Knowledge Extraction from source code based on Hidden Markov Models

Knowledge Extraction from Source Code

Azanzi Jiomekong
Po. Box 812 Yaoundé, Université de Yaoundé I, IRD-UMI 209, UMMISCO,Faculté des Sciences, Cameroon
E-mail: jiofidelus@gmail.com
Gaoussou Camara
Po. Box 812 30 Bambey, EIR-IMTICE, Université Alioune Diop de Bambey, Sénégal
E-mail: gaoussou.camara@uadb.edu.sn

Abstract. Large software systems evolve rapidly and these evolutions are usually integrated directly into source code without updating the conceptual model. As a consequence, implementation platforms evolve faster than business logic. Indeed, when extracting knowledge to enrich or build an ontology, business logic is not always a complete data source. To solve this problem, some authors have suggested to adopt an ontology learning approach in order to extract knowledge from the source code. In this paper, we show how to realize this task using Hidden Markov Models. Experiments on EPICAM(a tuberculosis surveillance system developed in JAVA) shows the relevance of this approach.

Keywords: Knowledge Extraction, Ontology Learning, Hidden Markov Model, Programming language, JAVA, Source Code, EPICAM

1. Introduction

The popularity of computer applications and the huge growth of new software development technologies has brought about the development of many applications and services [1] such as e-epidemiology and e-health platforms. Large software consist of many modules (small programs), possibly written by different programmers [2] and allow a great community of people to use and share a set of services [1]. To develop them, one may follows an approach among the existing ones such as Waterfall, Iterative development, Prototyping, Spiral and Rapid Application Development (RAD) [3]. In these approaches, interrelated activities are performed during the development process [3]. On the one hand, in developing and using large software, some problems could be encountered:

- Software requirements could be articulated on the web through historical email messages, discussion forums, etc. Once asserted, there is generally no "software requirements specification documents" [4]. In github for example, adding comments and some code updates are not always done in parallel with the updating of the conceptual model.
- In some Open Source software with large communities, many developers contribute to source code with their own vocabularies [4].
- Sometimes, developers focus on coding features rather than ensuring that they have a solid and complete documentation that facilitates the integration of newcomers [4].
- Changing the needs during the development process is still not managed by software process models. As a consequence, software projects do not always meet their expectations in terms of functionality, cost and delivery schedule [4].
- Integrating the evolution of platforms directly into source code can make new programmers take too much time to grab the source code [4, 5].

On the other hand, in using large software, some B2B applications require an effective communication between machines and, between human and machines [6]. For example, in epidemiological domain, one may need in real time climatic data from an accessible data source in order to explain the evolution of an outbreak.

In order to solve the above problems, one solution consists in establishing a standard for software development to ensure that every code submitted makes the system coherent [4]. But this solution has at least two limits: it requires the programmer to learn new sets of standards before coding and, one must set new standards each time new technologies are integrated into the project, and make it known to developers. These constraints can slow the development process. To make the source code coherent with the conceptual model and the data sources to inter-operate, an ontology that model the domain knowledge may be a good solution.

Studer and al. [7] defined an ontology as "a formal, explicit specification of a shared conceptualization". In the context of domain ontologies, conceptualization refers to the abstract model of the domain which is machine readable, and where all the elements are explicitly defined and accepted by a group of domain expert. Several domain ontologies define and organize relevant knowledge about activities, processes, organizations and strategies, in order to facilitate information exchange between machines and, between human and machines [6].

Building domain ontologies require the access to domain knowledge owned by domain experts or contained in formalized data sources. There is a lot of value in creating domain ontologies using existing documents of the domain. In fact, the domain evolves, experts are not easily accessible, the knowledge provided by domain experts is likely to be incomplete, subjective and even obsolete.

Domain ontologies are usually constructed by either using a top-down, middle-out, or bottom-up approach. With the bottom-up approach, data sources are used to build the ontology [6, 8]. Several types of data sources can be used [9]: Texts (documents of the domain, specifications, analysis and designed documents, user manuals, information from forums and blogs) [1, 6, 8, 10–13], databases [8, 11, 12, 14], XML files [8, 11] and UML / Meta-model diagrams [8, 15].

Source code is rarely used. Whereas, names of classes/data structures and variables are generally close to terms of the domain. On the other hand, the business logic that can be difficult to represent in the conceptual model may be directly put into the source code. Moreover, after initial analysis and design, further changes are often reported directly into the source code without updating conceptual models. Therefore, source code may evolve over time. For example, github codes undergo continuous modifications without appropriate updates of conceptual model.

Source code files are sometimes in thousands, the whole application can contain millions of lines of code. Thus, it can be costly to extract knowledge from these files manually. A solution to this problem consists in designing a tool that automatically extracts knowledge from the source code in order to build and/or to update the domain ontology [1, 14, 16].

According to Unbehauen and al. [17] "Knowledge Extraction is the creation of knowledge from structured (relational databases, XML) and unstructured (text, documents, images) sources". To automatically extract knowledge from data sources, there are symbolic techniques and statistical techniques [12]. Ontology learning applies statistical techniques, symbolic techniques or the mix of two to (semi-)automatically extract the ontological knowledge from data sources and build an ontology. In symbolic techniques, the extraction process consists in examining text fragments that match some predefined rules, looking for lexico-syntactic patterns corresponding for instance to taxonomic relations or scanning for various types of templates related to ontology elements. Several symbolic methods have been proposed in the literature to extract ontological knowledge from source code: **(1)** Ganapathy and Sagayaraj [16] extract metadata (for concepts and properties identification) using a tool called QDox generator[1]. **(2)** Shuxin Zhao and al. [14] propose a rule-based approach to extract knowledge from source code of Web applications. **(3)** Kalina Bontcheva and Marta Sabou [1] construct an ontology from multiple data sources (discussion forums, documentation, comments and source code) using the extraction tools integrated in GATE[2] software that are based on symbolic approaches. Although very powerful for particular domains, these methods are inflexible because of their strong dependency on the structure of the data. Statistical techniques are more general [12] and can be adapted for knowledge extraction from various source codes.

In this paper, we tackle the problem of knowledge extraction from source code written in Java program-

[1]https://github.com/paul-hammant/qdox

[2]https://gate.ac.uk/

ming language. More precisely, we propose a method based on Hidden Markov Models to extract relevant knowledge.

The rest of this paper is organized as follows. In section 2, we present an overview of ontology learning. In section 3 we describe our approach. In section 4, we experiment and evaluate the approach. Finally, in section 5, we conclude and present future works.

2. Ontology Learning

Acquiring knowledge for building an ontology from scratch, or for refining an existing ontology is costly in time and resources. Ontology learning techniques are used to reduce this cost during the knowledge acquisition process. Ontology learning refers to the extraction of ontological knowledge from unstructured, semi-structured or fully structured data sources in order to build an ontology from them with little human intervention [6, 11, 12]. In this section, we present the basic ontology components, data sources generally used for ontology learning, some ontology learning techniques and ontology learning evaluation.

2.1. Basic ontology components

An ontology is composed of different components [6]:

- *Concept* is a collection of objects that have similar properties. For instance $Health\_facility$ is the concept of all health facilities including health centers and clinics.
- *Property* is used to describe the characteristics of Individuals of a concept. They are composed of attributes properties and relations. Attributes are properties whose values are data types. For instance, *age*, of type *Integer* is the attribute of concept $Person$. Relations are special attributes whose values are individuals of concepts. For instance, *examined_in* defines relationship between the concept *Person* and the concept $Health\_facility$ ("A person is examined in a health facility").
- *Axiom* is used to model statements that are always true. They cannot be simply described by existing components. For example, the assertion "the concepts *Men* and *Women* are disjoints" is an axiom.
- *Rule* is a statement in the form $\frac{P_1,...,P_n}{P}$, this means that if the statement P is true, then, the statements $P_1, ..., P_n$ are true. Rules are used to infer new knowledge.
- *Individual* is instance of concept and corresponds to a concrete object. For example, from the concept *Person*, *Bob* is an individual.

To identify these ontological components, terms (linguistic realization of domain-specific concepts [13]) will be extracted from data sources.

2.2. Data sources for ontology learning

The process of developing an ontology requires knowledge extraction from any relevant sources. There are several possible sources of knowledge: domain experts or unstructured, semi-structured, and structured sources [9].

2.2.1. Domain experts

A domain expert is a person knowledgeable about a domain. To get the knowledge from domain experts, a knowledge engineer conduct interviews. This process might lead to knowledge loss or even worse, introduce errors because misunderstandings frequently arise in human communication. Additionally, domain evolves, experts are not easily accessible and the knowledge provide by domain experts is likely to be incomplete, subjective and even obsolete [9].

2.2.2. Unstructured data sources

Unstructured data sources contain data that do not have a pre-defined organization. These are all kinds of textual resources (Web pages, manuals, discussion forum posting, specification, analysis and conception document, source code comments) and multimedia contents (videos, photos, audio files) [1, 9, 10, 12, 13]. Unstructured sources are the most numerous and can make it possible to extract a more complete knowledge. However, they are easily accessible to human information processing only. For example, extracting formal specification from arbitrary texts is still considered a hard problem because sentences might be ambiguous and, in some cases, no unique correct syntactic analysis is possible [9].

2.2.3. Structured data sources

Structured data sources contain data described by a schema. The advantage of using these data sources is that they contain directly accessible knowledge [9]. Some structured data sources include:

- Ontologies: Before constructing an ontology from scratch, one may look at other ontologies that can be (maybe partially) reused [6, 9, 12, 18];

- knowledge base: In knowledge base, one can generate discovered rule as input to develop a domain ontology [19];
- Database schema: Terms to be used to build an ontology can be extracted from database schema [14, 20, 21].

2.2.4. Semi-structured data sources

Semi-structured data sources contain data having a structure that already reflect part of the semantic interdependencies. This structure makes it easier to extract a schema [9]. Semi-structured data sources are:

- Folksonomies/thesaurus: The advantage of using folksonomies or/and thesaurus to build an ontology is that, they reflect the vocabulary of their users [22, 23].
- XML (Extensible Markup Language): The aim of XML data conversion to ontologies is the indexing, integration and enrichment of existing ontologies with knowledge acquired from XML documents [24].
- UML/meta-model: To learn an ontology from UML or/and meta-model, one approach is to extract OWL class and properties from diagrams or to use Ontology UML Profile (OUP) which, together with Ontology Definition Meta-model (ODM), enable the usage of Model Driven Architecture (MDA) standards in ontological engineering [15].
- Entity-relation diagram: They are used to describe the need and the type of information that will be managed by the database. They can then be used to learn ontologies [25].
- Source code [1, 14, 16]: Generally, in source code, the names of data structures, variables, functions are closed to the terms of the domain.

A lot of work has been done on ontology learning from text, databases, XML files, vocabularies, and the use of ontologies to build or enrich other ontologies. This resulted in a wide range of models, techniques and tools for the generation of knowledge structure that can be considered as intermediate process when constructing ontologies. However, we noted only three works dealing with ontology learning from source code [1, 14, 16].

2.3. Ontology learning techniques

To extract knowledge from data sources, many techniques are used. Shamsfard and Barforoush proposed a classification of learning techniques by considering symbolic, statistics and multi-strategies [12].

2.3.1. Symbolic based techniques

A symbolic method can be rule based, linguistic based or pattern based.

1. A rule-based model is represented as a set of rules, where each rule consists of a condition and an action [11].
   (a) *Logical-based rules:* Learning methods may discover new knowledge by deduction (deduce new knowledge from existing ones) or induction (synthesize new knowledge from experience). Inductive logic programming can for example used to learn new concepts from data [12, 13, 26].
   (b) *Association-based rules:* Aim at finding correlations between items in a dataset. This technique is generally used to learn relations between concepts [10, 12, 13] and can be used to recognize a taxonomy of relations or to discover gaps in conceptual definitions [12, 13, 27].
2. *Linguistic-based* approaches (syntactic analysis, morpho-syntactic analysis, lexico-syntactic pattern parsing, semantic processing and text understanding) are used to derive knowledge from text corpus. This technique can be used to derive an intentional description of concepts in the form of natural language description [27].
3. *Pattern-based / Template-driven* approach allows to search for predefined keywords, templates or patterns. Indeed, a large class of entity extraction tasks can be accomplished by the use of carefully constructed regular expressions (regex). But, robust extraction requires the use of complex expressions. This effort is generally reduced by regular expression learning. Yunyao [28] proposes a method to learn and improve regex given an initial regex and labeled examples.

Symbolic techniques are precise and robust, but can be complex to implement, and difficult to generalize [12]. Moreover, they can be costly to adapt from one source code to another.

2.3.2. Statistic based techniques

Statistic analysis for ontology learning is performed on data gathered from input, to build a statistical model [11, 12]. Several statistical methods for extracting ontological knowledge have been identified in the literature [11–13].

1. *Co-occurrence and collocation analysis* are related and can be defined as the occurrence of some words in the same sentence, paragraph or document. Such occurrences hint a potential direct relation between words. After extracting terms from HTML documents, Brunzel and Marko [29] used co-occurrence frequencies to discover terms that are siblings to each other.
2. *Clustering* can be used to create groups of similar words (clusters) which can be regarded as representing concepts, and further hierarchically organize these clusters. This technique is generally used for concept learning by considering clusters of related terms as concepts, and learning of taxonomy relations by organizing these groups hierarchically [13]. Ontology alignment can use agglomerative clustering to find candidate groups of similar entities in ontologies [27].
3. A Hidden Markov Model (HMM) defines a generative statistical model that is able to generate data sequences according to rather complex probability distributions and that can be used for classifying sequential patterns [30–32]. Zhou and Su [33] used HMM for Named Entity Recognition; Maedche and Staab [10] used the n-gram models based on HMMs to process documents at the morphological level before supplying them to terms extraction tools.

Statistical methods are more computable, general, and scalable [12]. Then, they are easy to adapt from one source code to another.

2.3.3. Multi Strategy learning

Multi Strategy learning techniques leverage the strengths of the above techniques to extract a wide range of ontological knowledge from different types of data sources [11, 12]. Maeche and Staab [10] present for example, the use of clustering for concepts learning and association rules to learn relations between these concepts.

2.4. Ontology learning evaluation

Once the knowledge is extracted and the ontology is built, the evaluation will make it possible to judge if the ontology is good or not and conclude on the quality of the approach. The evaluation of ontologies is coined by several authors in the literature [6, 34–36]. Dellschaft and Staab [34] proposed to consider structural evaluation and functional evaluation and defined a scenario for ontology evaluation.

2.4.1. Ontology learning evaluation scenario

There are 2 scenario to evaluate ontologies [34]:

– **Quality Assurance During Ontology Engineering:** This will evaluate the choice of the correct corpus, the evaluation in the running application or, the evaluation depending on certain desirable criteria such as consistency, completeness, conciseness.;
– **Comparing ontology learning algorithms:** It consists of comparing ontology learning algorithms with each other. It can be used to improve existing learned algorithms or to find how changing the values of input parameters can change the results produce by the algorithm.

For example, one may compare the ontology obtained during the ontology learning process to a gold standard. The gold standard can be a corpus, a list of terms (validated by a domain expert) or an ontology representing an idealized outcome of the learning algorithm [34–36].

2.4.2. Structural evaluation of ontologies

The structural evaluation is related to the representation of an ontology as a graph. A gold standard can be used here and the learning algorithm will be considered to be good when the structure of learned ontology will have a high similarity with the structure of the gold standard [34].

2.4.3. Functional evaluation of ontologies

The functional evaluation of an ontology is related to its conceptualization [34]. For example, the quality assurance will evaluate if the ontology is consistent, complete, concise, and expandable [6, 34–36]. To do this, one may check if the target domain of the ontology is sufficiently modeled to fulfill the functional requirements and/or whether the ontology helps to improve the performance in the task for which it is designed [34]. Corpus-based approaches are used to check how far an ontology sufficiently covers a given domain. To measure the quality of the learning technique, functional evaluation consist in looking at the output (the learned ontology) and comparing it with the input (the content of the corpus). There are 2 ways to do this task [34–36]:

– Manual evaluation by human experts: Here, the learned ontology is presented to one or more do-

main experts who have to judge to what extend the knowledge extracted is correct.
- Compare the ontology to the gold standard to ensure that the ontology covers the content of the corpus.

3. A HMM approach for knowledge extraction from source code

Source code contains well-defined words in a language that everyone understands (case of the elements generally found on the user interface), some statements which has a particular lexicon specific to the programming language and to the programmer. For example, in Java programming language, the term "class" is used to define a class, the terms "if", "else", "switch", "case" are used to define the business rules (candidate terms to become the rules). Other term defined by the programmer such as "PatientTuberculeux" is used to define a name of class (candidate term to be concept); the term "examenATB" is used to define the relation (ObjectProperty) with cardinality (candidate term to become axiom) between the classes "PatientTuberculeux" and "Examen"; and the group of terms "int agePatient" is used to define a property (DataProperty) of the class "PatientTuberculeux". However, the applications generally manage several dozen or even hundreds of files (e.g. EPICAM has 7154 files) of different types (.txt, .java, .properties, .html, etc.) containing millions of lines of code (EPICAM contains 1326166 lines of code). In addition, the software tends to evolve over time (changing version, bugs correction) and related conceptual models or specification documents are not usually updated. It could be very costly to extract the knowledge from source code manually. Therefore, we propose to use Hidden Markov Models as an ontology learning approach for automatic knowledge extraction from source code.

3.1. Hidden Markov Models

Markov Chain is a random process having a finite set of states, and only the current state influence where it goes next [30]. HMMs are particular types of Markov Chain composed of a finite state automaton with edges between any pair of states that are labeled with transition probabilities. It also describes a 2-stage statistical process in which the behavior of the process at a given time t is only dependent on the immediate predecessor state. It is characterized by the probability between states $P(q_t|q_1, q_2, ..., q_{t-1}) = P(q_t|q_{t-1})$ and for every state at time t an output or observation o_t is generated. The associated probability distribution is only dependent on the current state q_t and not on any previous states or observations: $P(o_t|o_1, ..., o_{t-1}, q_1, ..., q_t) = P(o_t|q_t)$ [30, 31]. HMM is generally used for pattern recognition, automatic voice processing, automatic natural language processing, character recognition [30, 32], musical genres classification [37].

A first order HMM well described the source code which is sequentially typed by the programmer and the current word (corresponding to an assign hidden state) depends on the previous word. In this HMM, the observed symbol depends only on the current state [30–32]. Formula 1 presents the joint probability of a series of observations $O_{1:T}$ given a series of hidden state $Q_{1:T}$.

$$P(O_{1:T}, Q_{1:T}) = P(q_1)P(o_1|q_1)\prod_{t=2} P(q_t|q_{t-1})P(o_t|q_t) \quad (1)$$

Filtering, smoothing, prediction, and the most likely explanation are three usages of HMM. In the most likely explanation, the goal is to find the sequence of hidden states that best explain the sequence of observations (formula 3) [30–32] . The probability that a string X is emitted by an HMM M is calculated as the sum of all possible paths by:

$$P(X \mid M) = \sum_{q_1,\ldots,q_l} \prod_{k=1}^{l+1} P(q_{k-1} \to q_k)P(q_k \uparrow x_k) \quad (2)$$

Where q_0 and q_{l+1} are limited to q_I and q_N respectively and x_{l+1} is an end of word. The observable output of the system is the sequence of symbols emitted by the states, but the underlying state sequence itself is hidden.

Before using the model, its parameters (transition probability, emission probability and initial probability) must be calculated from data or, in the case where a model exist, by estimating them using Forward-Backward algorithm, Baum-Welch algorithm, or Expectation-maximization algorithm [30, 32].

It is common to search for a sequence of states $V(X \mid M)$ which has the greatest probability to produce an observation sequence [30–32]. For example, in automatic translation, one may want the most probable string sequence that corresponds to the string to be translated. In this case, instead of taking the sum of the probabilities, take the maximum (formula 3).

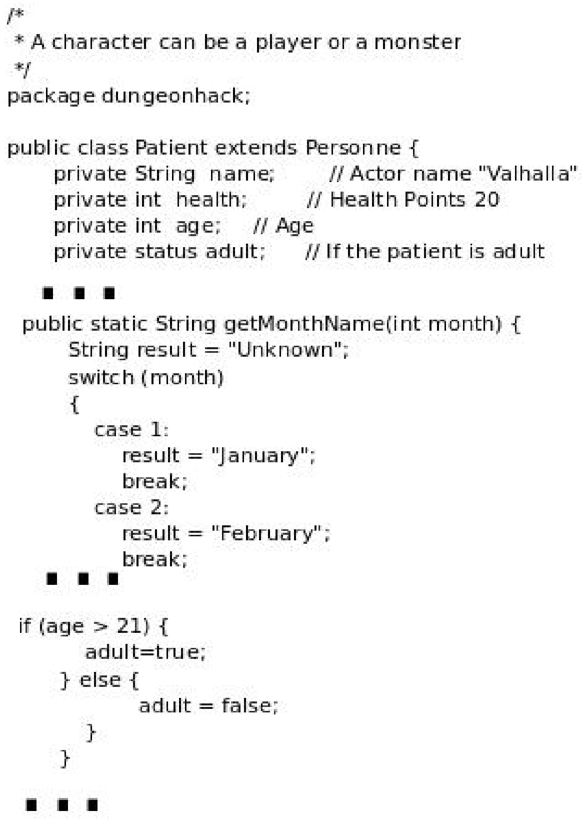
$$P(X \mid M) = argMax_{\,q_1 \ldots q_l \in Q^l} \prod_{k=1}^{l+1} P(q_{k-1} \rightarrow q_k) P(q_k \uparrow x_k) \tag{3}$$

The great popularity of this modeling technique results from its successful application in the field of automatic speech recognition. In this area of research, hidden Markov models have effectively replaced all competing approaches and constitute the dominant processing paradigm. This success is due to their superior ability to describe processes or signals evolving in time [30].

3.2. Source code versus HMM

The source code is any fully executable description of a software (in the form of texts as written by developers) designed for a specific domain such as e-health or e-epidemiology. 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 capture a set of knowledge of the domain. For example, EPICAM, an epidemiological surveillance platform[3] [4] allows health personnels to collect and share health information. Then, it captures knowledge of epidemiological surveillance of tuberculosis.

Source code is generally written according to good programming practices, including naming conventions [38] (e.g. in [39], Oracle presents good practices of Java programming). These practices tell programmers how to name variables, organize and present the source code. This organization can be used to model source code using HMMs. For example, from the source code of figure 1, we can say that at a time *t*, the programmer enters a word (e.g. "public" at the beginning of a Java source file). Thus, the key word "public" at time *t* condition the next word at time $t + d_t$ which in this case can be "class". We can say that *PRE* and *TARGET* are the hidden states and "public" and "class" are respectively their observations states. Source code can then be modeled using the HMM of figure 2.



```
/*
 * A character can be a player or a monster
 */
package dungeonhack;

public class Patient extends Personne {
     private String  name;       // Actor name "Valhalla"
     private int  health;       // Health Points 20
     private int  age;   // Age
     private status adult;    // If the patient is adult

   ■ ■ ■
  public static String getMonthName(int month) {
       String result = "Unknown";
       switch (month)
       {
          case 1:
             result = "January";
             break;
          case 2:
             result = "February";
             break;
    ■ ■ ■

 if (age > 21) {
        adult=true;
     } else {
              adult = false;
        }
     }

  ■ ■ ■
```

Fig. 1. Example of a JAVA source file

3.3. Source code description

Source code contains several types of files: files describing data, files processing data, user interface files and configuration files.

3.3.1. Files describing data

These files describe the data to be manipulated and some constraints on this data (e.g. data types). In Java EE for example, there are entities that will be transformed into tables in the database. These files often contain certain rules to verify the reliability of the data. Thus, from these files, we can retrieve concepts, properties, axioms and rules.

[3]www.epicam.cm
[4]http://github.com/UMMISCO/EPICAM

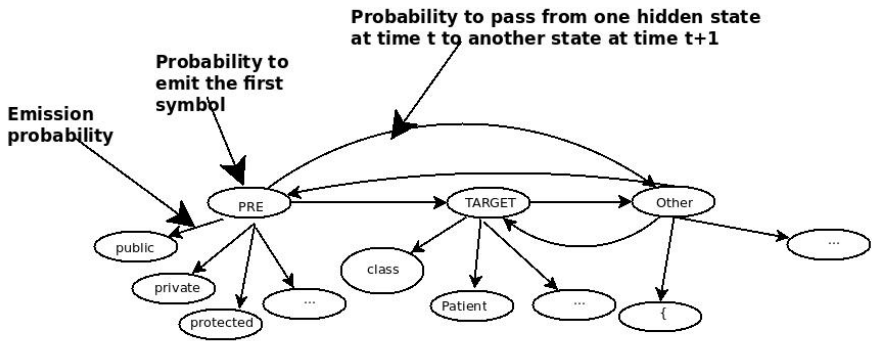

Probability to pass from one hidden state at time t to another state at time t+1
Probability to emit the first symbol
Emission probability
PRE
TARGET
Other
public
private
protected
...
class
Patient
...
{
...

Fig. 2. HMM example

3.3.2. Files containing data processing

Located between user interfaces files and data description files, this part of source code contains data processing, consisting in:

- **Control:** For example, restricting certain data to certain users (only the attending physician has the right to access to data), checking the validity of a field (checking whether the data entered in an "*age*" field is of integer type);
- **Calculation:** For example, converting a date of birth into an age, determining the date of the next appointment of a patient, calculating the body mass index of a patient based on its weight and height.

They are the algorithms implementing the business rules to be applied to the data. They are thus good candidates for axioms and rules extraction.

3.3.3. User interfaces files

The User interfaces are composed of files which describe the information that will be presented to users for data viewing or recording. Unlike the first three files types, these files contain the words of a human-readable vocabulary that can be found in a dictionary. User interfaces usually provide:

- Translations allowing navigation from one language to another, control for users to enter the correct data;
- An aid allowing users to know for example the role of a data entry field.

User Interfaces are therefore good candidates for concepts and their definitions, properties, axioms and rules extraction.

3.3.4. Configuration files

These files allow developers to specify certain information such as the type and path of a data source, different languages used by users, etc. For instance, from these files the languages labels (e.g. English, French, Spanish) for terms can be extracted.

The files we just presented generally contain comments that can be useful for knowledge extraction or ontology documentation. Knowledge extraction from user interfaces/web interfaces has already been addressed in [8, 14, 29], knowledge extraction from text has been presented in [1, 10, 13, 40]. In this article, we will focused on knowledge extraction from files describing data and their processing.

3.4. Knowledge extraction process

To extract knowledge from source code, we designed a method divided into five main steps: data collection, data preprocessing, entity labeling, formal language translation, and expert validation.

3.4.1. Data collection

Data collection step consists of the extraction of a dataset necessary for the next steps. In Java files, all class names, class attributes, class methods, statements containing the keywords "if", "else", "switch" are retained. Instructions for importing third-party libraries are ignored. To do this, we propose to define a regular expression allowing to extract the data that we need. This regular expression considers a source file as a sentence with a beginning and an end. The beginning of the sentence is marked with a set of alphabetic characters and the end is marked with an end character. Once the sentences are identified, they are preprocessed to identify relevant data.

3.4.2. Data preprocessing

The purpose of data preprocessing is to put data in a form compatible with the tools to be used in the next steps. 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 labeling. This is to determine the syntactic label of the words of a text [32]. In our case, it will be a matter of assigning a label to all the words of the source code and extracting the words marked as target word. This problem can be solved using HMM [31, 32]. In the following paragraphs, we will first present the HMM structure, then show how we learn the HMM on a dataset and finally, how to use it to extract the knowledge.

HMMs structure definition To define the structure of the HMMs, we manually studied the organization of the source code of Java language. Generally, data structures, attributes, conditions are surrounded by one or more specific 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:

- **PRE:** Corresponding to the preamble of the knowledge. This preamble is usually defined in advance;
- **TARGET:** The target, (i.e. the knowledge sought) may be preceded by one or more words belonging to the PRE set. The knowledge we are looking for are the names of "classes", the attributes, the classes methods, some relationships between "class". They are usually preceded by a metadata which describes them. For example, the metadata "class" allows to identify a class;
- **POST:** Any information that follows the knowledge sought. In some cases, POST is a punctuation character or a braces;
- **OTHER:** Any other word in the vocabulary that neither precedes nor follows the knowledge sought.

An example of HMM annotated with labels is given in figure 2. Concepts, properties, axioms, and rules are usually arranged differently in the source code. We propose to define two HMMs allowing to identify them: one to identify concepts, properties, axioms and another to identify rules.

Learning Model on Data There are several techniques to determine the parameters of a HMM: Statistical learning on data, Forward-Backward, Baum-Welch, Expectation-maximization algorithms [30, 32]. In this article, we used statistical learning on existing data. Then, once the model structure has been defined, the parameters of the transition and emission models are calculated from the training data. To do this, we assume that we have access to T source code files labeled f_t knowing that f_t is not just a sequence of words, but a sequence of words pairs with the word and its label as presented by the figure 2. To train the model, we assume that we can define the order in which the different words are entered by the programmer. We assume that before entering the first word, the programmer reflects on the label of that word and as a function of it, defines the label of the next word and so on. For example, before entering the word *public*, the programmer knows that its label is *PRE* and that the label of the next word is $TARGET$. Thus, the current word depends only on the current label, the following 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 4.

$$\begin{aligned} f_t &= [(w_1^t, e_1^t), ..., (w_d^t, e_d^t)], \\ words(f_t) &= [w_t^t, ..., w_d^t], \\ labels(f_t) &= [e_1^t, ..., e_d^t]. \end{aligned} \tag{4}$$

Where w_i and e_i are words and labels of f_i files respectively. In practice, w_i are words that can label the classes, attributes, relations and methods, or set of words that make up rules. When they are classes, they are composed of attributes and semantic relations with other classes. They are then labeled by e_i and, represent the hidden states of the HMM.

From the training data, we can extract statistics for each HMM on:

- The first label: $P(q_1)$ given by the formula 5. A priori probability that the first label is equal to $'a'$ is the number of times the first label in the source code is the word $'a'$ divided by the number of source code files.

$$P(Q_1 = a) = \frac{\sum_t freq(e_1^t = a, f_t)}{T} \tag{5}$$

- The relation between a word and its label $P(S_k \mid q_k)$ (formula 6). Conditional probability that the k^{th} word is $'w'$, knowing that the label is $'b'$ correspond to the number of times I saw the word $'w'$ associated with the label $'b'$ in the source code file f_t normalized with the fact that I saw the label $'b'$ associated with any other word in f_t source code. For example, "Patient" can be a concept, an attribute, but cannot be a rule.

Table 1
The initial vector: probability to have a state as the first label

f(PRE)	f(TARGET)	f(POST)	f(OTHER)

$$P(S_k = w \mid q_k = b) = \frac{\alpha + \sum_t freq((w,b), f_t)}{\beta + \sum_t freq(('*,b), f_t)} \quad (6)$$

To avoid zero probabilities for observations that do not occur in the training data, we added a smoothing terms (α and β).

- The relation between the adjacent syntactic label $P(q_k \mid q_{k+1})$ (formula 7). The probability that 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 source code of the training data divided by the number of times that $'b'$ is followed by any other label.

$$P(q_{k+1} = a \mid q_k = b) = \frac{\alpha + \sum_t freq(b,a), label(f_t)}{\beta + \sum_t freq(b,*'), label(f_t)} \quad (7)$$

To avoid zero probabilities for transitions that do not occur in the training data, we added a smoothing terms (α and β).

Let us consider the HMM of figure 2. Then, training corpus to identify concepts and attributes would be: [("public", PRE), ("class", TARGET), ("Patient", TARGET), ("extends", TARGET), ("ImogEntityImpl", TARGET), ("{", OTHER), (...), ("int", TARGET), ("age", TARGET), ...]. The table 2 presents the calculation of the frequencies that a state follows another state, table 3 presents the calculation of the frequencies that a state emits an observation and table 1 present the initial vector, which is the probability that the first label is PRE, TARGET, POST, and OTHER.

Knowledge extraction Once the model is defined, we can apply it to any Java source code. It will be to find from the files f_1, ..., f_n, a sequence of states q_1, ..., q_n that is plausible. For this, the formula 3 will be used to determine the most plausible string sequence. From this string, the hidden states will be identified and the targets (labeled $TARGET$) will be extracted. The Viterbi algorithm provides an efficient way of finding the most plausible sequence of hidden states [30–32]. Any source code can then be submitted to the HMM trained and a table similar to 3 containing the probability for the hidden states to emit a word from the source code is built.

Recoding variables Programmers usually use expressions made up of words from a specific lexicon, sometimes encoded with "ad hoc" expressions, requiring specific processing to assign a new name or a label understandable by human before using. These words are divided into words or groups of words according to the naming conventions of the programming language. For example, we can have "PatientTuberculeux" $\rightarrow$ "Patient tuberculeux", "agePatient" $\rightarrow$ "Age Patient", "listeExamens" $\rightarrow$ "liste examens", etc. Therefore, we separate different names extracted to find their real sense in human understandable language.

3.4.3. Entities labelling

The extraction of relevant terms has yielded data and metadata. These data and metadata will allow to identify to which ontology components they may belong to. For example, the code: "class Patient extends Person int age", submitted to a trained HMM to identify concepts and relations will identify three metadata ("class", "extends" and "int") that will then be used to identify two concepts (Patient and Person), one attribute of type integer and a hierarchical relation between "Patient" and "Person". 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 (ObjectProperty): 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 (subClassOf): 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'.
- Attribute (DatatypeProperty): 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, boolean, etc.), then, 'a' and 'b' are attributes of class 'A'.

3.4.4. Translation in a formal language

Once all relevant data is identified in the previous phase, they are automatically translated in a machine readable language. We use OWL language to represent concepts, properties and axioms and SWRL for rules.

Table 2
Example of a table presenting the frequency to move from one state to another

States	PRE	TARGET	POST	OTHER
PRE	f(PRE,PRE)	f(PRE,TARGET)	f(PRE,POST)	f(PRE,OTHER)
TARGET	f(TARGET,PRE)	f(TARGET,TARGET)	f(TARGET,POST)	f(TARGET,OTHER)
POST	f(POST,PRE)	f(POST,TARGET)	f(POST,POST)	f(POST,OTHER)
OTHER	f(OTHER,PRE)	f(OTHER,TARGET)	f(OTHER,POST)	f(OTHER,OTHER)

Table 3
Example of a table presenting the frequency for different states to emit an observation

	package	pac	;	public	class	patient	...
PRE	f(PRE,package)	f(PRE, pac)	f(PRE,;)	f(PRE,public)	f(PRE,class)	f(PRE,patient)	...
TARGET	f(TARGET,package)	f(TARGET, pac)	f(TARGET,;)	f(TARGET,class)	f(TARGET,patient)	...	
POST	f(POST,package)	f(POST, pac)	f(POST,;)	f(POST,public)	f(POST,class)	f(POST,patient)	...
OTHER	f(OTHER,package)	f(OTHER, pac)	f(OTHER,;)	f(OTHER,public)	f(OTHER,class)	f(OTHER,patient)	...

3.4.5. Validation by an expert

The method we have just presented does not aim at extracting a perfect knowledge model, but to help knowledge engineers to identify the knowledge artifacts from the source code. This help is particularly useful in our case because software has knowledge distributed in several files making millions of lines of source code. Thus, validation will make it possible to select the most important knowledge from the candidates and present them in a way that they could be easily exploitable by domain experts.

4. Experiment, results and evaluation

In this section, we will present the experiments conducted on EPICAM platform and GeoServer developed in Java. All our algorithms have been coded in Java[5] and, during the experiments, we extracted the candidate terms to be concepts, properties, axioms and rules. In the following paragraphs, we will first present EPICAM platform, the experiments of the approach on EPICAM source code, the results and evaluation of knowledge extracted from the source code of this platform and, finally, we will present the experiments conducted on GeoServer.

4.1. EPICAM platform

The EPICAM platform[6,7] is an Open Source platform for epidemiological surveillance of tuberculosis based on Imogene editor[8]. It was developed using the MDA (Model Driven Architecture) approach [15]. Thus, a meta-model was used to model tuberculosis surveillance. The applications have been generated, additional classes and some business constraints (form control, data access control) have been integrated manually in the source code. EPICAM helps health personnels to collect data in hospitals and these data are recorded in a postgresQL database. Once the data is in the database server, different users of different levels (Hospitals, Districts, Regions and Central) and different profiles (health professionals, district, regional and central officials) can view them (patient data, statistics) according to their rights. There is usually a need to inter-operate this information with other information to explain some phenomenon. For example, environmental data in conjunction with epidemiological data can be used to identify the source of an epidemic. In addition, adding new information into source code must be done with updating the conceptual model. For this to happen, we must put in place an ontology that models the domain knowledge and allows knowledge in conceptual model and source code to be coherent, different data sources to be integrated and the various stakeholders to communicate. Though EPICAM is developed in JAVA, we will exploit the structure of the JAVA source code [39] to extract knowledge in order to build this ontology.

[5]The whole source code is available on https://github.com/jiofidelus/knowExtractionSC

[6]www.epicam.cm

[7]http://github.com/UMMISCO/EPICAM

[8]https://github.com/medes-imps/imogene/wiki

4.2. *Knowledge extraction from EPICAM source code*

To extract the knowledge from EPICAM source code, we simply proceed step by step the method we presented in section 3.

4.2.1. *Data collection*

The source files of EPICAM platform are composed of statements, importing libraries and comments. Data collection involves removing the importing libraries and comments. To do this, we have defined a regular expression to identify them. Once identified, we wrote the code (see the listing 1) to delete them.

Listing 1: A part of source code for data collection

```
...
import java.util.regex.Pattern;
...
//Take a file and remove importing libraries and comments
private static String exp = "^import|^/\\*\\*|.\\*/|^\\s\\*
      |^\\s//";
public static String getDataFromFile(String fileName){
  String data = "";
  pattern = Pattern.compile(exp);
  FileReader fr;
  BufferedReader br;
  try {
    fr = new FileReader(fileName);
    br = new BufferedReader(fr);
    for(String line; (line=br.readLine())!=null;){
    matcher = pattern.matcher(line);
    if(!matcher.find()){
      data+=line;
    }
  }
} catch (IOException e) {...}
```

4.2.2. *Data preprocessing*

Data preprocessing consists of extracting the elements likely to be relevant from the source code and recoding them. To do this, the HMMs are defined manually, trained automatically on data, used to extract the knowledge, and the knowledge extracted is recoded if necessary. To extract knowledge from EPICAM, we distinguished two HMMs: a HMM for concepts, properties, and axioms identification and a HMM for rules identification.

1. **Definition of the HMM structure for concepts, properties and axioms** The HMM used to identify concepts, properties and axioms is defined by:

   (a) $PRE = \{public, private, protected, static, final\}$, the set of words that precedes TARGET;

   (b) $TARGET = \{package, class, interface, extends, implements, abstract, enum, w_i\}$, $\forall i, w_{i-1} \in PRE \mid\mid w_{i-2} \in PRE \wedge w_{i-1} \in PRE$, the set of all words that we are seeking;

   (c) $OTHER = \{";", "", "", w_i\}, w_i \notin PRE \wedge w_i \notin TARGET$, the set of all other words.

   Each HMM state emitted a term corresponding to a word from the source code. We have seen that the observation emitted by the *PRE* set can be enumerated. However, the observation of *TARGET* and *OTHER* sets cannot be enumerated because they depend on the programmer. Then, we have designated by *data* all the observation emitted by by *TARGET* and by *other* all the observation emitted by *OTHER*. We thus obtained the HMM presented by the initial vector (table 4) the transition model (table 5), and the observation model (table 6).

2. **Defining the HMM structure for rules** Rules can be contained in conditions. Then, we will exploit the structure of source code to extract the rules. For example, the portion of code (if (agePatient> 21) {Patient = Adult}) is a rule determining whether a patient is an adult or not. It must therefore be extracted.
   The HMM to identify the rules is composed of:

   (a) $PRE = \{"\}", ";", "\{"\}$, the set of words that precedes one or many TARGET;

   (b) $TARGET = \{if, else, switch, w_i\} \mid \exists k, r \in N \mid w_{i-k} \in PRE \wedge wi + r \in POST$: the set of all words that follow PRE and precedes POST;

   (c) $POST = \{"\}"\}$, the end of the condition;

   (d) $OTHER = \{w_i\} \mid w_i \notin PRE, TARGET, POST$: the set of all other words.

   Unlike the HMM of concepts, we can identify the beginning and the end of a condition represented here by the sets *PRE* and *POST* respectively. Note that all the observation emitted by *TARGET* and *OTHER* sets cannot be fully enumerated. Then, we have designated by *data* all the observation emitted by *TARGET*, and by *other* all the observation emitted by *OTHER*. We then obtained the HMM presented by the initial vector (table 7), the transition model (table 8) and the observation model (table 9).

3. **Training the HMMs** There are several methods to determine the parameters of a HMM: statistical learning on data, Baum-Welch algorithm, Expectation-maximization algorithm, etc. [30, 32]. In the training step, since it is possible to retrieve some source code on Internet, we automatically assigned parameters to the HMMs

(using statistical learning on data) by defining an algorithm which, based on some source code downloaded on Internet, constructs the transition model between hidden states and observation model between hidden states and observation states. The listing 2 is a part of the source code that permit to calculate the PRE vector which shows the probability to go from the PRE state to another state, and the listing 3 presents a part of the source code that permit to calculate the probability that the *PRE* state emit some observation states. A set of JAVA source codes (composed of 59 files and 2663 statements) were downloaded from github[9] and from these source codes, we trained the HMMs. Tables 4, 5, 6, 7, 8, 9 present the models obtained after the training step.

Listing 2: A part of source code that permit to calculate the PRE transition vector: defined the probability to move from the PRE state to the others states

```
public static void preTransition(List<String>
      listSouceCode, HMMConcept hmmConcept) {
//Initialization by the smothing term
 double nbPre = 4, double nbPrePre = 1, double
      nbPreTarget = 1, double nbPreOther = 1;
 List<String> PRE = hmmConcept.getPreObservation();
 String[] tmp;
 for (String doc : listSouceCode) {
  tmp=doc.split(" ");
  for (int i = 0; i < tmp.length; i++) {
   //Number of PRE in the document
   if(Helper.belongs2Array(tmp[i], PRE)) {
    nbPre++;
   }
  //Number of times that a PRE is followed by another
       PRE
  if(i<tmp.length &&
   Helper.belongs2Array(tmp[i], PRE) &&
   Helper.belongs2Array(tmp[i+1], PRE)) {
   nbPrePre++;
 }
  //Number of times that a PRE is followed by a TARGET
  if(i<tmp.length &&
   Helper.belongs2Array(tmp[i], PRE) &&
   !Helper.belongs2Array(tmp[i+1], PRE)) {
   nbPreTarget++;
   }
  }
 }
 Transition prePre = hmmConcept.getPrePreTransition();
 Transition preTarget =
      hmmConcept.getPreTargetTransition();
 Transition preOther =
      hmmConcept.getPreOtherTransition();
 prePre.setTransitionValue(nbPrePre/nbPre);
 preTarget.setTransitionValue(nbPreTarget/nbPre);
 preOther.setTransitionValue(nbPreOther/nbPre);
 hmmConcept.setPrePreTransition(prePre);
 hmmConcept.setPreTargetTransition(preTarget);
 hmmConcept.setPreOtherTransition(preOther);
}
```

[9]https://github.com/mafudge/LearnJava

Listing 3: A part of source code that permit to calculate the PRE observation vector: defined the probability to emit observations

```
public static double[] getObservPre(List<String>
      listDocs){
//Initialization by the smothing term
  double nbPublic=1, nbPrivate=1, nbProtected=1,
       nbStatic=1, nbFinal=1, nbPre=1;
  double publicPre=1, privatePre=1, protectedPre=1,
       staticPre=1, finalPre=1;
  String[] tmp;
  for (String doc : listDocs) {
    tmp = helper.removeSpaces(doc.split(" "));
    for (int i = 0; i < tmp.length; i++) {
    //Counting the number of times the PRE state emit
         the observation: public, private, etc.
    if(tmp[i].trim().equals("public")) nbPublic++;
    if(tmp[i].trim().equals("private")) nbPrivate++;
    if(tmp[i].trim().equals("protected"))
         nbProtected++;
    if(tmp[i].trim().equals("static")) nbStatic++;
    if(tmp[i].trim().equals("final")) nbFinal++;
    //The number of times we have PRE in a document
    if(helper.belongs2Array(tmp[i], PRE))
      nbPre++;
    }
  }
  double[] preObservMod = {nbPublic/nbPre,
       nbPrivate/nbPre, nbProtected/nbPre,
       nbStatic/nbPre, nbFinal/nbPre};
  return preObservMod;
}
```

For the initial probability, we consider that all the times, after the data collection step, we have a TARGET state that is the package declaration at the beginning of all the files.

Table 4
The initial vector for HMM concepts, properties and axioms

PRE	TARGET	POST	OTHER
0	1	0	0

Table 5
Transition model for HMM concepts, properties and axioms

	PRE	TARGET	OTHER
PRE	0.1598	0.8376	0.0026
TARGET	0.0013	0.7561	0.2426
OTHER	0.0667	0.0008	0.9325

Table 6
Observation model for HMM concepts, properties and axioms

	public	private	protected	static	final
PRE	0.6692	0.1589	0.0026	0.1026	0.0667
TARGET	0	0	0	0	0
OTHER	0	0	0	0	0

	;	{	}	other	package
PRE	0	0	0	0	0
TARGET	0	0	0	0	0.0684
OTHER	0.2716	0.1332	0.1186	0.4766	0

	class	extends	interface	implements
PRE	0	0	0	0
TARGET	0.0112	0.0076	0.0012	0.0087
OTHER	0	0	0	0

	abstract	enum	data
PRE	0	0	0
TARGET	0.0012	0.0025	0.8992
OTHER	0	0	0

Table 7
The initial vector for HMM rule

PRE	TARGET	POST	OTHER
0	0	0	1

Table 8
Transition model for HMM rule

	PRE	TARGET	POST	OTHER
PRE	0.0667	0.7999	0.0667	0.0667
TARGET	0.0010	0.9321	0.0659	0.0010
POST	0.0172	0.0172	0.0172	0.9484
OTHER	0.0072	0.0001	0.0001	0.9926

Table 9
Observation model for HMM rule

	{	}	;	if	else
PRE	0.8462	0.0769	0.0769	0	0
TARGET	0	0	0	0.0185	0.0031
POST	0	1	0	0	0
OTHER	0	0	0	0	0

	switch	data	other
PRE	0	0	0
TARGET	0.0010	0.9774	0
POST	0	0	0
OTHER	0	0	1

4. **Knowledge extraction** Once the HMMs are built, we can apply them to the source code of any Java applications in order to extract knowledge. Then, we will use them for knowledge extraction from the source code of EPICAM platform by calculating-giving this source code the most likely state sequence (formula 3) that produce it. To do this, we have implemented the Viterbi algorithm [30–32] in Java. In fact, we have exploited the structure of the HMM in the context of dynamic programming. It consists to break down the calculations into intermediate calculations which we structured in a table (see table 10). Every element of the table is being calculated using the previous ones. The different steps are as follow:

– Step 1: The definition of the elements of the table that we called $\alpha(i,t)$:
  $\alpha(i,t) = P(W_{1:t} = w_{1:t}, Q_{1:t-1} = q_{1:t-1}, Q_t = i)$. Where $W_{1:t}$ is the set of observations from time 1 to time t; $Q_{1:t-1}$ the set of optimal values stored in the table from time 1 to time $t-1$.
– Step 2: Decomposition into intermediate calculations:
  $\alpha(i,t+1) = Max_j P(W_{1:t+1} = w_{1:t+1}, Q_{1:t} = q_{1:t}, Q_t = j, Q_{t+1} = i)$
  $= Max_j P(W_{t+1} = w_{t+1}|Q_{t+1} = i)P(Q_{t+1} = i|Q_t = j)P(W_{1:t} = w_{1:t}, Q_{t-1} = Q_{t-1}, Q_t = j)$
  $= P(W_{t+1} = w_{t+1}|Q_{t+1} = i)Max_j P(Q_{t+1} = i|Q_t = j)\alpha(j,t)$
– Step 3: Fill the table given the HMM and the source code as input:
  * Calculate the elements of the first column (composed of three frames) of the table:
    $\alpha(i,1) = P(W_1 = package|Q_1 = i)P(Q_1 = i)$
    $\alpha(PRE,1) = P(W_1 = package|Q_1 = PRE)P(Q_1 = PRE) = 0$
    $\alpha(TARGET,1) = P(W_1 = package|Q_1 = TARGET)P(Q_1 = TARGET) = 1$
    $\alpha(OTHER,1) = P(W_1 = package|Q_1 = OTHER)P(Q_1 = OTHER) = 0$
  * Calculate the elements of others frames given the elements of the previous frames. For example, for the fourth column, calculate:
    $\alpha(i,4) = P(W_4 = public|Q_4 = i)Max_j P(Q_4 = i|Q_3 = j)\alpha(j,4)$
    $\alpha(PRE,4) = P(W_4 = public|Q_4 = PRE)Max_j P(Q_4 = i|Q_3 = j)\alpha(j,3)$
    $\alpha(TARGET,4) = P(W_4 = public|Q_4 =$

$i)Max_j P(Q_4 = i|Q_3 = j)\alpha(j,3)$
$\alpha(OTHER,4) = P(W_4 = public|Q_4 = i)Max_j P(Q_4 = i|Q_3 = j)\alpha(j,4)$

We obtain the dynamic programming table given by the table 10. Once the table is built, we find the Viterbi path by getting the frame that has the most greatest probability in the last column and given this frame, find all the frames that was used to build it. Listing 4 presents a part of source code used to build the dynamic programming table and the listing 5 presents a part of the source code used to extract the most likely states sequence based on the source code. Once the Viterbi path is identified, all the elements labeled *TARGET* are extracted.

Listing 4: The α table

```
public  static List<Column>
      fillAlphaStartTable4Concepts (HMMConcept
      hmmConcept, String sourceFile) {
        List<Column> alphaTable = new ArrayList<>();
    String[]tmp=sourceFile.split(" ");
        //Table initialization
    Column column = new Column();
    //The creation of the frames of the first column.
          To avoid infinitesimal numbers, we multiply
          by 1.0E300 at the beginning of the algorithm
    Frame framePRE0 = new Frame(tmp[0], "PRE", 0);
    Frame frameTARGET0 = new Frame(tmp[0], "TARGET",
          1.0E300);
    Frame frameOTHER0 = new Frame(tmp[0], "OTHER", 0);
    //Add the first frames to the column
    column.setPreFrame(framePRE0);
    column.setTargetFrame(frameTARGET0);
    column.setOtherFrame(frameOTHER0);
    //Add the first column to the table
    alphaTable.add(column);
        ...
        for (int i = 1; i <tmp.length; i++) {
        ...
        //Get the last element of the list
        column = alphaTable.get(alphaTable.size()-1);
        //Return the column of the last index with
              the state label PRE. e.g. alpha(PRE,
              2), alpha(TARGET, 2)
        framePRE = column.getPreFrame();
        frameTARGET = column.getTargetFrame();
        frameOTHER = column.getOtherFrame();
        //Calculate the PRE state
        frameUsed = getMaxFramePre (framePRE,
              frameTARGET, frameOTHER,
              prePreTransitionValue,
              targetPreTransitionValue,
              otherPreTransitonValue);
        //The value of the probability that is
              calculated
        calculPre = Math.max (Math.max
              (prePreTransitionValue *
              framePRE.getProbabilityValue(),
              targetPreTransitionValue *
              frameTARGET.getProbabilityValue()),
              otherPreTransitonValue *
              frameOTHER.getProbabilityValue());
        //Get the observation probability
        if(Helper.belongs2Array(tmp[i],
              hmmConcept.getPreObservation())) {
        preEmission = preEmissionProbability(tmp[i],
              hmmConcept);
        }
        //Create the new frame for PRE state and add
              to the colum (argMax \multiply
              observation probability)
        frameTMP = new Frame (tmp[i], "PRE",
              calculPre * preEmission, frameUsed);
        columnTMP.setPreFrame (frameTMP);
        //Calculation for the TARGET and OTHER state
        ...
        alphaTable.add(columnTMP);
        }
        return alphaTable;
}
```

Listing 5: Extraction of words labelled TARGET

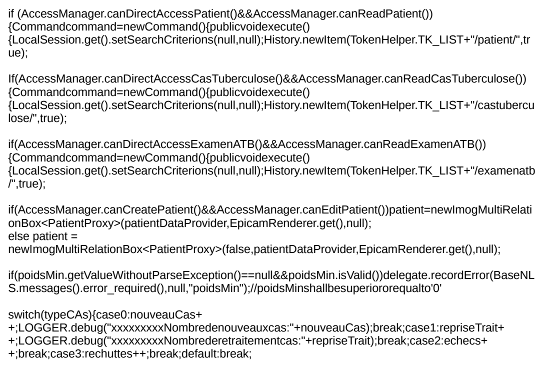
```
...
public static String knowledgeExtraction(List<Column>
      alphaTable) {
...
//Get the last colum of the alphaTable
  Column lasColumn =
        alphaTable.get(alphaTable.size()-1);
//Get the last frame which has the greatest
      probability
  frame = mostGreatestFrameProba(lasColumn);
//Get the frames that permit to have the above frame
  List<Frame> mostLikelyFrames = new ArrayList<>();
    mostLikelyFrames.add(frame);
//Browse the frame list to get the most likely frame
  for (int i = alphaTableSize; i > 0; i--) {
    frame = frame.getFrameBuilder();
    mostLikelyFrames.add(frame);
  }
  //Extract the knowledge from the above frame which
        hidden state is TARGET
  for (int i = mostLikelyFrames.size()-1; i>=0; i--) {
    if(mostLikelyFrames.get(i).getStateLabel().equals
          ("TARGET")&&
      StringUtils.indexOfAny
            (mostLikelyFrames.get(i).getObservedLabel(),
            falsePositive)==-1) {
      nbTarget++;
      label =
            mostLikelyFrames.get(i).getObservedLabel();
      mostLikelyExplanation+=label+"\n";
    }
    if(mostLikelyFrames.get(i).getStateLabel().equals
          ("TARGET")&&
      StringUtils.indexOfAny
            (mostLikelyFrames.get(i).getObservedLabel(),
      falsePositive)!=-1) nbFalsePositive++;
    }
  return mostLikelyExplanation;
}
```

The extraction of candidate terms from the above mentioned models gave a set of terms (figures 3 and 4), but also false positives (table 11 presents the statistics). The false positives consist of the set of terms that belongs to the PRE, POST or OTHER sets that normally should not be extracted as observations of TARGET. We have identified and deleted them automatically during the term extraction (see listing 4.2). It can be noted that the data extracted also contains metadata (e.g. "class", "if" or "boolean") which will be very useful in the entity identification phase.

Table 10

The dynamic programming table (α table) built using EPICAM source code

	package	**org.epicam**	**;**	**public**	**...**	**}**
PRE	0	$\alpha(PRE,2)$	$\alpha(PRE,3)$	$\alpha(PRE,4)$	...	$\alpha(PRE,t)$
TARGET	1	$\alpha(TARGET,2)$	$\alpha(TARGET,3)$	$\alpha(TARGET,4)$	...	$\alpha(TARGET,t)$
OTHER	0	$\alpha(OTHER,2)$	$\alpha(OTHER,3)$	$\alpha(OTHER,4)$	...	$\alpha(OTHER,t)$



```
if (AccessManager.canDirectAccessPatient()&&AccessManager.canReadPatient())
{Commandcommand=newCommand(){publicvoidexecute()
{LocalSession.get().setSearchCriterions(null,null);History.newItem(TokenHelper.TK_LIST+"/patient/",tr
ue);

If(AccessManager.canDirectAccessCasTuberculose()&&AccessManager.canReadCasTuberculose())
{Commandcommand=newCommand(){publicvoidexecute()
{LocalSession.get().setSearchCriterions(null,null);History.newItem(TokenHelper.TK_LIST+"/castubercu
lose/",true);

if(AccessManager.canDirectAccessExamenATB()&&AccessManager.canReadExamenATB())
{Commandcommand=newCommand(){publicvoidexecute()
{LocalSession.get().setSearchCriterions(null,null);History.newItem(TokenHelper.TK_LIST+"/examenatb
/",true);

if(AccessManager.canCreatePatient()&&AccessManager.canEditPatient())patient=newImogMultiRelati
onBox<PatientProxy>(patientDataProvider,EpicamRenderer.get(),null);
else patient =
newImogMultiRelationBox<PatientProxy>(false,patientDataProvider,EpicamRenderer.get(),null);

if(poidsMin.getValueWithoutParseException()==null&&poidsMin.isValid())delegate.recordError(BaseNL
S.messages().error_required(),null,"poidsMin");//poidsMinshallbesuperiororequalto'0'

switch(typeCAs){case0:nouveauCas+
+;LOGGER.debug("xxxxxxxxxNombredenouveauxcas:"+nouveauCas);break;case1:repriseTrait+
+;LOGGER.debug("xxxxxxxxxNombrederetraitementcas:"+repriseTrait);break;case2:echecs+
+;break;case3:rechuttes++;break;default:break;
```

Fig. 3. Screen capture of some terms extracted for rules identification

5. **Recoding terms and rules** To recode the terms extracted, we used JAVA naming convention and ontoEPICAM (an ontology of epidemiological surveillance developed with the help of domain expert - see paragraph 4.4.1 for more details). All the terms was browsed, comparing them to the terms of ontoEPICAM. When we noticed a term close to a term of ontoEPICAM, we verify if by removing the keywords of the languages we will have the same name. If yes, the keywords are removed and the term is retained. For example, if we consider the term *CasTuberculoseEditorWorkflow* that was extracted from the source code, then, *CasTuberculose* is a term that we have in ontoEPICAM, the terms *Editor* and *Workflow* are keywords of Google Web Toolkit - the technology used to built the EPICAM platform. Then, the terms *Editor* and *Workflow* are removed and the term *CasTuberculose* is retained. The terms that were not found in ontoEPICAM (e.g., the term *AccessPolicyFactory*) were separated using JAVA naming convention (in our case, we obtain *AccessPolicy Factory*) and the keyword(s) were removed. The term obtained was submitted to domain expert for validation. Rules have also been validated by domain experts and many of them were improved and re-injected into the source code. After the recoding, we move to the next step which is the translation into formal language.

4.2.3. Extraction of entities and translation into a formal language

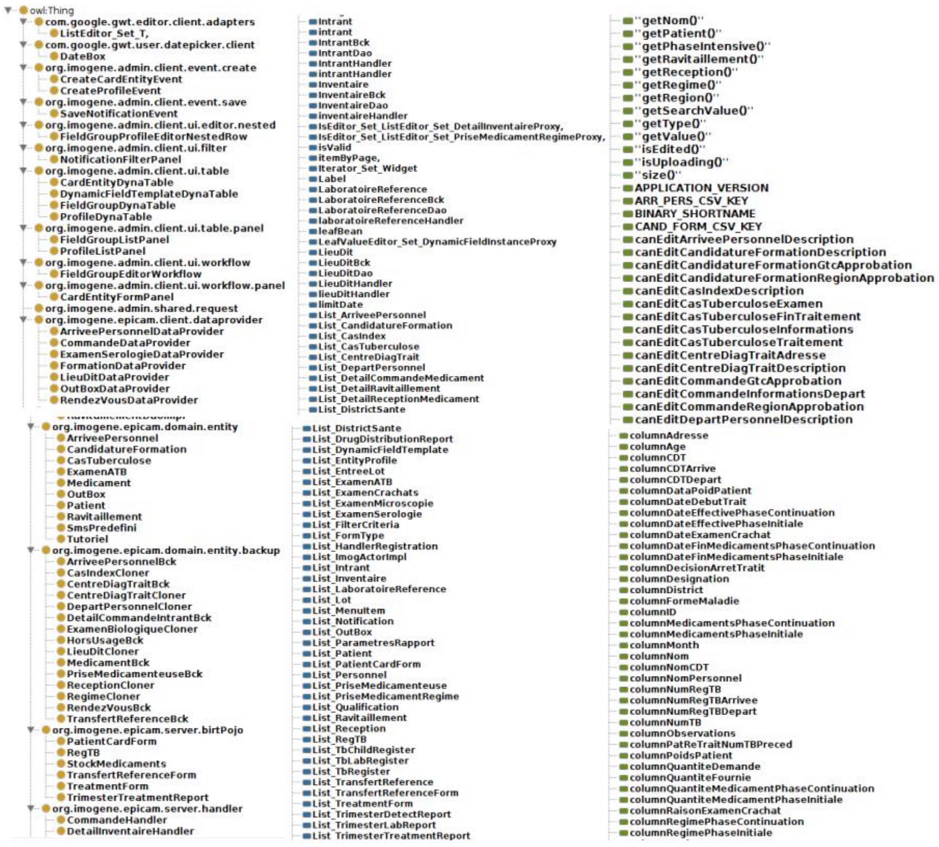
After the data preprocessing phase, we obtained files containing only the metadata (e.g "package", "class", "extends", "if", "switch") and data (e.g "patientManagement.Patient", "Patient" or "serology"). A simple algorithm makes it possible to browse these data in order to identify the knowledge that may be useful. Metadata allowed the identification of the candidates terms as concepts, properties and axioms. For example, if we have extracted "package minHealth.Region.District.hospitals.patientRecord ... class Patient extends Person ... int age ... List<Exam> listExam", then, a simple algorithm can be used to identify every element:

- **"package minHealth.Region.District.hospitals. patientRecord:"** This is used to identify the class hierarchy;
- **"class Patient extends Person":** This expression means that "Patient" and "Person" are candidate terms that will become concepts and there is a hierarchical relation between concepts labeled as "Patient" and "Person";
- **"int age; List <Exam> listExam":** This expression means that "age" and "listExam" are properties of the concept "Patient";
- **"List<Exam> listExamen":** This allows to define an axiom because it can be translated by: "a patient has one or more exams".

After the identification of entities, we wrote a code that automatically translated them to an OWL ontology. The listing 6 presents a method which transform a term to an OWL class, and listing 7 presents a method which based on two terms, creates an OWL object property domain.

Listing 6: Method which transform a term to an OWL class

```
public String genClassDeclarations(final String entity){
```



```
owl:Thing
  com.google.gwt.editor.client.adapters
    ListEditor_Set_T,
  com.google.gwt.user.datepicker.client
    DateBox
  org.imogene.admin.client.event.create
    CreateCardEntityEvent
    CreateProfileEvent
  org.imogene.admin.client.event.save
    SaveNotificationEvent
  org.imogene.admin.client.ui.editor.nested
    FieldGroupProfileEditorNestedRow
  org.imogene.admin.client.ui.filter
    NotificationFilterPanel
  org.imogene.admin.client.ui.table
    CardEntityDynaTable
    DynamicFieldTemplateDynaTable
    FieldGroupDynaTable
    ProfileDynaTable
  org.imogene.admin.client.ui.table.panel
    FieldGroupListPanel
    ProfileListPanel
  org.imogene.admin.client.ui.workflow
    FieldGroupEditorWorkflow
  org.imogene.admin.client.ui.workflow.panel
    CardEntityFormPanel
  org.imogene.admin.shared.request
  org.imogene.epicam.client.dataprovider
    ArriveePersonnelDataProvider
    CommandeDataProvider
    ExamenSerologieDataProvider
    FormationDataProvider
    LieuDitDataProvider
    OutBoxDataProvider
    RendezVousDataProvider

  org.imogene.epicam.domain.entity
    ArriveePersonnel
    CandidatureFormation
    CasTuberculose
    ExamenATB
    Medicament
    OutBox
    Patient
    Ravitaillement
    SmsPredefini
    Tutoriel
  org.imogene.epicam.domain.entity.backup
    ArriveePersonnelBck
    CasIndexCloner
    CentreDiagTraitBck
    CentreDiagTraitCloner
    DepartPersonnelCloner
    DetailCommandeIntrantBck
    ExamenBiologiqueCloner
    HorsUsageBck
    LieuDitCloner
    MedicamentBck
    PriseMedicamenteuseBck
    ReceptionCloner
    RegimeCloner
    RendezVousBck
    TransfertReferenceBck
  org.imogene.epicam.server.birtPojo
    PatientCardForm
    RegTB
    StockMedicaments
    TransfertReferenceForm
    TreatmentForm
    TrimesterTreatmentReport
  org.imogene.epicam.server.handler
    CommandeHandler
    DetailInventaireHandler
```

```
Intrant
intrant
IntrantBck
IntrantDao
IntrantHandler
intrantHandler
Inventaire
InventaireBck
InventaireDao
inventaireHandler
IsEditor_Set_ListEditor_Set_DetailInventaireProxy,
IsEditor_Set_ListEditor_Set_PriseMedicamentRegimeProxy,
isValid
itemByPage,
Iterator_Set_Widget
Label
LaboratoireReference
LaboratoireReferenceBck
LaboratoireReferenceDao
laboratoireReferenceHandler
leafBean
LeafValueEditor_Set_DynamicFieldInstanceProxy
LieuDit
LieuDitBck
LieuDitDao
LieuDitHandler
lieuDitHandler
limitDate
List_ArriveePersonnel
List_CandidatureFormation
List_CasIndex
List_CasTuberculose
List_CentreDiagTrait
List_DepartPersonnel
List_DetailCommandeMedicament
List_DetailRavitaillement
List_DetailReceptionMedicament
List_DistrictSante

List_DistrictSante
List_DrugDistributionReport
List_DynamicFieldTemplate
List_EntityProfile
List_EntreeLot
List_ExamenATB
List_ExamenCrachats
List_ExamenMicroscopie
List_ExamenSerologie
List_FilterCriteria
List_FormType
List_HandlerRegistration
List_ImogActorImpl
List_Intrant
List_Inventaire
List_LaboratoireReference
List_Lot
List_MenuItem
List_Notification
List_OutBox
List_ParametresRapport
List_Patient
List_PatientCardForm
List_Personnel
List_PriseMedicamenteuse
List_PriseMedicamentRegime
List_Qualification
List_Ravitaillement
List_Reception
List_RegTB
List_TbChildRegister
List_TbLabRegister
List_TbRegister
List_TransfertReference
List_TransfertReferenceForm
List_TreatmentForm
List_TrimesterDetectReport
List_TrimesterLabReport
List_TrimesterTreatmentReport
```

```
''getNom()''
''getPatient()''
''getPhaseIntensive()''
''getRavitaillement()''
''getReception()''
''getRegime()''
''getRegion()''
''getSearchValue()''
''getType()''
''getValue()''
''isEdited()''
''isUploading()''
''size()''
APPLICATION_VERSION
ARR_PERS_CSV_KEY
BINARY_SHORTNAME
CAND_FORM_CSV_KEY
canEditArriveePersonnelDescription
canEditCandidatureFormationDescription
canEditCandidatureFormationGtcApprobation
canEditCandidatureFormationRegionApprobation
canEditCasIndexDescription
canEditCasTuberculoseExamen
canEditCasTuberculoseFinTraitement
canEditCasTuberculoseInformations
canEditCasTuberculoseTraitement
canEditCentreDiagTraitAdresse
canEditCentreDiagTraitDescription
canEditCommandeGtcApprobation
canEditCommandeInformationsDepart
canEditCommandeRegionApprobation
canEditDepartPersonnelDescription

columnAdresse
columnAge
columnCDT
columnCDTArrive
columnCDTDepart
columnDataPoidPatient
columnDateDebutTrait
columnDateEffectivePhaseContinuation
columnDateEffectivePhaseInitiale
columnDateExamenCrachat
columnDateFinMedicamentsPhaseContinuation
columnDateFinMedicamentsPhaseInitiale
columnDecisionArretTratit
columnDesignation
columnDistrict
columnFormeMaladie
columnID
columnMedicamentsPhaseContinuation
columnMedicamentsPhaseInitiale
columnMonth
columnNom
columnNomCDT
columnNomPersonnel
columnNumRegTB
columnNumRegTBArrivee
columnNumRegTBDepart
columnNumTB
columnObservations
columnPatReTraitNumTBPreced
columnPoidsPatient
columnQuantiteDemande
columnQuantiteFournie
columnQuantiteMedicamentPhaseContinuation
columnQuantiteMedicamentPhaseInitiale
columnRaisonExamenCrachat
columnRegimePhaseContinuation
columnRegimePhaseInitiale
```

Fig. 4. Screen capture of some terms extracted for concepts, properties and axioms identification

```
  String class2Add = "";
  class2Add = (class2Add + "\n<Declaration>");
  class2Add = (((((class2Add + "\n\t <Class IRI=") + "\"#")
       + entity) + "\"") + "/>");
  class2Add = (class2Add + "\n</Declaration>");
  return class2Add;
}
```

Listing 7: Method which create an object property from two terms

```
//Generate all object properties domain
public String genObjectPropertiesDomain(String
     relationField, String entity){
  String objectProperty="";
  objectProperty = objectProperty +
       "\n<ObjectPropertyDomain>";
  objectProperty = objectProperty+"\n\t <ObjectProperty
       IRI="+"\"#"+relationField+"\"/>";
  objectProperty = objectProperty+"\n\t <Class
       IRI="+"\"#"+entity+"\"/>";
  objectProperty = objectProperty +
       "\n</ObjectPropertyDomain>";
  return objectProperty;
}
```

Rules were also translated into formal language (we have used SWRL language). An example of rule specifying the rights of a doctor on patient data is given by:

doctorsRule = "Personnel (?pers) ∧ personnel_login (?pers, login) ∧ personnel_passwd (?pers, passwd) ∧ Patient (?p) ∧ RendezVous (?rdv) ∧ hasRDV (?rdv, ?p) ∧ patient_nom (?p, ?nom) ∧ patient_age (?p, ?age) ∧ patient_sexe (?p, ?sexe) ∧ patient_telephoneUn (?p, ?telephone) ∧ rendezVous_dat eRendezVous (?rdv, ?datardv) ∧ rendezVous_honore (?rdv, ?honore) ∧ rendezVous_honore (?rdv, Non) → sqwrl:select (?nom, ?age, ?sexe, ?telephone, ?datardv, ?honore)";

4.3. Results

From EPICAM source code, we have extracted 60363 candidates terms/group of terms to be concepts, properties and axioms (the group of terms for axioms identification describe the cardinalities between classes). From these terms, we found 4796 false positives (precision=92.05%). We also found that the number of TARGET in the source code is 76934-the number of false negative is 21367 (recall=72.22%). Table 11 presents the statistics of candidates terms/group of terms that were extracted. We have identified automatically the false positives in the terms extracted and we deleted them. After the deletion, we obtain different types of terms/group of terms:

- **Irrelevant terms/group of terms:** These are utilities classes and temporary variables. Utilities classes are classes that the programmer defines to perform certain operations (these classes usually contain constants and methods). The names of these classes are usually not related to the domain. Temporary variables (e.g., the variable used in a loop) are used temporary in the source code and are not related to the domain.
- **Relevant terms/group of terms:** These are candidates terms found. These terms are composed of synonyms (terms of identical meaning) and redundancies (terms that come up several times). A simple algorithm enable to remove redundancies terms automatically.

We have also extracted 18816 candidates conditions to be rules and 1182 false positives (precision=94.09%). The false positives were mainly the terms labeled OTHER that were extracted. We have also counted the conditions that the EPICAM source code normally contains (using a regular expression) and we found that it contains 21961 conditions (recall=85.68%). From these conditions, we found:

- **Irrelevant conditions:** These are conditions that are not really important. For example, testing whether a temporary variable is positive or is equal to certain value. These conditions were the most numerous;
- **Relevant conditions:** Conditions corresponding to a business rule (e.g., testing if a user has the right to access to certain data).

Once translated into OWL and SWRL, the knowledge obtained was opened in Protege to facilitate their visualization and evaluation by domain experts.

Table 11
Statistics on terms/group of terms extracted

Candidates terms	Relevant	Irrelevant
Concepts	1840 (72.87%)	685 (27.13%)
Properties	38355 (81.44%)	38741 (18.59%)
Axioms	3397 (83.22%)	685 (16.78%)
Rules	1484 (07,89%)	17332 (92.11%)

4.4. Evaluation

The evaluation permit to judge if the knowledge extracted is good or not. Because these knowledge can be used to build or enrich a domain ontology, we will use the ontology evaluation techniques to evaluate it. Therefore, we have defined a gold standard for the structural and the functional evaluation (see section 2.4).

4.4.1. The gold standard

Before the proposition of the HMM-based approach for knowledge extraction from source code, we have built a basic ontology for the epidemiological surveillance of tuberculosis (ontoEPICAM) from the database schema and meta-model of the EPICAM platform. Figure 5 presents a screen capture of some concepts and properties. We have in this ontology 97 concepts, 159 properties and 68 axioms. This ontology has been validated by domain experts. In the next paragraphs, we will consider this ontology as the gold standard for the evaluation of the relevance of the knowledge extracted by our HMMs.

4.4.2. Functional evaluation

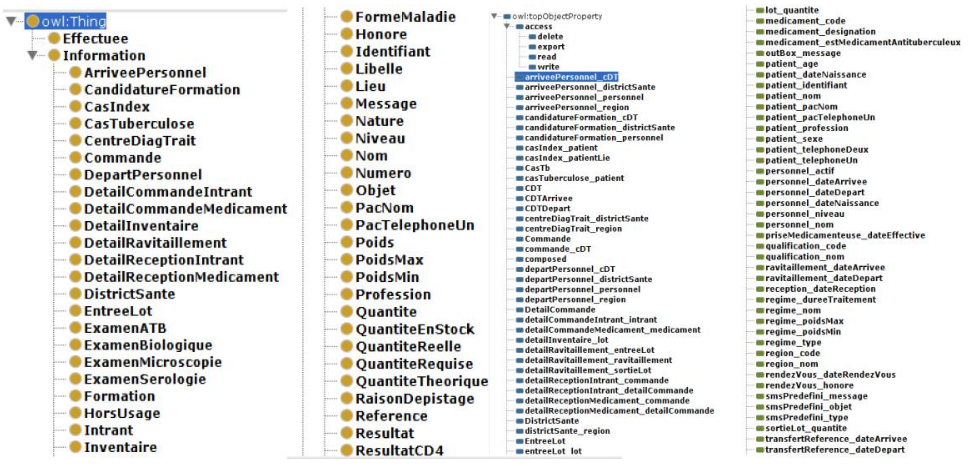
Functional evaluation was done at two levels:

1. **Concepts, properties and axioms evaluation:** The candidates terms were compared to the gold standard and we found that all extracted terms were in the gold standard. We also found that 16,18%, 7,43%, 13,94% of all candidates extracted to be concepts, properties, and axioms respectively were new terms and were validated by domain experts.
2. **Rules evaluation:** Rules extracted were validated by domain experts. Some of them, particularly those concerning user access to patient data have been better improved by domain experts during the validation. Then, these improvements were re-injected into the source code.

At the end of this evaluation, we found that from source code, we can recover all the terms contained in the gold standard. Moreover, we have found new concepts, properties, axioms and rules. Thus, we can conclude that source code may contain more knowledge than the meta-model and database and must be considered when building or enriching a domain ontology.

4.4.3. Structural evaluation

Our technique allows us to extract terms composed of metadata and data. In the structural evaluation, the



```
owl:Thing
    Effectuee
    Information
        ArriveePersonnel
        CandidatureFormation
        CasIndex
        CasTuberculose
        CentreDiagTrait
        Commande
        DepartPersonnel
        DetailCommandeIntrant
        DetailCommandeMedicament
        DetailInventaire
        DetailRavitaillement
        DetailReceptionIntrant
        DetailReceptionMedicament
        DistrictSante
        EntreeLot
        ExamenATB
        ExamenBiologique
        ExamenMicroscopie
        ExamenSerologie
        Formation
        HorsUsage
        Intrant
        Inventaire
        FormeMaladie
        Honore
        Identifiant
        Libelle
        Lieu
        Message
        Nature
        Niveau
        Nom
        Numero
        Objet
        PacNom
        PacTelephoneUn
        Poids
        PoidsMax
        PoidsMin
        Profession
        Quantite
        QuantiteEnStock
        QuantiteReelle
        QuantiteRequise
        QuantiteTheorique
        RaisonDepistage
        Reference
        Resultat
        ResultatCD4
```

```
owl:topObjectProperty
    access
        delete
        export
        read
        write
    arriveePersonnel_cDT
    arriveePersonnel_districtSante
    arriveePersonnel_personnel
    arriveePersonnel_region
    candidatureFormation_cDT
    candidatureFormation_districtSante
    candidatureFormation_personnel
    casIndex_patient
    casIndex_patientLie
    CasTb
    casTuberculose_patient
    CDT
    CDTArrivee
    CDTDepart
    centreDiagTrait_districtSante
    centreDiagTrait_region
    Commande
    commande_cDT
    composed
    departPersonnel_cDT
    departPersonnel_districtSante
    departPersonnel_personnel
    departPersonnel_region
    DetailCommande
    detailCommandeIntrant_intrant
    detailCommandeMedicament_medicament
    detailInventaire_lot
    detailRavitaillement_entreeLot
    detailRavitaillement_ravitaillement
    detailRavitaillement_sortieLot
    detailReceptionIntrant_commande
    detailReceptionIntrant_detailCommande
    detailReceptionMedicament_commande
    detailReceptionMedicament_detailCommande
    DistrictSante
    districtSante_region
    EntreeLot
    entreeLot_lot
```

```
lot_quantite
medicament_code
medicament_designation
medicament_estMedicamentAntituberculeux
outBox_message
patient_age
patient_dateNaissance
patient_identifiant
patient_nom
patient_pacNom
patient_pacTelephoneUn
patient_profession
patient_sexe
patient_telephoneDeux
patient_telephoneUn
personnel_actif
personnel_dateArrivee
personnel_dateDepart
personnel_dateNaissance
personnel_niveau
personnel_nom
priseMedicamenteuse_dateEffective
qualification_code
qualification_nom
ravitaillement_dateArrivee
ravitaillement_dateDepart
reception_dateReception
regime_dureeTraitement
regime_nom
regime_poidsMax
regime_poidsMin
regime_type
region_code
region_nom
rendezVous_dateRendezVous
rendezVous_honore
smsPredefini_message
smsPredefini_objet
smsPredefini_type
sortieLot_quantite
transfertReference_dateArrivee
transfertReference_dateDepart
```

Fig. 5. ontoEPICAM: Terms extracted from database and meta-model and validated by domain experts

quantity and the quality of the metadata were verified by comparing the candidate terms-concepts, properties to those of the gold standard. In fact, the extracted metadata make it possible to describe the data, their structure and, to indicate to which ontological knowledge each extracted term belongs. At the end of this evaluation, we found that the terms extracted from source code have the same structure as ontoEPICAM. Then, from the source code, a graph structure can be extracted that can be used to build an ontology.

4.5. *Knowledge extraction from GeoServer source code*

GeoServer is an Open Source map server developed in Java[10]. It allows users to edit, process and share geospatial data. The source code downloaded from github[11] contains 13038 files and 2150161 instructions. The HMMs we have built was used to extract the candidate to be concepts, properties, axioms and rules from GeoServer source code. About 178757 candidates terms have been extracted to become concepts, properties and axioms with 15787 false positives (precision 91.17%, recall=65.72%) and 38519 candidates conditions to become rules (precision 95.69%, recall=86.14%).

4.6. *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:

[10]http://geoserver.org/
[11]https://github.com/geoserver/geoserver

– **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).
– **Ease to use:** With our approach, to extract another type of elements, the knowledge engineer modifies the sets PRE, $POST$, $OTHER$, which is less difficult than to define/modify a regular expression or to 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 based on our approach is developed, it is easy to use.
– **Performance:** Unlike parsers, with our approach, one may have false positives when extracting knowledge. However, by training correctly the model, one may have good performances [30]. In addition, the number of terms redundancies that generally occurs in the source code makes that all the candidate terms can be extracted.

5. Conclusion and future work

In this paper, we are proposing an approach for knowledge extraction from JAVA source code. This approach consists of the definition of a Hidden Markov Model by providing PRE, $POST$, and $OTHER$ sets, training the HMM and using trained HMM to extract the knowledge from any JAVA source code. We experimented this approach by extracting knowledge from EPICAM-an epidemiological surveillance platform developed in JAVA and, GeoServer-an Open Source map server. We evaluated the knowledge extracted from EPICAM source code using a gold standard we built with domain experts. However, on one hand, structural evaluation shows that source code can be used to extract an ontological structure while on the other hand, functional evaluation shows that the source code is a more complete data source than the combination of the meta-model and the database. With the approach presented in this paper, by modifying the PRE, $POST$ and $OTHER$ sets, one can build a model for knowledge extraction from any other typed programming language.

Our approach has been tested for Java language having a particular structure. It would be interesting to generalize it for knowledge extraction from other programming languages having different programming paradigm. In fact, all programming languages have a structure making it possible to define PRE, $POST$ and $OTHER$ sets.

6. Acknowledgements

Our gratitude goes to Dr Nolna Désiré and Dr Abena Jean Louis from the National Program to Fight again Tuberculosis in Cameroon; Dr Gaetan Texier from Centre Pasteur du Cameroon; Dr TAPAMO Hippolyte from UMMISCO in Cameroon; Mr. Laurent Braak and Julien Dupouy from MEDES in France and finally, all those who contributed to the EPICAM project.

References

[1] K. Bontcheva and M. Sabou, Learning Ontologies from Software Artifacts: Exploring and Combining Multiple Choices, *Semantic Web Enabled Software Engineering* **17** (2014), 235.

[2] F. DeRemer and H.H. Kron, Programming-in-the-Large Versus Programming-in-the-Small, *IEEE Transactions on Software Engineering* **SE-2**(2) (1976), 80–86.

[3] R. Kaur and J. Sengupta, Software Process Models and Analysis on Failure of Software Development Projects, *CoRR* **abs/1306.1068** (2013).

[4] W. Scacchi, Understanding the requirements for developing open source software systems, *IEE Proceedings - Software* **149**(1) (2002), 24–39.

[5] C.V. Ramamoorthy, V. Garg and A. Prakash, Programming in the large, *IEEE Transactions on Software Engineering* **SE-12**(7) (1986), 769–783.

[6] A. Gómez-Pérez, M. Fernández-López and O. Corcho, *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. (Advanced Information and Knowledge Processing)*, Springer-Verlag New York, Inc., 2007.

[7] R. Studer, V.R. Benjamins and D. Fensel, Knowledge Engineering: Principles and Methods, *Data Knowledge Engineering.* **25**(1–2) (1998), 161–197.

[8] I. Bedini and B. Nguyen, Automatic Ontology Generation: State of the Art. http://bivan.free.fr/Docs/Automatic_Ontology_Generation_State_of_Art.pdf.

[9] P. Hitzler, M. Krötzsch and S. Rudolph, *Foundations of Semantic Web Technologies*, Chapman and Hall/CRC Press, 2010.

[10] A. Maedche and S. Staab, Semi-automatic engineering of ontologies from text, *Proceedings of the 12th Internal Conference on Software and Knowledge Engineering. Chicago, USA* (2000).

[11] L. Zhou, Ontology learning: state of the art and open issues, *Information Technology and Management* **8** (2007), 241–252.

[12] M. Shamsfard and A. Abdollahzadeh Barforoush, The State of the Art in Ontology Learning: A Framework for Comparison, *Knowl. Eng. Rev.* **18**(4) (2003), 293–316.

[13] P. Cimiano, *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387306323.

[14] S. Zhao, E. Chang and T.S. Dillon, Knowledge extraction from web-based application source code: An approach to database reverse engineering for ontology development., IEEE Systems, Man, and Cybernetics Society, 2008, pp. 153–159.

[15] D. Djuric, D. Gasevic and V. Devedzic, Ontology Modeling and MDA., *Journal of Object Technology* **4** (2005), 109–128.

[16] G. Gopinath and S. S, To Generate the Ontology from Java Source Code **2** (2011).

[17] J. Unbehauen, S. Hellmann, S. Auer and C. Stadler, Knowledge Extraction from Structured Sources, in: *Search Computing: Broadening Web Search*, S. Ceri and M. Brambilla, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 34–52.

[18] H.S. Pinto, A. Gómez-Pérez and J.P. Martins, Some issues on ontology integration, 1999.

[19] F. Kharbat and H. El-Ghalayini, *Building Ontology from Knowledge Base Systems*, INTECH Open Access Publisher, 2008.

[20] F. Cerbah and N. Lammari, Perspectives in Ontology Learning, AKA / IOS Press. Serie, 2014, p. 30, Chap. Ontology Learning from Databases: Some Efficient Methods to Discover Semantic Patterns in Data.

[21] N. Cullot, R. Ghawi and K. Yetongnon, DB2OWL : A Tool for Automatic Database-to-Ontology Mapping., in: *SEBD*, M. Ceci, D. Malerba and L. Tanca, eds, 2007, pp. 491–494.

[22] S. Wang, W. Wang, Y. Zhuang and X. Fei, An ontology evolution method based on folksonomy, *Journal of Applied Research and Technology* **13**(2) (2015), 177–187.

[23] A. Garciá-Silva, L.J. Garciá-Castro, A. Garciá and O. Corcho, Building Domain Ontologies Out of Folksonomies and Linked Data, *International Journal on Artificial Intelligence Tools* **24**(02) (2015).

[24] M. Hacherouf, S.N. Bahloul and C. Cruz, Transforming XML documents to OWL ontologies: A survey, *Journal of Information Science* **41**(2) (2015), 242–259.

[25] M. Fahad, ER2OWL: Generating OWL Ontology from ER Diagram, in: *Intelligent Information Processing IV: 5th IFIP International Conference on Intelligent Information Processing, October 19-22, 2008, Beijing, China*, Z. Shi, E. Mercier-Laurent and D. Leake, eds, Springer US, Boston, MA, 2008, pp. 28–37.

[26] F.A. Lisi, Learning Onto-Relational Rules with Inductive Logic Programming, *CoRR* **abs/1210.2984** (2012).

[27] A. Wroblewska, T. Podsiadly-Marczykowska, R. Bembenik, G. Protaziuk and H. Rybinski, Methods and Tools for Ontology Building, Learning and Integration âĂŞ Application in the SYNAT Project **390** (2012).

[28] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan and H.V. Jagadish, Regular Expression Learning for Information Extraction, in: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, Association for Computational Linguistics, Stroudsburg, PA, USA, 2008, pp. 21–30.

[29] M. Brunzel, The XTREEM Methods for Ontology Learning from Web Documents, in: *Proceedings of the 2008 Conference on Ontology Learning and Population: Bridging the Gap Between Text and Knowledge*, IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008, pp. 3–26.

[30] G.A. Fink, *Markov Models for Pattern Recognition. From Theory to Applications*, Springer-Verlag London, London, 2014.

[31] K. Seymore, A. Mccallum and R. Rosenfeld, Learning Hidden Markov Model Structure for Information Extraction, in: *In AAAI 99 Workshop on Machine Learning for Information Extraction*, 1999, pp. 37–42.

[32] S.J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd edn, Pearson Education, 2003, pp. 556–606873883.

[33] G. Zhou and J. Su, Named Entity Recognition Using an HMM-based Chunk Tagger, in: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, Association for Computational Linguistics, Stroudsburg, PA, USA, 2002, pp. 473–480.

[34] K. Dellschaft and S. Staab, Strategies for the Evaluation of Ontology Learning, in: *Bridging the Gap between Text and Knowledge Selected Contributions to Ontology Learning and Population from Text*, P. Buitelaar and P. Cimiano, eds, IOS Press, Amstedam, 2008-01.

[35] J. Brank, M. Grobelnik and D. Mladenić, A Survey of Ontology Evaluation Techniques, in: *Proc. of 8th Int. multi-conf. Information Society*, 2005, pp. 166–169.

[36] C. Brewster, H. Alani, S. Dasmahapatra and Y. Wilks, Data-driven Ontology Evaluation, in: *Proceedings of the Language Resources and Evaluation Conference (LREC 2004)*, Lisbon, Portugal, 2004, pp. 164–168.

[37] S. Iloga, O. Romain, L. Bendaouia and M. Tchuente, Musical genres classification using Markov models, in: *2014 International Conference on Audio, Language and Image Processing*, 2014, pp. 701–705.

[38] D. Binkley, M. Davis and D. Lawrie, To camelcase or under_score, in: *IEEE 17th International Conference on Program Comprehension*, ieee.org, 2009.

[39] Oracle, *Java Code Conventions*, 1997.

[40] A. Maedche, E. Maedche and R. Volz, The Ontology Extraction Maintenance Framework Text-To-Onto, in: *In Proceedings of the ICDM'01 Workshop on Integrating Data Mining and Knowledge Management*, 2001.