging to four different domains, and the generated dictionaries evaluated by developers. The results of the evaluation confirm that most of the entries in the dictionaries are domain specific relationships and general relations in Java source code.

The rest of the paper is structured as follows. Section II-B explains in detail the V-O relationship in object-oriented programs and naming rules of methods, while Section III presents the algorithm used to build a dictionary of V-O relations from source code. Section IV discusses the evaluation experiment and its results. Finally, Section V discusses related works, while Section VI gives our conclusions and future works.

II. Verb-Object Relations in Object-Oriented Programs

This section explains in detail the V-O relationship in methods of object-oriented programs. First, the naming rule for identifiers in object-oriented languages, especially Java, is given. Then the V-O relationship between words in identifiers, which are related to method declarations, is described.

A. Naming rule for identifiers

In general, it is recommended that an identifier has a specific and obvious name that expresses its role. Since the role of identifiers is sometimes complicated in object-oriented programs, an identifier in an OO program is often expressed as a compound name, consisting of several words.

Unfortunately, white spaces are prohibited in identifiers in many programming languages, and therefore, alternative ways are employed to express compound names, i.e., camel case and snake case. Camel case refers to a concatenated string of words, whose first letters are capitalized (e.g., CamelCase). Snake case is a concatenated string of words with underscores between words (e.g., snake_case). In Java source code, camel case is recommended and is most popular [7].

Commonly, the name of a method in an object-oriented program includes a verb. In most cases, the head of the method name is a verb or verb clause, followed by a noun or adjectives. In other cases, the head of the method name is a noun or adjective, or a noun or adjective clause, followed by a verb in the past tense. For example, `java.awt.event.ActionLister` in the Java API [8] has a method `actionPerformed(ActionEvent)`.

In a few cases, the method name contains no verbs. For example, the names of the `toString()` method of `java.lang.Object` and `newInstance()` method of `java.lang.Class` do not contain any verbs. From one point of view, these methods omit the obvious verbs, such as *convert* or *create*. The alternative view is that `to` and `new` act as verbs in the methods. This paper adopts the latter view.

B. Verb and Object Words in a method

Object-oriented programs contain many statements describing operations on targets. An operation and a target correspond to a verb and an object, respectively. Receiver or parameter objects are used as the targets of the operation. Fry *et al.*[9] proposed a method to extract verbs and direct objects from method names and parameters or class names, respectively.

In source code, there are many pairs of verbs and objects that seldom appear in natural language texts. For example, `java.net.Socket` has a method `bind(SocketAddress)`. The method name actually means "bind SocketAddress to Socket". However, in natural language text, except for programming documents, socket almost never acts as the object of the verb "bind".

Moreover, the verb-object relations that appear in source code are sometimes specific to a certain program domain, since programs in different domains use different words, or the same words with a different meaning. For instance, in a database domain, noun *cursor* is used as a pointer to selected data for the purpose of accessing the data one-by-one. On the other hand, in the GUI application domain, cursor means the editing point of a text area. Only in a database domain does *fetch from cursor* mean to acquire one set of data from the database.

III. Approach to Obtaining V-O Relations from Source Code

This section describes a method to build a dictionary of V-O relations by extracting the relations from object-oriented programs. The input to the method is source files for a certain domain written in an object-oriented language, while the output is a dictionary composed of tuples consisting of a verb (**V**), direct-object (**DO**), and indirect-object (**IO**) specific to the domain. The IO field may be empty. An outline of the approach is shown in Figure 1.

The approach consists of the following three steps.

- **Step 1: Obtaining the identifiers related to each method**
  Retrieve all method declarations in the input source, and then for each declaration, obtain identifiers that relate to a declaration.
- **Step 2: Extracting V-O relations**
  Analyze the output of step 1 to obtain the words for each identifier and the word class for each word. This information is expressed as a tuple consisting of V, DO, and IO.
- **Step 3: Filtering V-O relations**
  From the output of step 2, select the tuples that appear in more than a certain number of software products.

The following section describes each step.

A. Step 1: Obtaining the identifiers related to each method

This step extracts all the method declarations and related identifiers from the input source files. In particular, source files are parsed to extract all method declarations. Then, for each method declaration, the return type, name of the method, names and types of formal parameters, and the name of the class to which the method belongs, are extracted.

B. Step 2: Extracting V-O relations

This step analyzes the output of step 1 to obtain the words comprising the identifiers and a part of speech for each word. This information is expressed as a tuple consisting of V, DO, and IO.

Extracting the V-O relations is done in the following stages.

- **Obtaining the method property**
  Acquire the words relating to the method, together with their parts of speech, (the **method property**) from the output of the previous step.
- **Extract V-O relations by pattern matching**
  Acquire the tuples consisting of <V, DO, IO> by applying a pattern matching technique to the method property.

Details of the method property and how we obtain the information is described below. Thereafter, we explain the patterns and pattern matching algorithm.

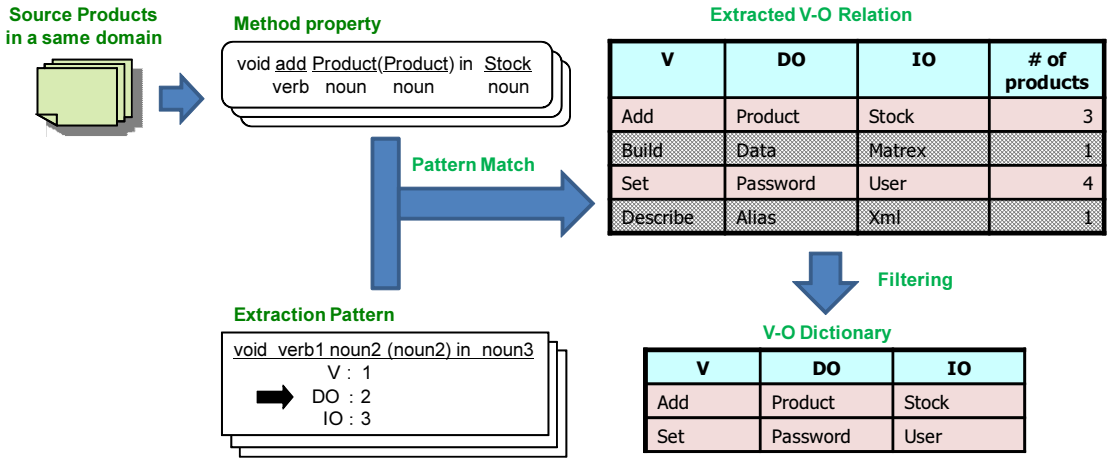

Figure 1. Overview of our approach

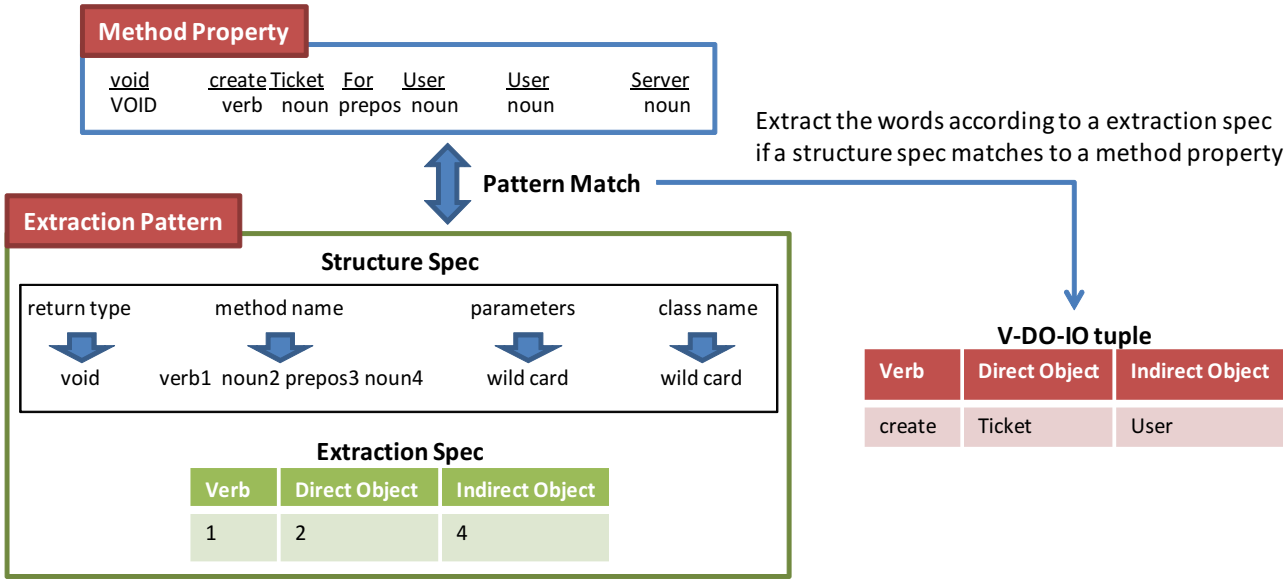

Figure 2. Method property, extraction pattern and pattern matching

1) Obtaining the method property: Method property is expressed as a tuple of four sequences of words together with their respective parts of speech. (Figure 2) The four sequences respectively correspond to the return type of the method, the name of the method, the parameters of the method, and the name of the class to which the method belongs. If the method has one or more parameters, two tuples are produced for the method property; one contains the sequence of types of the parameters in the third element, while the other contains the sequence of names of the parameters. The procedure to acquire the method property from the identifiers obtained in the previous step is described below.

The first element of the method property is a sequence that only contains the return type of the method. If the method has a non-void return type, the name of the return type is treated as a noun. Otherwise, the return type `void` is treated as a special part of speech *"VOID"*.

The fourth element is almost the same as the first element; it is a sequence containing only the class name, which is treated as a noun.

The third element is a sequence with length greater than 0. The n-th formal parameter of the method corresponds to the n-th element of the sequence. As described above, formal parameter names and their types are respectively used for two method properties. Since the procedure to build the sequence is the same, parameter names and types are not distinguished and are simply referred to as the "name" in this paragraph. Each of the names is treated as a single noun.

The third element is a sequence derived from the method name using OpenNLP [1]. First, a sequence of simple words from the compound name of the method is obtained. Then the parts-of-speech of each simple word

[1]OpenNLP is a natural language processing system. http://opennlp.sourceforge.net/

are determined by OpenNLP. As described in Section II-A, headings *to* and *new* are treated as verbs.

2) Extracting V-O relation by pattern matching: Pattern matching is used to obtain a tuple <V, DO, IO> from the method property (see Figure 2). The pattern (**extraction pattern**) is composed of two parts, the **structure spec** and **extraction spec**.

The structure spec is represented by a tuple of four elements. These four elements correspond, respectively, to the return type, name of the method, parameters, and class name. Unlike the method property, the elements of the tuple are not sequences of words. An element of a structure spec is a wild card or a sequence of pairs consisting of a part-of-speech and a word number.

A structure spec matches the method property only when all of the following conditions are satisfied.

1) For each n-th element of the structure spec and the method property, the element of the structure spec is a wild card, or both sequences have strictly the same part-of-speech in the same order.
2) If the structure spec has the same word number at several points, the corresponding words in the method property are the same.

The extraction spec is represented by a tuple of three word numbers. The third number may be empty. If the structure spec matches a method property, the words that correspond to the word numbers specified in the extraction spec are extracted as V, DO, and IO words, respectively. If the third number is empty, IO is also empty.

Several patterns are defined by hand prior to pattern matching. The method properties obtained in the previous stage are collated with the patterns. If two or more patterns match a single method property, multiple tuples for each pattern are extracted from the single method property.

C. Step 3: Filtering V-O relation

This step filters the output of step 2, and builds a dictionary of V-O relations that frequently appear in source code. In particular, only those tuples that appear in a certain number of software products are included in the dictionary. The threshold is tuned by hand.

IV. Evaluation Experiment

We performed an experiment to evaluate the validity of the dictionary built using the proposed method. This section describes the experiment in detail, together with its result, and then examines and discusses the result.

A. Experimental setup

We prepared 31 extraction patterns by hand (Table I). The target of extraction was source code for 37 open source software products (see Table II). The products were classified into four domains, Web Applications, XML Processing, Databases, and GUIs, which are abbreviated as Web, XML, DB, and GUI, respectively. For each domain, source files were analyzed and a dictionary built.

Table III
NUMBER OF METHODS AND EXTRACTED TUPLES

	# of methods	# of methods that match any patterns	matched ratio	# of tuples
Web	74707	67276	90%	67429
XML	55812	46885	84%	49926
DB	74127	60326	81%	63087
GUI	298696	247918	83%	273202

Table IV
FREQUENCY DISTRIBUTION OF TUPLES

	# of products producing tuples					
	1	2	3	4	5	6
Web	67147	258	18	4	2	0
XML	49379	465	63	13	5	1
DB	62415	609	28	1	32	2
GUI	272795	339	38	23	5	2

B. Resulting dictionaries

Table III gives the output of the analysis before filtering. Since several methods produce two or more tuples, # of tuples is greater than # of methods that match any patterns.

Table IV shows the frequency distribution of tuples obtained from a certain number of software products.

Based on the above results, we set 2 as the threshold for filtering; i.e., tuples appearing in 2 or more software products were included in the dictionaries.

C. Evaluation process

The resulting dictionaries were evaluated by 6 students in a software engineering laboratory. The participants all had experience in software development in Java. Moreover, they evaluated the dictionaries for the domains in which they had some experience.

For each domain dictionary, the tuples <V, DO, IO> were evaluated from the following perspectives.

1) The V-O relation of the tuple is actually used in the domain or in Java programs.
2) The verb, direct-object, and indirect-object are suitable.
3) The tuple is useful for appropriate naming of identifiers.

Based on the perspectives, we prepared several questions for the participants.

The following three questions were based on the first perspective.

- **Q1**
  Is this <V, DO, IO> tuple popular in the domain of the dictionary?
- **Q2**
  Is this <V, DO, IO> tuple popular in common Java programs?
- **Q3**
  Is this <V, DO, IO> tuple popular in another domain? If so, give the domain.

The following question was prepared according to the second perspective.

Table I
LIST OF THE EXTRACTION PATTERNS

structure spec				extraction spec		
return type	method name	parameters	class name	verb	DO	IO
*	verb1 noun2 prepos3 noun4	*	*	verb1	noun2	noun4
*	verb1 prepos2 noun3	*	noun4	verb1	noun4	noun3
*	verb1 noun2	*	noun3	verb1	noun2	noun3
*	verb1 prepos2 noun3	noun4	*	verb1	noun4	noun3
*	verb1 noun2 prepos3	noun4	*	verb1	noun2	noun4
void	verb1	(empty)	noun2	verb1	noun2	(empty)
void	verb1 prepos2 noun3	(empty)	noun4	verb1	noun4	noun3
void	verb1 noun2	noun2	noun3	verb1	noun2	noun3
void	verb1 noun2 prepos3 noun4	*	noun5	verb1	noun2	noun5
void	verb1	noun2	noun3	verb1	noun2	noun3
void	verb1 noun2	noun3	noun4	verb1	noun3	noun4
void	verb1 noun2	noun3	noun2	verb1	noun2	noun3
void	verb1 noun2	(empty)	noun2	verb1	noun2	(empty)
void	verb1 noun2	(empty)	noun2	verb1	noun2	(empty)
void	noun1 verb2	noun3	noun4	verb2	noun1	noun3
void	noun1 verb2	noun3	noun4	verb2	noun1	noun4
void	noun1 verb2	noun1	noun3	verb2	noun1	noun3
noun1	verb2 noun1	noun3	noun1	verb2	noun1	noun3
noun1	verb2 noun1 prepos3 noun4	noun4	noun3	verb1	noun1	noun4
noun1	verb2 noun3 prepos4 noun5	(empty)	noun6	verb2	noun3	noun 3
noun1	verb2 prepos3 noun4	noun5	noun6	verb2	noun6	noun4
noun1	verb2 noun1	(empty)	noun3	verb2	noun1	noun3
noun1	verb2	noun3	noun4	verb2	noun4	noun3
noun1	verb2 prepos3	noun4	noun5	verb2	noun5	noun4
noun1	verb2 prepos3 noun4	*	*	verb2	noun1	noun4
noun1	verb2 prepos3 noun4	(empty)	noun1	verb2	noun1	noun4
noun1	verb2 prepos3	noun4	noun4	verb2	noun4	noun4
noun1	verb2 noun3	(empty)	noun1	verb2	noun3	noun1
noun1	verb2 noun1	(empty)	noun3	verb2	noun3	(empty)
noun1	verb2 noun3	(empty)	noun3	verb2	noun3	noun1
noun1	verb2 noun1	(empty)	noun1	verb2	noun1	(empty)

Table II
TARGET OF EXTRACTION

Web Applications				
BBS-CS 8.0.3	JForum 2.1.8	JGossip 1.1.0.005	mvnForum 1.2.1	Yazd Discussion Forum Software 3.0
Order Portal 1.2.4	Arianne RPG 0.80	JBoss Wiki Beta2	JSP Wiki 2.8.3	SnipSnap 1.0b3
XML				
Castor 1.3	DOM4J 1.6.1	JDOM 1.1.1	Piccolo 1.04	Saxon-HE 9.2.0.5
Xalan-J 2.7.1	Xbeans 2.0.0	Xerces-J 2.9.0	XOM 1.2.4 XPP3 1.1.4	Xstream 1.3.1
Databases				
Axion 1.0 Milestone 2	Apache Derby 10.5.3	H2 1.2.128	HSQLDB 1.8.1.1	Berkeley DB Java Edition 4.0.92
Mckoi 1.0.3	MyOODB 4.0.0	NeoDatis 1.9.22.674	OZONE 1.1	tinySQL 2.26
GUIs				
ArgoUML 0.28.1	BlueJ 2.5.3	Eclipse Classic 3.5.1	jEdit 4.3.1	NetBeans 6.8
vuze 4.3.1.2	LimeWire 5.4			

- **Q4** V, DO and IO are correctly extracted? If you do not think so, identify the incorrect words.

The following three questions were based on the third perspective.

- **Q5**
  Should this <V, DO, IO> tuple be given as a good example to a developer who is naming an identifier in a program in this domain?
- **Q6**
  Should this <V, DO, IO> tuple be given as a good example to a developer who is naming an identifier in a common Java program?
- **Q7**
  Should this <V, DO, IO> tuple be given as a good example to a developer who is naming an identifier in a program in another domain? If so, give the domain.

The participants selected answers for Q1, Q2, Q5 and Q6 from (A) strongly agree, (B) agree, (C) disagree, (D) strongly disagree, or (Z) no idea.

Each of the dictionaries was evaluated by two participants, and each of the participants evaluated two different dictionaries. 15 tuples that appeared in three or more software products and 15 tuples that appeared in two software products were evaluated by the participants. The

Table V
RESPONSE TO Q1

	(A)	(B)	(C)	(D)	(Z)
Web	21	35	7	3	24
XML	44	18	5	7	16
DB	32	36	4	9	9
GUI	42	26	9	6	7

Table VI
RESPONSE TO Q2

	(A)	(B)	(C)	(D)	(Z)
Web	16	29	10	30	5
XML	15	11	17	42	5
DB	22	13	32	17	6
GUI	32	37	9	6	6

tuples were randomly and exclusively selected from the dictionary. However, since the dictionary for the web application domain only contains 24 tuples appearing in three or more products, only 6 tuples were evaluated by two participants.

D. Results of the evaluation

First, we describe the results obtained according to perspective 1.

Table V gives the results for Q1, and Table VI the results for Q2. The sum of the percentages of (A) strongly agree and (B) agree is 62% to 75% for Q1, and 38% to 76% for Q2. These results show that the greater part of the dictionaries consist of domain specific relations. However, several domain independent relations are also included.

Table VII gives the responses for Q3. All dictionaries contain some V-O relations from another domain. The dictionary could be improved by separating these relations into another dictionary.

Next, we give the results for perspective 2. Table VIII gives the responses for Q4. The percentage of tuples with incorrect V, DO, or IO values varies between 6% and 13%. This indicates that the 31 extraction patterns we prepared could be improved.

Finally, we present the results for perspective 3.

Table IX gives the results for Q5, and Table X the results for Q6. The sum of the percentages for (A) and (B) is between 53% and 71% for Q5, and between 30% and 61% for Q6. The results for Q5 and Q1 indicate that we need not only to improve precision of the dictionaries, but also to select and provide the relations that developers prefer. On the other hand, the results for Q6 show that the dictionaries contain many tuples that are suitable as naming examples for common Java programs. These tuples should be separated into a domain independent dictionary.

Table XI gives the responses to Q7. Similar to the results for Q3, the responses for Q7 confirm the need to separate these tuples into another dictionary.

1) Follow-up clarification of the results: Several tuples were determined to be undesirable to developers, despite the tuples being popular either in the domain or in common Java programs. We asked the participants to give reasons for this. The answers are given below.

- The tuple contains uncertain words. (e.g. abbreviation)
- The tuple is common sense for average developers.
- The tuple is used not in the whole domain, but in the programs that dependent on a specific library.

Table VIII
RESPONSE TO Q4

	wrong verb	wrong DO	wrong IO	two or more wrong words
Web	3	1	3	6
XML	5	7	1	12
DB	1	6	5	11
GUI	8	1	0	9

Table IX
RESPONSE TO Q5

	(A)	(B)	(C)	(D)	(Z)
Web	19	32	11	4	24
XML	33	15	10	16	16
DB	35	29	10	10	6
GUI	28	30	13	11	8

E. Discussion

The results for Q2 and Q3 show that the dictionaries contain V-O relations in another domain. The contamination might be as a result of the following two reasons.

- The threshold for filtering is too low to remove noise.
- Since software products generally span several domains, the relations in the domains covered by each of the input products, also span several domains.

The first problem is easy to solve by increasing the number of input products.

Regarding the second problem, we have a solution as described below. First, increase the number of domain categories and input products, and then classify the products into the domain categories on a nonexclusive basis. Thereafter, we track the origin products of the tuples. If a tuple is a candidate of two or more dictionaries, the best dictionary to include the tuple is selected based on the origin products and their domains.

Table X
RESPONSE TO Q6

	(A)	(B)	(C)	(D)	(Z)
Web	13	19	23	30	5
XML	14	13	12	46	5
DB	31	13	24	16	6
GUI	29	26	15	13	7

Table XI
RESPONSE TO Q7 (NUMBERS IN PARENTHESES MEAN THE NUMBER OF SAME ANSWERS)

XML	Data Analysis(1), GUI(1), Parser(1), Resource Management(1), Tree Structure(1), Graph Processing(1)
DB	GUI(5), Web Application(1)

Table VII
RESPONSES TO Q3 (NUMBERS IN PARENTHESES MEAN THE NUMBER OF SAME ANSWERS)

Web	Database(16), I/O processing(6), Common Java Programs(2)
XML	Data Analysis(2), GUI(1), Parser(1), Resource Management(1), Tree Structure(1), Graph Processing(1)
DB	GUI(5), Web Application(1), String Processing(1)
GUI	DB(1), Networking(1), Program Test Cases(1), Archiver(1), Common Java Programs(4),

F. Threats to validity

From the viewpoint of the dictionary, the data assignment is performed randomly, since the tuples are randomly extracted from the dictionaries. However, the input software products were collected intentionally, and the number of products may be insufficient.

Regarding the participants, they are students in a graduate school, and not professional software developers. However, since they have experience of software development through part-time jobs or research projects, they have sufficient knowledge of several target domains to evaluate the dictionaries. Since the number of participants was not large enough, we could not randomly assign participants to dictionaries.

V. RELATED WORKS

This section introduces related works on V-O relations in the source code, and the differences between these works and our approach.

Shepherd *et al.* [10] and Fry *et al.*[9] extracted V-O relations from methods or comments in source code written in an object-oriented language, and used the relations for feature location and aspect mining. Hill *et al.* [11] extracted the <V, DO, IO> tuples from source code and applied the tuples to feature location and aspect mining. Our approach is to build domain-specific dictionaries suitable for providing identifier names.

Høst *et al.* built a dictionary of verbs from the name and body of methods [12], and created several naming rules from the dictionary to detect and fix incorrect naming[13]. Our approach is more relaxed compared to that in [13]; many good examples can be obtained using a verb with an associated object.

VI. CONCLUSION AND FUTURE WORK

This paper proposed an approach for building a domain specific dictionary of verb-object relations. The entries in the dictionary, i.e. tuples consisting of <V, DO, IO>, are extracted from identifiers related to a method.

As a future work, we aim to improve the precision of the dictionaries and to build larger dictionaries from a larger set of source code files. To support naming of identifiers, a system is needed that can suggest an example name to the software developer in the context of the source code. To help achieve good naming, visualization of the dictionaries is helpful for developers who have little experience in certain domains.

ACKNOWLEDGEMENT

This work was supported by KAKENHI (21700031).

REFERENCES

[1] R. K. Fjeldstad and W. T. Hamlen, "Application program maintenance study: Report to our respondents," in *Proceedings GUIDE 48*, April 1983.

[2] T. A. Corbi, "Program understanding: challenge for the 1990's," *IBM Syst. J.*, vol. 28, no. 2, pp. 294–306, 1989.

[3] A. von Mayrhauser and A. M. Vans, "Identification of dynamic comprehension processes during large scale maintenance," *IEEE Trans. Softw. Eng.*, vol. 22, no. 6, pp. 424–437, 1996.

[4] N. Pennington, "Comprehension strategies in programming," in *Empirical studies of programmers: second workshop*. Norwood, NJ, USA: Ablex Publishing Corp., 1987, pp. 100–113.

[5] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? a study of identifiers," in *ICPC '06: Proceedings of the 14th IEEE International Conference on Program Comprehension*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 3–12.

[6] Y. Hayase, M. Ichii, and K. Inoue, "A novel approach for building a thesaurus for program comprehension (in Japanese)," in *Proceedings of winter workshop 2008 in Dogo*, no. 3, 2008, pp. 33–34.

[7] Oracle Corporation, "The java tutorials," http://java.sun.com/docs/books/tutorial/java/javaOO/index.html.

[8] Sun Microsystems, "Java platform, standard edition 6 API specification," http://java.sun.com/javase/6/docs/api/.

[9] Z. P. Fry, D. Shepherd, E. Hill, L. Pollock, and K. Vijay-Shanker, "Analysing source code: looking for useful verb–direct object pairs in all the right places." *IET Software*, vol. 2, no. 1, pp. 27–36, 2008.

[10] D. Shepherd, L. Pollock, and K. Vijay-Shanker, "Towards supporting on-demand virtual remodularization using program graphs," in *AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development*. New York, NY, USA: ACM, 2006, pp. 3–14.

[11] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of NL-queries for software maintenance and reuse," in *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 232–242.

[12] E. W. Høst and B. M. Østvold, "The programmer's lexicon, volume i: The verbs," in *SCAM '07: Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 193–202.

[13] ——, "Debugging method names," in *Genoa: Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 294–317.