An Approach for Knowledge Extraction from Source Code (KNESC) of Typed Programming Languages

Azanzi Jiomekong[1(✉)] and Gaoussou Camara[2]

[1] UMMISCO, Faculty of Science, University of Yaoundé I, Yaoundé, Cameroon
jiofidelus@gmail.com
[2] EIR-IMTICE, Université Alioune Diop de Bambey, Bambey, Senegal
gaoussou.camara@uadb.edu.sn

Abstract. Knowledge extraction is the discovery of knowledge from structured and/or unstructured sources. This knowledge can be used to build or enrich a domain ontology. Source code is rarely used. But implementation platforms evolve faster than business logic and these evolutions are usually integrated directly into source code without updating the conceptual model. In this paper, we present a generic approach for knowledge extraction from source code of typed programming languages using Hidden Markov Models. This approach consist of the definition of the HMM so that it can be used to extract any type of knowledge from the source code. The method is experimented on EPICAM and GeoServer developed in Java and on MapServer developed in C/C++. Structural evaluation shows that source code contains a structure that permit to build a domain ontology and functional evaluation shows that source code contains more knowledge than those contained in both databases and meta-models.

Keywords: Knowledge extraction · Ontology learning
Hidden Markov Models · Source code · JAVA · C · C++

1 Introduction

Domain ontologies can be constructed using a bottom-up approach, using data sources [6]. Several types of data sources can be used: texts (specifications, analysis and design documents, user manuals, information on forums and blogs, etc.) [7,15], databases [14,15], XML files [15], UML/Meta-model diagrams [4] and source code [2,14].

Source code is any fully executable description of a software designed for a specific domain: medical, industrial, military, communication, aerospace, commercial, scientific, etc. It can be used for the collection, organization, storage and communication of information. It is designed to facilitate repetitive tasks or to process information quickly. To do this, it must be able to capture a set of knowledge of the domain. For example, EPICAM, an epidemiological surveillance

Á. Rocha et al. (Eds.): WorldCIST′18 2018, AISC 745, pp. 122–131, 2018.
https://doi.org/10.1007/978-3-319-77703-0_12

platform [11] allows health personnels to collect and share health information. Then, it can capture knowledge of epidemiological surveillance.

Source code is written in a programming language that can be typed or not. A programming language is a formal language that specifies a set of statements that can be used to produce different types of output. They are distinguished according to the underlying programming paradigm. Some are designed to support a paradigm (Java for example supports object-oriented programming, Haskell supports functional programming), while other support several paradigms (C++, C#).

Knowledge extraction from source code is rarely addressed in literature. Bontcheva and Sabou [2], Zhao et al. [14] proposed to extract some aspect of ontological knowledge from source code. Their approach have two drawbacks: firstly, it is difficult to use it to extract any type of ontological knowledge and secondly, it is difficult to adapt it from one source code to another.

In a previous work, we proposed an approach for knowledge extraction from JAVA source code [1]. In this paper, we generalize this approach for knowledge extraction from source code of typed programming languages. Then, the Sect. 2 presents an overview of Knowledge extraction principles and methods. In Sect. 3, we present our approach, in Sect. 4 we present the experiments and evaluation, and, in Sect. 5, we conclude.

2 Knowledge Extraction

According to Unbehauen et al. Knowledge Extraction is defined as "the creation of knowledge from structured (relational databases, XML) and unstructured (text, documents, images) sources" [12]. Knowledge that results from the extraction process must be readable and interpretable by the machine and what distinguishes it from the extraction of knowledge found in areas such as Automatic Language Processing or Data Warehouses are the result of the extraction. Indeed, when extracting knowledge, one do not expect only to obtain structured information or the transformation into relational schema, but also the semantics that this information can have. It is for this reason that learning ontologies can be considered as a sub-domain of knowledge extraction [12]. Knowledge extraction uses a wide range of methods such as machine learning, knowledge acquisition, natural language processing, information retrieval, artificial intelligence, reasoning and database management [12,15].

Knowledge extracted from data sources can be very useful when building/enriching a domain ontology. In fact, more the knowledge of the field evolves and more the experts are distributed, more experts are not easily accessible and knowledge is likely to be incomplete, subjective and even obsolete. To overcome the difficulties of collaborative approaches, users often turn to other data sources such as dictionaries [7], Web documents [14], database schemas [14], meta-models [4], UML diagrams [4], source code [2,14], etc. to extract knowledge.

Knowledge extraction from source code aims to extract knowledge embedded in source code of software through an automated process. The technique generally used is to do a reverse engineering to analyse the source code in order to

extract knowledge [2,14]. Some works propose the use of source code to enrich an ontology constructed from schemas (for example, a database schema) [14], others use user interfaces only [14], others propose to use the source code to generate concepts [2,14]. Knowledge extraction from source code can be performed by statistics, symbolic and multi-strategies approaches [10]. Symbolic techniques are more precise, more robust, but they can be complex to implement, difficult to generalize, inflexible, and it can be costly to adapt from one source code to another. Statistical methods are more computable, general, scalable, easy to adapt from one source code to another. For these reasons we will used a statistical approach to extract knowledge from source code.

3 KNESC: A Generic Approach for Knowledge Extraction from Source Code

3.1 Source Code vs HMM

Hidden Markov Model (HMM) [8,9] is a statistical model composed by: **(1)** $Q = \{q_1, q_2, \ldots, q_n\}$, a set of states; **(2)** $O = \{o_1, o_2, \ldots, o_n\}$, a set of observations; **(3)** $T : q_t \rightarrow q_{t+1}$, a unidirectional transition function between states; **(4)** $S : S(q) = o$, a function for observation emission [8,9]. In this definition, q_i are the states of the model, q_1 is the initial state and q_n is the final state, o_i are the observed symbols, each transition of the T function is associated with its probability to be taken from a state q_t at time t to another state q_{t+1} at time $t+1$. Each observation is associated with its probability of being emitted by a state q_t at a time t. HMMs are generally used for pattern recognition, automatic voice processing, automatic natural language processing, character recognition [8], etc.

Source code can be modelled using HMMs. In fact, source code can be seen as a text composed of a set of words organized by following a certain syntax. Because source code follows a syntax, we assume that we can define the order in which different words are entered by the programmer. We assume that before entering the first word (at time t), the programmer reflects on the label of that word and as a function of it, defines the label of the next word (at time $t + d_t$) and so on. For example, in C, before entering "struct", the programmer knows its label (a word that describes a data structure) and the label of the next word (the name of data structure closed to domain vocabulary). Thus, the current word depends only on the current label, the next label depends on the previous label, and so on. An example of HMM annotated with class labels for Java source code is given in Fig. 1. In the next section, we present our approach composed of four main steps: Model definition, model training, model using and knowledge extracted validation.

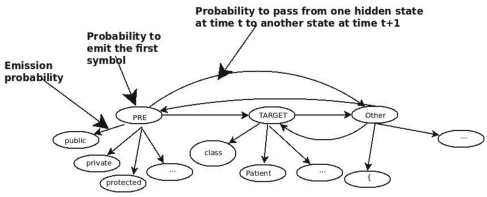

Fig. 1. HMM example

3.2 HMM Definition

To define the structure of the HMM, we studied (manually) the organization of the source code of the typed languages. Generally, data structures, attributes, conditions are surrounded by one or more words. Some of these words are predefined in advance in the programming language. To label the source code, we have defined four labels, corresponding to four hidden states of the HMM: **(1) PRE:** corresponding to the preamble of the information. This preamble is usually defined in advance for typed programming languages; **(2) TARGET:** is the target, i.e., the information sought. This information may be preceded by one or more words belonging to the PRE set. The information we are looking for are the names of the data structures, the attributes, some relationships between data structures. They are usually preceded by a meta-data that describes them. For example, in Java, the meta-data "class" allows to identify a class. The meta-data (e.g. "class") and the data ("Patient", for example) will all be marked as targets; **(3) POST:** any information that follows the information sought. In some cases, POST is a punctuation character or a braces (";" or "}"); **(3) OTHER:** any other word in the vocabulary that neither precedes nor follows the information sought.

3.3 HMM Training

Once the model structure is defined, the parameters of the transition and emission models must be estimated from the training data. To do this, we assume that we have access to T corpus labelled f_t knowing that f_t is not just a sequence of words but a sequence of word pairs with the word and its label as presented by the Fig. 1. To train the model, we assume that we can define the order in which different words are entered by the programmer. We assume that before entering the first word, the developer reflects on the label of that word and as a function of it, defines the label of the next word and so on. The current word depends only on the current label, the next label depends on the previous label, and so on. The process continues until the end of the file. We model this situation by the equation:

$$f_t = [(w_1^t, e_1^t), ..., (w_d^t, e_d^t)], \tag{1}$$

$$words(f_t) = [w_t^t, ..., w_d^t], \tag{2}$$

$$labels(f_t) = [e_1^t, ..., e_d^t]. \tag{3}$$

Where w_i and e_i are sentences and labels of f_i files respectively. In practice, w_i are concepts, properties, axioms or words that make up the rules. When they are concepts, they are composed of properties and semantic relations with other concepts. They are then labelled by e_i, and represent the hidden states of the HMM. From the training data, we can extract the statistics for the HMM:

– On the first label: $P(q_1)$ given by the formula 4. The a priori probability that the first label is 'a' is the number of times the first label in the training corpus is equal to 'a' in all documents divided by the number of documents.

$$P(H_1 = a) = \frac{\sum_t freq(e_1^t = a, f_t)}{T}. \tag{4}$$

– On the relation between a word or a sentence and its syntactic class $P(S_k \mid q_k)$ (formula 5). Conditional probability that the k^{th} word is 'w' knowing that the label is 'b' is the number of times I have seen the word 'w' associated with the label 'b' in the document f_t divided by the number of times I see the label 'b' associated with any other word in the f_t documents. For example, "Patient" can be a concept, a property, but can not be a rule.

$$P(S_k = w \mid q_k = b) = \frac{\sum_t freq((w, b), f_t)}{\sum_t freq(('*, b), f_t)} \tag{5}$$

– On the relation between the adjacent syntactic label $P(q_{k-1} \mid q_k)$ (formula 6). Probability q_{k+1} is equal to label 'a' knowing that q_k is equal to label 'b' (previous hidden state) is the number of times 'a' follows 'b' in the training data divided by the number of times that 'b' is followed by any other label.

$$P(H_{k+1} = a \mid H_k = b) = \frac{\sum_t freq(b, a), label(f_t) + 1}{\sum_t freq(b, *'), label(f_t) + 1}. \tag{6}$$

To avoid zero probabilities for transitions or emissions that do not occur in the training data, we have added a smoothing term (+1).

For example, let us consider the HMM of Fig. 1. Then, training corpus for identifying concepts and properties would be: [("public", PRE), ("class", TARGET), ("Patient", TARGET), ("extends", TARGET), ("ImogEntityImpl", TARGET), ("{", OTHER), (...), ("int", TARGET), ("age", TARGET), ...]. The training phase is an automatic phase. In fact, once the sets PRE, POST and OTHER have been defined, a simple algorithm can make it possible to identify them automatically in the training data and to count the occurrence of each element in order to calculate the above probabilities.

3.4 Knowledge Extraction

During this phase, potentially relevant knowledge will be identified and retrieved, some entities will be re-encoded. The problem of extracting knowledge from the

source code has been reduced to the problem of syntactic labelling. This is to determine the syntactic label of the words of a text [8]. In our case, it will be a matter of assigning a tag to all the words of the source code and extracting the words marked as target word. This problem can be solved using HMM [8,9]. To do this we retrieve a sequence of states $V(X \mid M)$ which has the greatest probability of producing an observation sequence. For example, in our case, it will be to find for the files f_1, ..., f_n, a sequence q_1, ..., q_n that is plausible. For this, the formula 7 will be used to determine the most plausible strings.

$$P(X \mid M) = argMax_{q_1 \ldots q_l \in Q^l} \prod_{k=1}^{l+1} P(q_{k-1} \rightarrow q_k) P(q_k \uparrow x_k). \tag{7}$$

From the extracted knowledge, two candidate terms to be concepts are related if one is declared in the structure of the other. One may identify three types of relations:

- Association: if two classes 'A' and 'B' are candidate terms to be concepts and class 'B' is declared as attribute of class 'A', then classes 'A' and 'B' are related. Class 'A' is the domain, class 'B' the range and the cardinality of the association will be used to express relations of higher arity.
- Taxonomy: if two classes 'A' and 'B' are candidates terms to be concepts and class 'B' extends the class 'A' (in JAVA, the keyword "extends" is used), then, one can define a taxonomic relation between the classes 'B' and 'A'.
- Attributes: if a class 'A' is a candidate term to be a concept and contains the attributes 'a' and 'b' of basic data types (integers, string, etc.), then, 'a' and 'b' are attributes of class 'A'.

3.5 Knowledge Validation

The knowledge obtained can be validated manually by a domain expert or semi-automatically [3]. In fact, our goal is not to provide accurate knowledge, but to facilitate the work of knowledge engineers and domain experts during the phase of knowledge elicitation of the ontology engineering process.

In the next section, we experiment the approach on EPICAM and GeoServer developed in Java and MapServer developed in C/C++. All our experiments have been coded in Java and during it, we considered that the knowledge we need are concepts, properties, axioms and rules.

4 Experiment and Validation

To validate our approach, we experiment it on JAVA and C/C++ programming languages. In the next sections, we exploit the structure of Java source code to define the HMMs (HMM for concepts, properties, axioms and, HMM for rules) and we use these HMMs to extract the knowledge.

4.1 Knowledge Extraction from JAVA Source Code

Defining HMM Model for JAVA. A set of Java source code allowed to identify the elements corresponding to the sets PRE, TARGET, POST and OTHER. Concepts, properties, and axioms can be identified by class names, class attributes, relationships between classes respectively. Rules can be identified under conditions (if (condition) - else). In [1], we present the definition of two HMMs for JAVA source code: one for the identification of concepts, properties and axioms and the second for the identification of rules. In these HMMs, each state emitted a word corresponding to a word from the source code; Elements that cannot be fully enumerated (for example, the TARGET state), a name is used to designate all the symbols emitted by this state (for example, the word "data" is used to designate all the terms emitted by the state TARGET).

Meta-data allowed to identify the candidates terms to be concepts, properties of concepts and axioms. For example, if we have extracted the following terms: "package edu.hospital.patientRecord ... class Patient extends Person ... int age ... List<Exam> listExam", then, a simple algorithm can be used to identify every element:

- **"package edu.hospital.patientRecord":** identify which package contains all other elements and can be used to identify the class hierarchy;
- **"class Patient extends Person":** means that "Patient" and "Person" are candidate terms to be concepts and there is a hierarchical relationship between "Patient" and "Person";
- **"int age, List <Exam> listExam**": means that "age" and "listExam" are properties of the concept "Patient" and there is a relationship between "Patient" and "Exam" class;
- **"List<Exam> listExamen":** allows to define an axiom because it can be translated by: "a patient has one or more exams".

Training HMM on Data. Currently, the corpus is labelled by hand. In our case, since it was possible to identify the PRE and POST sets, we automatically labelled our corpus by defining an algorithm which, based on the source code, constructs the transition model between hidden states and emission models between different states. A set of JAVA source codes were downloaded from github and from these source codes we trained the HMMs. This data source consists of 24 files. These files contain 1186 instructions and 12 conditions.

The HMMs we have just construct can be used to extract candidate terms to be concepts, properties, axioms and rules of any Java source code.

Knowledge Extraction from EPICAM. The EPICAM platform [11] is an Open Source platform for epidemiological surveillance of tuberculosis. It helps health personnels to collect and share useful health information. Because EPICAM is developed in Java, we will exploit the structure of the Java source code to extract the knowledge. From this source code, we have extracted 377, 5205, 260, 263 candidates terms to be concepts, properties, axioms and rules respectively.

Knowledge Extraction from GeoServer Source Code. GeoServer is an Open Source map server developed in Java [5]. It allows users to edit, process and share geospatial data. The source code downloaded from github contains 13038 files and 2150161 instructions. The HMM we have just presented was used to extract the candidate terms to be concepts, properties, or axioms from GeoServer source code. We have extracted 3522, 22020, 1404 candidates terms to be concepts, properties and axioms respectively.

4.2 Knowledge Extraction from C Source Code

Defining HMM Structure for Concepts. A set of source code written in C was downloaded from github and it was used to identify the elements corresponding to the PRE, POST, TARGET and OTHER sets automatically. The candidates terms to be concepts is identified by the name of data structures (for example, "struct Coordinates {" identifies "Coordinates" as a candidate term to be a concept). The HMM is defined by:

- $PRE = \{struct\}$, a set of words that precede TARGET.
- $TARGET = \{struct, w_i\},\ \forall i, w_{i-1} \in PRE$, the set of all words that we are looking for.
- $POST =$ { "{,", * }, the end of the condition.
- $OTHER = \{w_i\}, w_i \notin PRE \wedge w_i \notin TARGET$, the set of all other words.

Training HMM on Data. As we did for JAVA source code, the HMM built in the previous step will automatically run on a training data. To do this, a set of C source codes was downloaded from github. This data source consists of 24 files. The HMM obtained can be used to extract knowledge from any source code written in C language.

Extracting Knowledge from MapServer Source Code. MapServer is an Open Source map server developed at the University of Minnesota [13]. Its source code contains 711 files and 425266 instructions. We used the HMM trained to extract candidate terms to be concepts and have extracted 294 terms. We noticed without being surprised that certain terms (e.g., x, y, xy, Map, Coords, Line, Point, data, etc.) are both in the list of MapServer and GeoServer candidate terms. Indeed, both applications are in the same domain.

4.3 Evaluation

To validate our approach, we evaluated the knowledge extracted from EPICAM source code. To do it, we have considered structural evaluation (representation of the ontology has a graph) and functional evaluation (conceptualization of the ontology) [3]. We have compared the knowledge extracted to a gold standard [3] manually constructed with domain experts in our previous work using EPICAM meta-model and database. Structural evaluation shows that from the

source code, we can extract a structure allowing to build a domain ontology and functional evaluation shows that source code contains more knowledge than meta-model and database; It is the only data source containing the rules.

4.4 Advantages of the Approach

To show the benefits of our approach, we compare it to a parser-based approach. To do it, we defined four comparison criteria:

- **Genericity:** our approach uses a set of simple keywords to identify terms in the source code in order to train the model. This makes it possible to extract any type of terms from any type of source code. With the parser approach, there are two possibilities: define a generic parser that uses a regular expression to identify terms or develop a parser for each programming language. In both cases, this work is not obvious for a knowledge engineer who does not always have the knowledge on programming or on the definition of regular expressions (the syntax is less intuitive than what we propose). If we take the example of our experiment, we used the same source code to extract terms from JAVA and C source code.
- **Ease to use:** with our approach, to modify the elements to extract, the knowledge engineer modifies the sets PRE, POST, OTHER, which is less difficult than to define a regular expression or modify the source code of a parser.
- **Difficulties in the implementation:** the development of a tool based on our approach is more difficult than the development of a parser because there are many libraries allowing the development of the parsers. However, once the tool is developed, it is easy to use.
- **Performance:** unlike parsers, with our approach, we usually have false positives (terms that were extracted and were not affected by the extraction). But, by training correctly the model, one have good performances.

5 Conclusion

To conclude, we propose an approach consisting of a generic method allowing the extraction of knowledge from source code of typed languages. It consists of defining a HMM by providing PRE, POST and OTHER sets, training the HMM on data sources and use it for any software for this programming language. This approach can then be extended to any programming language because all have a structure making it possible to define PRE, POST and OTHER sets. We experiment this approach by extracting knowledge from EPICAM and GeoServer developed in Java, and MapServer developed in C/C++. The experimentations were conclusive.

Our approach have therefore a number of shortcomings to be addressed. It was experimented on typed programming language such as JAVA, C, and C++ having a particular structure. It would be interesting to experiment it in another type of programming languages like functional languages (Haskel, Lisp) or untyped languages (PHP).

References

1. Azanzi, J., Gaoussou, C.: Knowledge extraction from source code based on Hidden Markov Model: application to EPICAM. In: Proceedings of the 14th ACS/IEEE International Conference on Computer Systems and Applications (2017)
2. Bontcheva, K., Sabou, M.: Learning ontologies from software artifacts: exploring and combining multiple choices. Seman. Web Enabled Softw. Eng. **17**, 235 (2014)
3. Dellschaft, K., Staab, S.: Strategies for the evaluation of ontology learning. In: Buitelaar, P., Cimiano, P. (eds.) Bridging the Gap between Text and Knowledge Selected Contributions to Ontology Learning and Population from Text. IOS Press (2008)
4. Djuric, D., Gasevic, D., Devedzic, V.: Ontology modeling and MDA. J. Object Technol. **4**, 109–128 (2005)
5. Foundation OSG: Geoserver (2014). http://geoserver.org/
6. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. Springer, New York (2007)
7. Maedche, A., Staab, S.: Semi-automatic engineering of ontologies from text. In: Proceedings of the 12th Internal Conference on Software and Knowledge Engineering Chicago, USA (2000)
8. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Pearson Education, Upper Saddle River (2003)
9. Seymore, K., Mccallum, A., Rosenfeld, R.: Learning Hidden Markov Model structure for information extraction. In: AAAI 1999 Workshop on Machine Learning for Information Extraction, pp. 37–42 (1999)
10. Shamsfard, M., Abdollahzadeh Barforoush, A.: The state of the art in ontology learning: a framework for comparison. Knowl. Eng. Rev. **18**(4), 293–316 (2003)
11. UMMISCO, MEDES, PNLT, CPC: Plate-forme de surveillance de la tuberculose (2016). http://github.com/UMMISCO/EPICAM
12. Unbehauen, J., Hellmann, S., Auer, S., Stadler, C.: Knowledge extraction from structured sources. In: Search computing, pp 34–52. Springer, Heidelberg (2012)
13. University of Minnesota: MapServer (2017). www.mapserver.org
14. Zhao, S., Chang, E., Dillon, T.S.: Knowledge extraction from web-based application source code: an approach to database reverse engineering for ontology development. In: IRI, IEEE Systems, Man, and Cybernetics Society, pp. 153–159 (2008)
15. Zhou, L.: Ontology learning: state of the art and open issues. Inf. Technol. Manage. **8**, 241–252 (2007)