Faculty of Computer Science

ALEXANDRU IOAN CUZA UNIVERSITY of IAȘI

Alexandru Ioan Cuza University of Iaşi, Romania
Faculty of Computer Science

Automation Technology for Software Maintenance and System Administration

Extended Abstract

Author:
Andrei Panu

Supervisor:
Professor Dr. Henri Luchian

June 2017

Automation Technology for Software Maintenance and System Administration

Abstract

The increasing adoption of cloud computing services changed the way how software is developed, deployed, and executed. Nowadays the applications are usually composed of multiple services that run "in the cloud", in various environments. They can also be monolithic applications of different sizes. Even mobile applications that are installed locally use backend services hosted "somewhere". All these are executed in some pre-configured environments. After the initial development and deployment, keeping software applications and their execution environments up to date comes with some challenges. System administrators have no insight into the internals of the applications running on their infrastructure, thus if an update is available for the interpreter or for a library packaged separately on which an application depends, they do not know if the new release will bring some changes that will break some parts of the application. It is up to the development team to assess the changes and to support the new version. Such tasks take time to accomplish. The problem is very serious in the case of interpreted languages, because errors appear at runtime, only when certain blocks of code get executed, not initially when everything gets compiled, as is the case with compiled languages. Our focus is on interpreted applications. In this thesis we propose an approach consisting on automatic analysis of applications and automatic examination if the changes in a new version of a software dependency affect them. It is aimed at improving productivity in software administration and software maintenance fields. To accomplish this, we designed a process that models the human approach for learning, from the manuals, the functionalities offered by the interpreter of a programming language or by various libraries. Then, we designed a solution that automates the process, regardless of the programming language at hand, by using machine learning and natural language processing techniques. Finally, we developed and evaluated a platform that implements our approach.

Keywords: Information Extraction, Named Entity Recognition, Knowledge Engineering, Software Programs Ontology, Machine Learning, Web Mining, Semantic Web

Table of Contents

List of Publications

1. Andrei Panu. A Novel Method for Improving Productivity in Software Administration and Maintenance. In *Proceedings of the 12th International Conference on Software Technologies* (ICSOFT 2017), Spain, July 24–26, 2017. (accepted for publication)

2. Lenuta Alboaie, Sinica Alboaie, Andrei Panu. Levels of Privacy for eHealth Systems in the Cloud Era. In *Proceedings of the 24th International Conference on Information Systems Development* (ISD 2015), pp. 243-252, Harbin, China, 2015.

3. Serban Ungureanu, Andrei Panu, Lenuta Alboaie. A Cloud Gaming System Design Insights. In *Proceedings of the 12th International Conference on e-Business* (ICE-B 2015), pp. 144-151, Colmar, France, 2015.

4. Sabin C. Buraga, Andrei Panu. A Web Tool for Extracting and Viewing the Semantic Markups. In *Proceedings of the 6th International Conference on Knowledge Science, Engineering and Management* (KSEM 2013), pp. 570-579, China, 2013.

5. Andrei Panu, Sabin C. Buraga, Lenuta Alboaie. Qsense – Learning Semantic Web Concepts by Querying DBpedia. In *Proceedings of the 10th International Conference on E-Business* (ICE-B 2013), pp. 351-356, Iceland, 2013.

6. Claudia Gheorghiu, Andrei Panu, Lenuta Alboaie. A Semantic Web Platform for Legal Knowledge in Cloud. In *Proceedings of the 12th International Conference on Intelligent Software Methodologies*, Tools and Techniques (SOMET 2013), pp. 121-126, Hungary, 2013.

7. Lenuta Alboaie, Sinica Alboaie, Andrei Panu. Swarm Communication – a Messaging Pattern Proposal for Dynamic Scalability in Cloud. In *Proceedings of the 15th IEEE International Conference on High Performance Computing and Communications* (IEEE HPCC 2013), pp. 1930-1937, China, 2013.

8. Alina Gozman (Munteanu), Daniel Munteanu, Andrei Panu, Lenuta Alboaie, Sabin-Buraga. SINUX - Ubuntu spiced up with Semantic Web, *10th Networking in Education and Research - RoEduNet IEEE Conference*, ISSN 2247-5443, pp.199-204, Romania, 2011.

1 Thesis Overview

Contents

The increasing adoption of cloud computing services changed the way how software is developed, how it is deployed and executed, and how we use and interact with it (as developers or as simple users). Nowadays the applications are usually composed of multiple services that run in "in the cloud", in various environments. They can also be monolithic applications of different sizes. Even mobile applications that are installed locally use backend services hosted on servers. All these are executed in some pre-configured environments. After the initial development and deployment, some challenges appear regarding the maintenance of the execution environment and of the application. For example, system administrators face a dilemma when an update is available for the interpreter of a certain language (PHP, Python, etc.), especially if it is a major one. They are not the developers of the hosted applications or services that rely on the interpreter, thus they do not know if the update will bring changes that will break some parts of the software. It is also not in their responsibility to know any details about the internals of the applications. If the developers are faced with the task to support a new version of the interpreter, they must make an assessment of the changes brought by the update and the changes to be made in the application. The same problem appears when updating a library on which the software depends. These tasks require some effort and time.

In this thesis we propose a novel approach [22] that gives the administrators an insight for the mentioned problem, and the developers information about the changes to be made, all that in an automatic manner, improving their efficiency for such kind of tasks. Our solution uses machine learning and natural language processing techniques. It is independent on the language in which the software was developed.

1.1 Thesis Structure

The introductory chapter presents a short overview about the current landscape of cloud services, artificial intelligence, and Internet of Things, pointing out the complexity of the environments and the need for automation. We briefly state the addressed problem and introduce our vision. This chapter also presents the thesis' objectives and our contributions.

Chapter 2 presents the concepts and technologies on which our approach is based on. It contains an overview of information extraction task, with the focus on named entity extraction. The implementation of our approach is deployed in cloud and is designed to be easily integrated in various environments, thus overviews on cloud computing and integration technologies are also presented.

Chapter 3 presents the addressed problem and details our approach. It also contains an analysis on related solutions that are similar to our proposal.

Chapter 4 presents the design and implementation of the platform that enables our vision. It also presents various experimental results and provides some insights about deployment scenarios.

Chapter 5 presents key aspects about modeling and sharing knowledge and the enabling technologies, with a focus on the Semantic Web. In this chapter we also propose an ontology for our addressed domain, used for structuring the extracted information according to the defined concepts.

Chapter 6 presents various scenarios in multiple fields in which our approach has applicability. It also offers some insights regarding potential privacy concerns that can appear by employing our solution.

In the last chapter we present the concluding remarks and future directions of research that are enabled by our work.

1.2 Contributions

In this thesis we propose a novel approach aimed at improving productivity in software administration and software maintenance fields. Our main objective is to provide a framework for the optimization of the process of maintaining software applications up to date. The means to achieve that consist of an original method which assists all the persons involved in the respective activities by automating some specific tasks, thus reducing the time needed to accomplish them. The contributions of this thesis can be summarized as follows:

- we designed a process that models the human approach for learning from the manuals the functionalities offered by the interpreter of a programming language or by various software libraries;

- we designed a solution that automates the process and reduces the time needed to solve the addressed problem, regardless of the programming language at hand or the used software library;
- we developed and evaluated the platform that implements the approach;
- we proposed a comprehensive ontology that models the programming domain, which we use for creating the knowledge base that contains all the concepts utilized/learned by the platform;
- we proposed a named entity hierarchy for the programming domain, defining new possible types to be extracted in the Named Entity Recognition field;
- we provide insights into the way our solution can be applied in other areas: for the Semantic Web and for software development in the context of Internet of Things.

1.3 A New Approach to Automation in Software Maintenance and System Administration

In this vast landscape of software applications of different sizes that run in execution environments configured directly on barebone servers, on virtual machines, or in containers, the management of the applications and the environments after the initial development and deployment is a challenge. Our proposal addresses a specific management problem [15, 14], regarding keeping the software up to date.

System administrators maintain the infrastructure for running different applications. One of their major tasks is to maintain the systems up to date and this generates some difficulties. When they are faced with the situation of updating the execution environment for the deployed applications, for example updating an interpreter (for PHP, Python, Perl etc.) to a new version, they have to answer questions like:

- Will the existing applications run on the new version of the interpreter?
- Are there any parts of the applications that will not run because of the changes?

These are not questions that can be answered easily, because the administrator is usually not the developer of the application(s). The same thing applies in case of libraries which are packaged and installed independently and on which the applications depend. Given the fact that the sysadmin does not have any knowledge about the application (except the necessary versions of the interpreter/libraries when the application was developed and deployed), he/she can only base its decision on assumptions to make the update. One intuitive assumption is that if there is a minor version update for the interpreter or library, everything should be fine, as no major changes in functionality occurred. In the majority of cases, this holds true, but it is still an assumption, not a certainty. The problem arises when there is a major update and the development team does not plan to update the application to support the new release. The problem is very serious in the case of interpreted languages, because errors appear at run-time, only when certain blocks of code get executed, not initially when everything gets compiled, as is the case with compiled languages. Thus, some parts of the application may work, while other parts may not. The system administrator simply will not know if there will be parts of the application that will not execute, he/she is not the developer, he/she is not the tester. This problem scales, because a single administrator can have multiple applications running on his/her infrastructure. Our proposal offers a solution for this situation, by automatically analyzing and providing information about whether the new version will bring some changes that will break the application or not.

Nowadays there is also a shift regarding the management of the execution environment, from the dedicated system administrators to the teams developing the applications (DevOps), by using virtual machines or containers like *Docker*. This does not solve the problem, only shifts it to the developers, although it does not mean that administrators do not care about the situation of the environments of the containers running on their infrastructure. Thus, when the development team wants to update the interpreter or some libraries used, it faces the same problem described above. Even though the team knows the application, it does not know exactly which blocks of code will execute and which will not. The solution is to analyze the changelog/migration guide and to assess the changes that need to be done. This manual procedure is time consuming. Our solution helps reduce this time.

The approach that we propose towards solving this problem is a technology that is able to automatically scan the code and verify if the used functionalities are still supported in a targeted version of the interpreter/library. This technology is independent of the used programming language. As we have stated earlier, the focus is on interpreted languages. This technology is comprised of three parts:

1. A tool that automatically analyzes the code, extracts the used functionalities, and queries a knowledge base that helps to answer the following questions: is the functionality X supported in the new version N? If not, what are the changes that were made?

2. A knowledge base [20] created automatically that contains information about the functionalities supported in every version of the interpreter/library;

3. A platform that extracts specific entities from online or offline manuals, independently of the programming language, and populates the knowledge base.

The general process is depicted in Figure 1-1. The tool is the part that must have access to the code and which uses the (remote) knowledge base to verify if there are functionalities

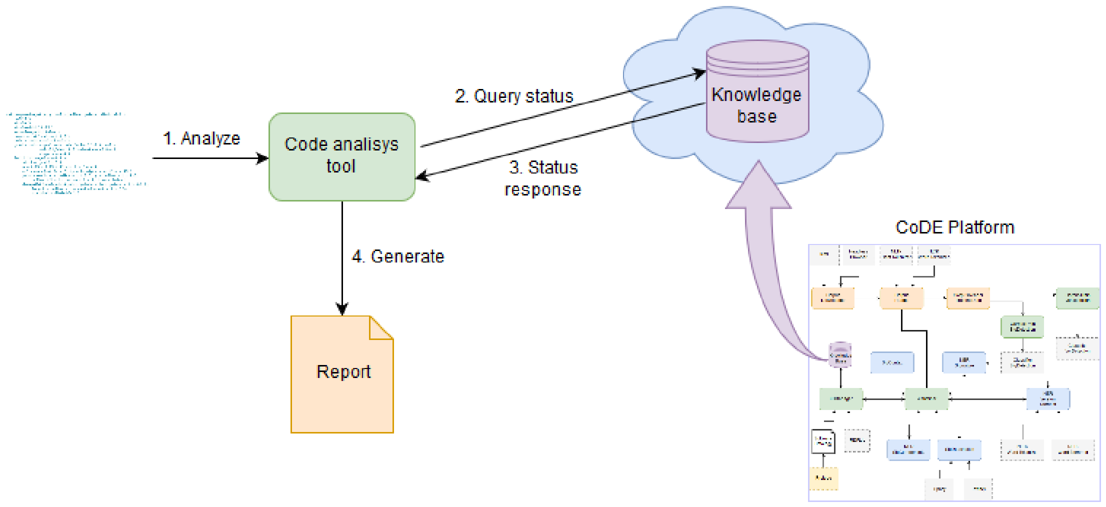

Figure 1-1: General Workflow.

used that are not supported anymore (or are marked to be removed in the future) in the targeted version. It generates a report based on the findings. The key enabler of our technology is the knowledge base. The most important aspect is the contained information and the way it is obtained. For the current version of the platform that populates the knowledge base, the functionalities taken into consideration are the supported functions in all versions of the interpreter/library (the provided APIs). Thus, the analysis tool is a pretty basic component, all that it needs to do is to extract all the functions in the code, eliminate those declared locally, and query the knowledge base to check for support in the targeted version. The only functionality that is more complex is the filtering of the functions provided by a certain used library or by the interpreter.

In this thesis, the focus of our work is on the platform that creates the knowledge base. As we have mentioned, the data contained consists of details regarding all the supported functions in the interpreter/library. For each function, we have different attributes, like its signature's components (the function's name, the number of arguments, the types of arguments, the order of the arguments), the return type, its short description, its availability (supported, deprecated, obsolete), and the version number of the interpreter/library in which it is supported. All the information is extracted from online manuals available on the Web or offline ones. The platform that we designed and implemented is capable to automatically extract the desired data, independently of the content (certain manuals for certain languages/libraries) or the structure of the web pages. The extraction technology does not have implemented any adapters for specific manuals. It supports any manual for any language. The only restriction is regarding the syntax used for writing the functions in the manual. This capability is achieved using machine learning algorithms and different natural language processing (NLP) techniques [25, 9].

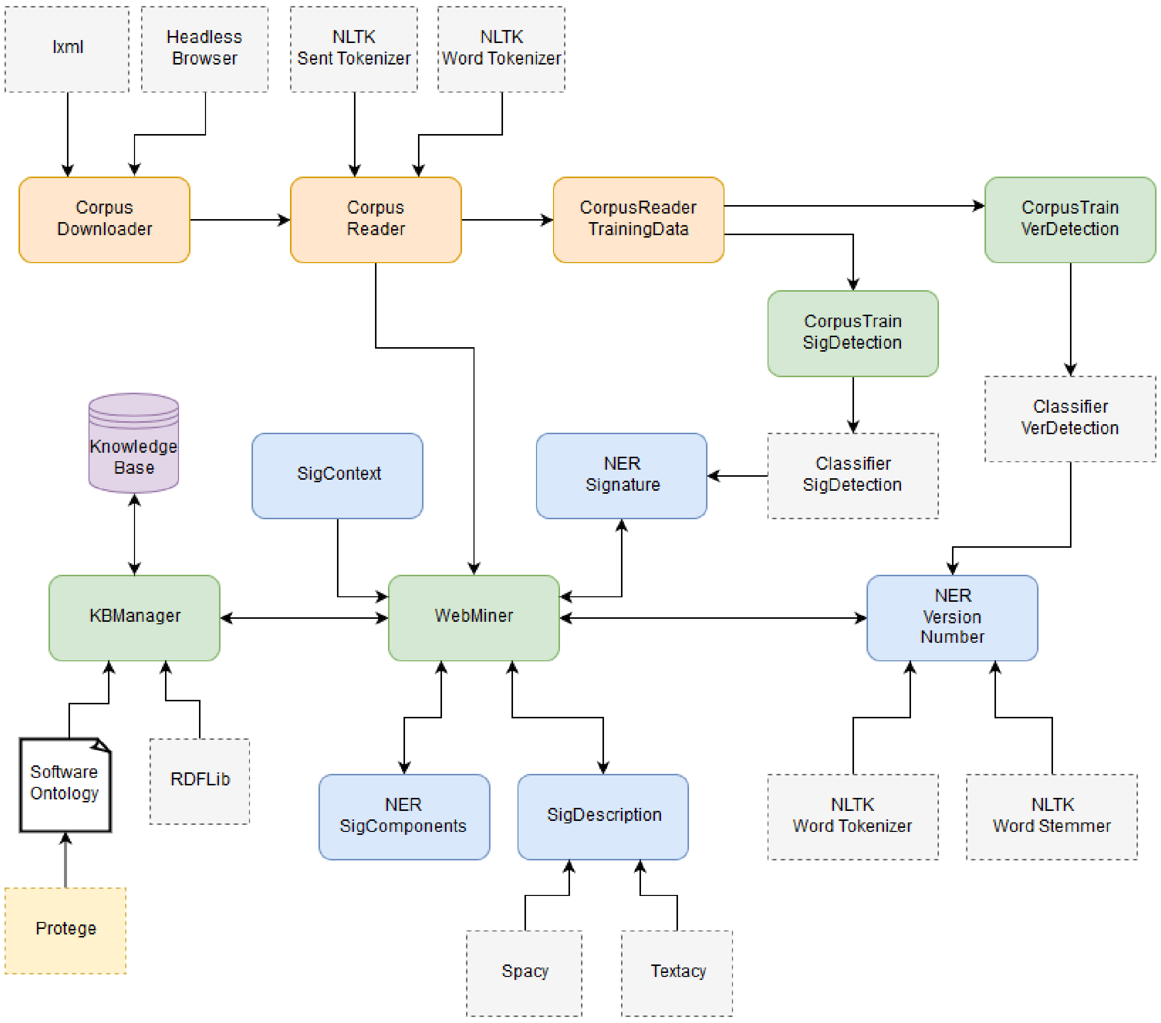

Figure 1-2: CoDE Platform Architecture

1.4 CoDE – The Extraction Platform

In this section we present the architecture of the platform that is capable to automatically access information on the Web, identify and extract the specific entities, and populate the knowledge base. Its components are decoupled and context independent. Each component was designed to be part of a distributed architecture. It is fully implemented in Python. Figure 1-2 depicts the system's architecture. The platform has four main components, *CorpusTrain SigDetection*, *CorpusTrain VerDetection*, *WebMiner*, *KBManager*, and its functionality is split into two phases, a training phase and an extraction phase.

In the first phase we train two Naive Bayes classifiers to detect signatures and version numbers in manuals pages. Although it is one of the simplest models, we have chosen Naive Bayes because it is known to give good results for various use cases and because it does not need a large amount of training instances. We have manually created two data sets for training.

For the classifier that detects signatures, the training set contains 279 positive examples and 296 negative ones, which were taken from PHP and Python manuals. The positive examples are entire sentences that contain signatures. The negative examples are sentences that contain various explanations which mention at least a function, or different blocks of code found on the respective pages. They have been chosen specifically like this in order to lower the rate of false positives. The first 70% of the labeled instances for each label are used for training, and the rest for testing. The trained model has an accuracy of around 98% on the test data set. The *CorpusReader* and *CorpusReader TrainingData* components are used for loading the training data into memory, and *CorpusTrain SigDetection* creates the feature sets, splits the data, and trains the classifier, generating *Classifier SigDetection*. The generated classifier is further used by *NER Signature* component, which offers a single service: it receives a list of sentences and identifies which ones are signatures.

For the classifier that detects version numbers, the training instances are created from examples taken from PHP, Python, and jQuery manuals. The total number of instances is 518 (62 positive examples and 456 negative ones). The negative set contains various numbers that do not represent a version. The feature sets are split, the first 70% of the labeled instances for each label are used for training, and the rest for testing. The model has an accuracy of around 95%. The same two components mentioned above are used for loading the training data. *CorpusTrain VerDetection* is the component which creates the feature sets, does the splitting and trains the classifier, generating *Classifier VerDetection*. The generated classifier is used by *NER VersionNumber* component, which offers a single service: it receives signatures with their descriptive contexts and identifies the words that represent version numbers. For both models, we say they that have an accuracy value around a certain percent because at train time we randomize the instances, thus the value is slightly different on different executions.

The second step, the extraction phase, consists of accessing online information, analyzing it and extracting specific knowledge. The main orchestrator of all these operations is *WebMiner* component. The other components involved are *CorpusDownloader*, *CorpusReader*, *NER Signature*, *SigContext*, *NER VersionNumber*, *NER SigComponents*, *SigDescription*. The extracted data is then saved in the knowledge base, operation managed by *KBManger*. *CorpusDownloader* is the component which is pointed to the online pages that contain the desired data. It accesses the content using a headless browser, thus providing the information exactly like a human sees it rendered in a browser, without any code in it (e.g., HTML tags, CSS properties, JavaScript code). Further, the information is split into sentences and each sentence in a list of words by *CorpusReader*. *WebMiner* sends this data to *NER Signature* component, which identifies the existing signatures by using the trained classifier. Each signature is then sent to *NER SigComponents* to extract the following elements: the function's name, its return type, and its parameters. For each parameter, we have its name, its type, and its order. This is accomplished by using hand-coded rules based on regular expressions. All this data is further sent to *NER SigContext*, which identifies the

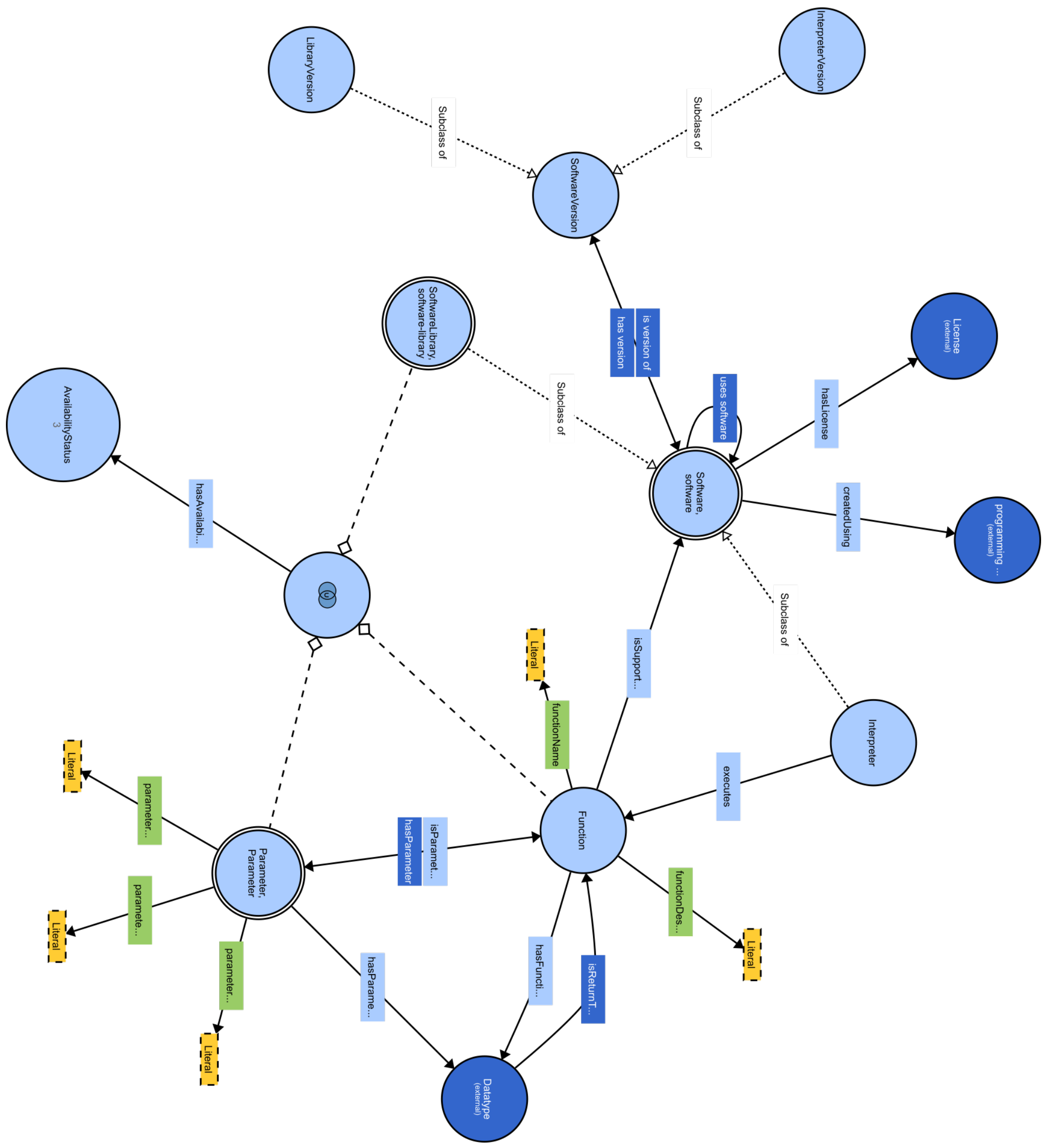

Figure 1-3: Visual representation of the ProgO Ontology (image generated using VOWL [16])

descriptive contexts of each signature, through the use of hand-coded rules. We define a descriptive context as the list of sentences that contain details about a single signature. The descriptive context is sent to *NER VersionNumber* to extract the version number(s) of the interpreter/library in which the function is available or in which it was marked as deprecated or it was removed. This component uses the trained classifier and some hand-coded rules that check the context of the word representing a version number to identify the status (available, deprecated, removed). The final step is to extract a single description of the function from the context. This is done by *SigDescription*, which uses a dependency parser to analyze the sentences and hand-coded rules that model some observed patters used in formulating descriptions. All the extracted information is provided to *KBManager*, which manages the knowledge base.

The *KBManager* component receives the extracted information and generates instances for the knowledge base. The data is expressed using the RDF (Resource Decription Framework) model [1]. The RDF triples contain information structured according to concepts illustrated in ProgO Ontology, an ontology [20, 11] we designed for our addressed domain. The existing ontologies that are specific to this domain [17, 33, 21] do not contain all the conceptual descriptions that are needed in our case. ProgO Ontology can be classified as an Application Ontology [12], meaning it was designed with the intent to be use as part of our solution. In the process of designing such an ontology, we have to take into consideration two requirements: reusing as much as possible existing classes and properties from existing ontologies and taking into consideration that any new added classes or properties have to fit the intent of the domain of application for the ontology.

The application domain of the ProgO ontology is the representation of parts of a (computer program) source code with the intent of tracking its exposed functions and aspects like availability status, parameters and at the same time the returned datatypes. Thus some of the important concepts that are encapsulated by our ontology consist of: function name, function return type, function description, parameter name, parameter type, parameter order, parameter description and software version number and availability status. The ontology's representation is available in Figure 1-3.

The ontology was created using Protege [2]. The data is stored in a Triple Store provided by Virtuoso [3].

1.5 Experimental Results

We have conducted tests on various online manuals in order to evaluate the general performance of the platform in extracting all the needed information. We have pointed the platform to extract data from different pages selected randomly from Node.js (version 7.7.0) [4], Ruby [5], PHP [6], Python (version 3.6.0) [7], and Laravel (version 5.4) [8] manuals.

Table 1.1 summarizes the performance of the platform in detecting the signatures for each case. The second column contains the total number of functions that exist in each page, and the last column the percentage of the detected signatures. We obtained the following results:

- In case of Node.js, it extracted all the functions, without any false positives, being

[1] https://www.w3.org/RDF/
[2] http://protege.stanford.edu/
[3] https://virtuoso.openlinksw.com/
[4] https://nodejs.org/api/util.html
[5] https://ruby-doc.org/docs/ruby-doc-bundle/Manual/man-1.4/function.html
[6] http://php.net/manual/en/function.chmod.php
[7] https://docs.python.org/3/library/os.path.html
[8] https://laravel.com/docs/5.4/helpers

Table 1.1: Signature detection performance.

Man page	**No. of functions**	**Detection rate**
Node.js	26	100%
Ruby	59	64.4%
PHP	1	100%
Python	30	100%
Laravel	80	98.75%

able to filter all other existing functions in the code examples or in the page's menu;

- In case of PHP, it extracted the only existing function. It also identified 5 more (false positives), because there are a lot of comments with code examples in the respective page, being unsuccessful in filtering all of the functions mentioned there;
- In case of Python, it extracted all the functions, without any false positives, being able to filter all other existing functions in the code examples or in the page's menu;
- In case of Laravel, it missed one function who's name contains only one character (namely the *e()* function). We did not have any false positives;
- In case of Ruby, it detected only 38 functions from a total of 59. This result is not very good because there are many functions in the page which are not written using parenthesis (e.g., *at_exit*, *binding*, *chop*, *fork*), this being a very important feature when searching for the pattern.

For each of the identified functions, the component *NER SigComponents* successfully extracted all of their elements (e.g., the name and return type, the parameters, etc.). Regarding the extraction of version numbers, we obtained the following results:

- for Node.js: the page contains 26 functions, 5 of them having specified a single version representing when it was added, the rest containing two versions, when it was added and since when it is deprecated. The platform correctly identified all of them, with their corresponding status. It also identified the general version (*7.10.0)* mentioned outside of the contexts of functions, having a lower priority due to the existence of the others. We do not have any false positives or false negatives;
- for Ruby: the page does not contain any version numbers, thus the system correctly did not identify any;
- for PHP: the page contains a single function with 3 version numbers mentioned (*PHP 4*, *PHP 5*, *PHP 7*). The platform successfully identified all of them;
- for Python: the page contains 30 functions, 4 of them having specified two versions, 1 of them having specified 3 versions, and the rest only 1 version. The system correctly

Table 1.2: Description detection performance.

Man page	No. of det. functions	Detection rate
Node.JS	26	76.9%
Ruby	38	68.4%
PHP	6	100%
Python	30	93.3%
Laravel	79	96.2%

identified all of them, without any false positives or negatives. It also identified the general version (*3.6.0*) outside of the functions' descriptive contexts.

- for Laravel: the page does not mention any details about versions inside the context of the functions, thus the system correctly did not identify any. It identified the general version (*5.4*) mentioned outside the contexts.

For the detection of the functions' descriptions, the system obtained the results presented in Table 1.2. The second column represents the total number of signatures that were detected (each having a single description), and the last one the percentage of the detected descriptions. In case of Node.JS, it failed to identify 6 descriptions. For Ruby, it missed 12 descriptions. Regarding PHP, it successfully identified the description of the single function. For the other 5 false positives, it did not detect anything because they are examples of code, thus they do not have descriptions. In case of Python, it did not identify the descriptions of 2 functions. Finally, for Laravel, it failed to extract 3 descriptions.

1.6 Related Solutions

Verifying Code Compatibility

The standard approach with updating the execution environment of an application (e.g., the interpreter or some libraries it depends on) is to establish a test environment (nowadays usually in a virtual machine) with the new version(s) and to test the entire application. This is theoretically the best approach, as it is done by professionals who know the application and verify all the functionalities offered. The purpose is to make sure that the entire code gets executed. The disadvantages with this approach is that it is time consuming and can be prone to human errors (e.g., they can omit testing some functionalities). The burden of doing manual tests can be reduced or completely eliminated by the existence of automated tests. Unfortunately, the situations where automated tests for the entire application are developed are very rare. The most common situation is when the core components of the application are covered by automated tests, leaving the others for manual testing, or no tests at all.

There are some tools that were created to automate this compatibility verification, but they are rather limited in scope and are also language dependent. For example, for PHP, there is a tool, *PHP CodeSniffer* [9], that tokenizes the source code and detects and fixes violations of code formatting standards, without executing the code. This is used by another tool, *PHPCompatibility* [10], that contains a set of sniffs for the former and is able to check for PHP version compatibility. The sniffs are a *PHPCompatibility* code standard created to detect the use of backwards incompatible code. This tool generates a report containing all the identified problems. Compared to our solution, its advantage is coverage for many more changes (but not 100%), not only deprecated/obsolete functions. As disadvantages, we can mention that the sniffs (its "knowledge base") are created manually, supports only PHP, and only changes introduced since PHP 5.0 and not earlier versions. Various similar custom tools were developed for different scopes and languages and libraries, like *jquery-migrate* [11] for migrating older jQuery code to jQuery 3.0+, *wp-deprecated-checker* [12], a Wordpress deprecated function checker, *2to3* [13], an automated Python 2 to 3 code translation program, *PHP7MAR* [14] or *php7cc* [15], which are PHP 7 migration assistants, *depcheck* [16], another PHP deprecated function checker. The problems with these are the same, they have restricted coverage and are developed for a single language or library. More tools can be found for different languages/libraries, but not quite for everything.

There are also professional-grade IDEs that offer automatic code inspection tools. One such example is *PhpStorm* [17]. It features a static code analyzer that can detect various code inefficiencies. The most common tasks covered are the detection of probable bugs, the identification of code that never gets executed, the detection of possible performance problems, the violation of coding guidelines and standards, the conformance to specifications. All these inspections are manually created. Support for the detection of code compatibility problems for various versions of the PHP interpreter is very limited, given by the existence of manually created inspections that check for this particular situations. The focus is not on offering this kind of functionality. Another category of tools that offer code analysis are "linters" (e.g., PHPMD [18], Pylint [19]). But again, their focus is on checking for different coding standard violations and various errors in how the code is written, not on verifying the use of deprecated/obsolete functions in interpreters/libraries. Table 1.3 summarizes our findings, according to the following criteria:

- language independence – if the solution supports multiple languages or was developed

[9]http://pear.php.net/package/PHP_CodeSniffer/
[10]https://github.com/wimg/PHPCompatibility
[11]https://github.com/jquery/jquery-migrate
[12]https://gist.github.com/jbuchbinder/7419000
[13]https://docs.python.org/3/library/2to3.html
[14]https://github.com/Alexia/php7mar
[15]https://github.com/sstalle/php7cc
[16]https://www.phpclasses.org/package/9084-PHP-Find-deprecated-functions-and-suggest-replacements.html
[17]https://www.jetbrains.com/phpstorm/
[18]https://phpmd.org/
[19]https://www.pylint.org/

Table 1.3: Comparison of related solutions for compatibility checking

Solution/Criterion	**language independence**	**library support**	**automatic kb creation**	**check for function support**	**check other features**
Our solution	✓	✓	✓	✓	×
PHPCompatibility	×	×	×	✓	✓
jquery-migrate	×	×	×	✓	✓
wp-deprecated-checker	×	×	×	✓	✓
2to3	×	×	×	✓	✓
PHP7MAR	×	×	×	✓	✓
php7cc	×	×	×	✓	✓
depcheck	×	×	×	✓	×
PhpStorm	×	×	×	×	✓
PHPMD	×	×	×	×	✓
PyLint	×	×	×	×	✓

for a single language;

- library suppport – if the solution checks for features offered by various libraries (not for just a single one);
- automatic knowledge base creation – if the solution's source of information is manually created and maintained or automatically;
- check for function support – if the solution is able to verify if all the used functions are still available, or which has been deprecated, etc.;
- check for other features – if the solution is able to check for other changes, like language constructs, syntax, keywords, etc.;

A lot of academic research is conducted towards automating the assessment of backward compatibility of software components, but the solutions do not offer the same functionality as our proposal, they provide a complementary one [23, 31, 13, 29]. Their source of information is software repositories, which they monitor and analyze. The approaches are based on different techniques that evaluate the differences between the old source code and the new one (entire code or only the interfaces). The focus is on assessing if a new version of a software component is compatible with its old version, from a functional point of view. They do not mention anything about the capability to provide information about the change of support status of functionalities in different versions of libraries/interpreters and also they work by comparing two code repositories, which is not our case.

Extracting Entities in the Programming Domain

Our proposal has also a contribution regarding the possibility to identify and extract entities in the programming domain. In [18] we can find a good overview on different supported entity types for the NER task. The term ”named entity” was first used in the MUC-6 (Sixth

Table 1.4: Comparison of related commercial solutions for entity extraction

Solution/Extraction domain	Programming	General
CoDE	✓	×
IBM Watson (AlchemyAPI)	×	✓
OpenCalais	×	✓
MeaningCloud	×	✓

Message Understanding Conference) [10]. The extraction tasks were focused on identifying proper names (e.g., person names, organization names, location names) and numeric expressions (e.g., time, date, money, percent). Regarding the recognition of names [3], the three examples mentioned represent the most studied entity types, categorized as "enamex". Fine-grained subcategories were approached, like "politician" and "entertainer" for "person" [8], or "city", "state", "country" for "location" [7]. The CONLL conferences use the type "miscellaneous" to include proper names that do not fall in the "enamex" category. Other categories created in MUC are "timex", which includes entity types like "date" and "time", and "numex", which includes entity types like "money" and "percent". For specific needs, marginal types were defined, like "film" and "scientist" [5], "email" and "phone number" [32], "research area" and "project name" [34], "book title" [2, 32], "job title" [4]. A lot of work towards automatic entity extractions is also done in the bioinformatics field, recognizing types such as "protein", "DNA", "RNA", "cell type", "cell line" [30, 28, 27]. Some related studies were done towards "drug" [24] and "chemical" [19] names.

Another category of NER systems, called "open domain" NERs [1, 6], include those that were not developed for specific types, they do not restrict the possible entity types to extract. Although these systems can be used in theory for any domain, all the experiments and the fine tuning were done for the most well known entity types in the newspaper domains. Furthermore, for this context of "open domain" NERs, an extended named entity hierarchy, which includes around 200 categories, was defined [26]. It added categories like color, animal, religion, substance, and subcategories like airport, museum, river, color. Again, their hierarchy is designed for newspaper domains, it is not a specialized one for a particular domain. Thus, it does not contain anything related to the programming domain. We could not find any NER system that was tested and works with entities in our addressed domain. In this regard, we propose an extended NE hierarchy that contains new categories that cover the concepts our approach works with. It is not intended to be a comprehensive hierarchy for computer software. Depending on the needs, it will be extended.

Known commercial solutions that offer natural language processing for text analysis also focus on extracting general terms. We tested several products on some manual pages chosen randomly from PHP [20], Python [21], and Node.js [22]. Table 1.4 summarizes our findings.

[20] http://php.net/manual/en/function.chmod.php
[21] https://docs.python.org/3/library/os.path.html
[22] https://nodejs.org/api/util.html

IBM's Watson Natural Language Understanding module [23], which integrated AlchemyAPI, one of the most well known companies that offered NLP solutions, has a broad list of supported entity types and subtypes [24]. Among them, we can find some subtypes applicable to our domain, as "ProgrammingLanguage", "ProgrammingLanguageDesigner", "Software", "SoftwareDeveloper", "SoftwareLicense". On our test cases, it did not detect anything of interest. From Node.js page, it detected the terms "len" as being a person and "1 2 3" as being a quantity. From PHP page it could not extract anything. From Python page, it detected the terms "unc" as an organization and "r" as a person. Taking into account the mentioned types that are supported, we consider that it should have detected at least the terms "PHP", "Node.js", and "Python".

Another solution that is well known and very much used is OpenCalais [25]. It correctly detected the programming languages and some other terms in the IT domain, but nothing relevant for our purpose.

MeaningCloud [26] also offers text analytics services. Its online demo limits the input text to 500 characters, therefore we have selected only 8 functions from the Node.js page and 8 functions from the Python page. The PHP page contains only a single function, but has many examples of code. From this page we have selected only the function with its descriptive section. Unfortunately, it also did not detect anything relevant.

All these results are expected, because the solutions are designed for general newspaper domains, not for the particular domain of programming that our solution addresses. They recognize entities from categories like people, places, dates, companies, products, not entities in the programming context.

[23] https://www.ibm.com/watson/developercloud/natural-language-understanding.html

[24] https://www.ibm.com/watson/developercloud/doc/natural-language-understanding/entity-types.html

[25] http://www.opencalais.com/

[26] https://www.meaningcloud.com/

Bibliography

[1] Enrique Alfonseca and Suresh Manandhar. An Unsupervised Method for General Named Entity Recognition And Automated Concept Discovery. In *In: Proceedings of the 1 st International Conference on General WordNet*, 2002. (Cited on page 14.)

[2] Sergey Brin. Extracting Patterns and Relations from the World Wide Web. In *The World Wide Web and Databases, International Workshop WebDB'98, Valencia, Spain, March 27-28, 1998, Selected Papers*, pages 172–183, 1998. (Cited on page 14.)

[3] Sam Coates-Stephens. The Analysis and Acquisition of Proper Names for the Understanding of Free Text. *Computers and the Humanities*, 26(5-6):441–456, 1992. (Cited on page 14.)

[4] William W. Cohen and Sunita Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-Markov extraction processes and data integration methods. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 89–98, 2004. (Cited on page 14.)

[5] Oren Etzioni, Michael J. Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the Web: An experimental study. *Artif. Intell.*, 165(1):91–134, 2005. (Cited on page 14.)

[6] Richard Evans. A framework for named entity recognition in the open domain. *Recent Advances in Natural Language Processing III: Selected Papers from RANLP*, 260(267-274):110, 2003. (Cited on page 14.)

[7] Michael Fleischman. Automated Subcategorization of Named Entities. In *Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Companion Volume to the Proceedings of the Conference: Proceedings of the Student Research Workshop and Tutorial Abstracts, July 9-11, 2001, Toulouse, France.*, pages 25–30, 2001. (Cited on page 14.)

[8] Michael Fleischman and Eduard H. Hovy. Fine Grained Classification of Named Entities. In *19th International Conference on Computational Linguistics, COLING 2002, Howard International House and Academia Sinica, Taipei, Taiwan, August 24 - September 1, 2002*, 2002. (Cited on page 14.)

[9] Ralph Grisham. Information Extraction: Capabilities and Challenges, 2012. (Cited on page 5.)

[10] Ralph Grishman and Beth Sundheim. Message Understanding Conference- 6: A Brief History. In *16th International Conference on Computational Linguistics, Proceedings*

of the Conference, COLING 1996, Center for Sprogteknologi, Copenhagen, Denmark, August 5-9, 1996, pages 466–471, 1996. (Cited on page 14.)

[11] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993. (Cited on page 9.)

[12] N. Guarino. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy.* IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st edition, 1998. (Cited on page 9.)

[13] Ahmed Hatim, Ali A.S.A. Elgamal, Hisham E. Elshishiny, and Mahmoud Rashad Ibrahim. Verification of backward compatibility of software components. `https://www.google.com/patents/US20150169320?utm_source=gb-gplus-sharePatent`, 2015. (Cited on page 13.)

[14] ISO/IEEE. International Standard - ISO/IEC 14764 IEEE Std 14764-2006. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998*, pages 1–46, 2006. (Cited on page 3.)

[15] ISO/IEEE. ISO/IEC/IEEE Standard for Systems and Software Engineering - Software Life Cycle Processes. *IEEE Std 12207-2008*, pages 1–138, Jan 2008. (Cited on page 3.)

[16] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. Visualizing Ontologies with VOWL. *Semantic Web*, 7(4):399–419, 2016. (Cited on page 8.)

[17] James Malone, Andy Brown, Allyson Lister, Jon Ison, Duncan Hull, Helen Parkinson, and Robert Stevens. The software ontology (swo): a resource for reproducibility in biomedical data analysis, curation and digital preservation. In *Journal of Biomedical Semantics*, 2014. (Cited on page 9.)

[18] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007. (Cited on page 13.)

[19] Meenakshi Narayanaswamy, K. E. Ravikumar, and K. Vijay-Shanker. A Biological Named Entity Recognizer. In *Proceedings of the 8th Pacific Symposium on Biocomputing, PSB 2003, Lihue, Hawaii, USA, January 3-7, 2003*, 2003. (Cited on page 14.)

[20] Natalya F. Noy, Deborah L. McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001. (Cited on pages 4 and 9.)

[21] Daniel Oberle, Stephan Grimm, and Steffen Staab. An ontology for software. In *Handbook on ontologies*, pages 383–402. Springer, 2009. (Cited on page 9.)

[22] Andrei Panu. A novel method for improving productivity in software administration and maintenance. In *12th International Conference on Software Technologies (ICSOFT 2017), Madrid, Spain, July 24–26 (accepted for publication)*, 2017. (Cited on page 1.)

[23] A Ponomarenko and V Rubanov. Backward compatibility of software interfaces: Steps towards automatic verification. *Programming and Computer Software*, 38(5):257–267, 2012. (Cited on page 13.)

[24] Thomas C Rindflesch, Lorraine Tanabe, John N Weinstein, and Lawrence Hunter. EDGAR: extraction of drugs, genes and relations from the biomedical literature. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, page 517. NIH Public Access, 2000. (Cited on page 14.)

[25] Sunita Sarawagi. Information extraction. *Foundations and trends in databases*, 1(3):261–377, 2008. (Cited on page 5.)

[26] Satoshi Sekine and Chikashi Nobata. Definition, Dictionaries and Tagger for Extended Named Entity Hierarchy. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal*, 2004. (Cited on page 14.)

[27] Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, pages 104–107. Association for Computational Linguistics, 2004. (Cited on page 14.)

[28] Dan Shen, Jie Zhang, Guodong Zhou, Jian Su, and Chew-Lim Tan. Effective adaptation of a hidden markov model-based named entity recognizer for biomedical domain. In *Proceedings of the ACL 2003 workshop on Natural language processing in biomedicine-Volume 13*, pages 49–56. Association for Computational Linguistics, 2003. (Cited on page 14.)

[29] Efstratios Tsantilis. Method and system to monitor software interface updates and assess backward compatibility. `https://www.google.com/patents/US7600219`, 2009. (Cited on page 13.)

[30] Yoshimasa Tsuruoka and Jun'ichi Tsujii. Boosting precision and recall of dictionary-based protein name recognition. In *Proceedings of the ACL 2003 workshop on Natural language processing in biomedicine-Volume 13*, pages 41–48. Association for Computational Linguistics, 2003. (Cited on page 14.)

[31] Yannick Welsch and Arnd Poetzsch-Heffter. Verifying backwards compatibility of object-oriented libraries using boogie. In *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs*, pages 35–41. ACM, 2012. (Cited on page 13.)

[32] Ian H Witten, Zane Bray, Malika Mahoui, and William J Teahan. Using language models for generic entity extraction. In *Proceedings of the ICML Workshop on Text Mining*, 1999. (Cited on page 14.)

[33] Michael Würsch, Giacomo Ghezzi, Matthias Hert, Gerald Reif, and Harald C. Gall. SEON: a pyramid of ontologies for software evolution and its applications. *Computing*, 94(11):857–885, 2012. (Cited on page 9.)

[34] Jianhan Zhu, Victoria S. Uren, and Enrico Motta. ESpotter: Adaptive Named Entity Recognition for Web Browsing. In *WM 2005: Professional Knowledge Management - Experiences and Visions, Contributions to the 3rd Conference Professional Knowledge Management - Experiences and Visions, April 10-13, 2005, Kaiserslautern, Germany*, pages 505–510, 2005. (Cited on page 14.)