Towards the Design of User Friendly Search Engines for Software Projects

Rafaila Grigoriou and Andreas L. Symeonidis

Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki, Thessaloniki, Greece
rafaila@ee.auth.gr, asymeon@eng.auth.gr

Abstract. Current work proposes a linguistic approach for supporting the identification of User requirements and Software Specifications. We introduce an NLP-based tool, PYTHIA, that serves as a search engine capable of handling software engineering terminology, aiming to close the loop between the end-user and the software developer. It is an ontology-based question answering system that employs semantic analysis as well as external (both generic use and domain-specific) dictionaries in order to handle term disambiguation, as posed in user defined queries.

1 Introduction

When developers get down to sketching and designing software, they are equipped with too few tools that could enable reuse of the optimal set of functional requirements and well engineered software modules satisfying these requirements. Were such tools available and this information properly stored, developers would be able to access other Software Engineers' solutions to similar projects and could reuse them as off-the-shelf components, or could adjust them to their own needs.

Taking this argument one step further, one could argue that such a "search-engine" for software projects should be interactive, allowing users to progressively identify the required software constructs, and adaptable, in order to increase its knowledge base. Question Answering (QA) systems could provide the means to realize such search engines, given that they illustrate these types of features.

Towards this direction we have designed and developed PYTHIA (**P**rogrammer's d**Y**namic **TH**ematic **I**nteractive **A**dvisor), a QA tool that can provide guidance through requirements elicitation and design specifications of a software project. Information related to already implemented software projects is stored in an especially-designed ontology scheme and offers engineers the ability to access previous design paradigms, reuse, or even evolve them.

2 Related Work

QA systems can be classified in two major categories, the open-domain and the restricted-domain systems [4]. The former may provide information on any

E. Métais, M. Roche, and M. Teisseire (Eds.): NLDB 2014, LNCS 8455, pp. 164–167, 2014.

topic/domain using any information they can access (on the web or knowledge bases), while the latter answer queries on specific topics/domains, using a focused set of information sources. The incorporation of semantics in QA systems can be performed in various stages of the query processing/query answering process, thus leading to four categories of systems: Semantics-based, Inference-based, Logic-based and Hybrid QA systems.

PYTHIA is an ontology-based, restricted-domain application and falls into the category of hybrid QA systems. Apart from Apple's Siri engine [5], one should also mention AQUA[3], which combines ontologies, logic and NLP technologies to exploit semantically annotated web pages and MOQA[1], which attempts to answer sequences of questions by using a fact repository comprising instances of ontological concepts. AQUAINT[6] employs a model-based approach, based on the idea that relations relevant to a question are best captured by an expressive model of events, while Unger and Cimiano [7] perform compositional meaning construction on the Semantic Web. Finally, Freya[2] combines syntactic parsing with a set of heuristic rules, as well as ontology-based annotations, in order to identify the answer type of input questions.

Although efficient, the above discussed systems cannot cope with the complexity and particularities of the software engineering process life-cycle. The use of a general-purpose lexicon was found inefficient, since most of engineering terms entail semantics and are not handled properly. PYTHIA users have the ability to perform multi-level queries, each one refining search, while they can also enrich the system knowledge base, in case the semantics of a concept are not properly handled. Section 3 discusses the main architectural components of PYTHIA.

3 System Architecture

3.1 PYTHIA Modules

PYTHIA[1] is a web-application that allows users to perform queries either in natural language, or by compiling advanced queries through the corresponding web pages. In both cases, the system is able to deal with term disambiguation with the help of external dictionaries. Information in PYTHIA is stored to and retrieved from a two-ontology scheme[2]: a) *RequirementsOnt*, where functional requirements are represented and stored in an actor-action-system-constraints form, and b) $UMLOnt$, which stores information on UML class diagrams and metadata defining the relation between entities. *RequirementsOnt* and $UMLOnt$ are related through references on functional requirements.

PYTHIA comprises the following subsystems, which interact with each other during the flow of a query:

Spellchecking Subsystem - SPS: It provides similar functionality with Google's *did-you-mean* feature, checking whether the input text is orthographically correct. If not, the user is prompted to select from a number of suggested words,

[1] Available at: http://155.207.18.187:8080/pythia
[2] More information at: http://issel.ee.auth.gr/software-algorithms/

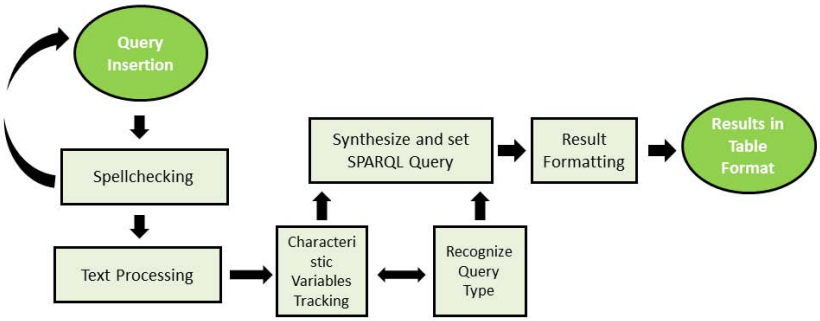

Fig. 1. System workflow diagram

ranked by similarity to the user input. SPS employs the Lucene Spellchecker API.

Text Processing Subsystem - TPS: TPS performs grammar and syntax analysis on the user input. It employs the Stanford Parser API in order to recognize 14 categories of dependencies, useful for the identification of the subject, the verb and the object or copula, as well as prepositional modifiers, noun compound modifiers and adjectival modifiers. Special cases such as passive voice, phrasal verbs and conjunctions are also well-handled.

Characteristic Variables Tracking Subsystem - CVTS: CVTS implements a Lexical Similarity Calculating Algorithm in order to identify the corresponding labels of the entities residing in the ontology, searching among class names and specific instances. CVTS analysis generates the subject, object, property and filter values that will be used for the creation of a SPARQL query to post to the semantic layer. Synonyms are also taken under consideration during the process, in order to increase the possibility of finding the correct relation, using WordNet.

Query Type Recognizing Subsystem - QTRS: QTRS interacts with CVTS, in order to identify the type of the query performed. Three types of queries can be performed: Simple, Dependency and Advanced Queries. Query categories and their properties are further discussed in next subsection. QTRS employs the Apache Jena API.

SPARQL Query Synthesis Subsystem - SQSS: SQSS composes the query based on the QTRS output and posts it to the ontology scheme.

Result Forming Subsystem - RFS: Finally, RFS is responsible for formatting results and generating the final user view. In all result representations, links to lower level information are provided.

3.2 Query Types Supported

`PYTHIA` supports 3 query categories: *Simple*, *Advanced* and *Dependency* queries. The user is confronted with a simple 2-tab interface, and is prompted to select between the *Simple* and *Advanced* query pane. In the former case, the user provides information in free text, while in the latter, the user can "guide" the

query process by selecting the entities to be involved and probably specific query variables.

Simple queries are distinguished into the following subcategories: a) Simple queries in a Subject-Verb-Object (SVO) format, b) Subject-instance queries, c) Object-instance queries and, d) Dependency queries. Advanced queries can be of the following types: a) Advanced queries in SVO format, b) Object-instance advanced queries, c) Advanced queries with keywords instead of properties, iv) Dependency queries and, v) Step-by-step user created dependency queries.

If the subject and/or object entities are not identified as directly related to the ontology, queries are categorised as *Dependency* queries and multiple level SPARQL queries are built getting information from the user. Practically, when `PYTHIA` fails to retrieve any results related to a query, it provides the user with the capability to construct queries step-by-step, in a question answering manner. In all cases that the multiple level queries lead to results, the query is stored to an auxiliary database as a Dependency query. This way the system evolves automatically, enriching the variety of questions it is capable of answering.

4 Conclusions - Future Work

This paper describes the first approach towards the creation of a search engine for developers that want to retrieve information about past projects. `PYTHIA` is capable of replying to specific types of questions and learn from user queries. The presented methodology seems to be appropriate; however, specific actions could improve system's performance. User feedback could provide a useful means for quality improvement. Additionally, the generation of a software engineering lexicon and the replacement of WordNet, could also lead to better search results. Finally, keeping an archive of inserted questions and their results, as most search engines do nowadays, could help the system achieving quick and accurate responses to the most common queries.

References

1. Beale, S., Lavoie, B., McShane, M., Nirenburg, S., Korelsky, T.: Question answering using ontological semantics. In: Proceedings of the 2nd Workshop on Text Meaning and Interpretation, pp. 41–48 (2004)
2. Damljanovic, D., Agatonovic, M., Cunningham, H.: Identification of the question focus: Combining syntactic analysis and ontology-based lookup through the user interaction. In: LREC. Citeseer (2010)
3. Enrico, M.V.-V., Motta, E., Domingue, J.: Aqua: An ontology-driven question answering system, Stanford University. Citeseer (2003)
4. Merkel, A.: Using language models in question answering (2008)
5. Naone, E.: Tr10: Intelligent software assistant (March-April 2009)
6. Sinha, S., Narayanan, S.: Model-based answer selection. In: Proceedings of the AAAI Workshop on Inference for Textual Question Answering (2005)
7. Unger, C., Cimiano, P.: Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In: Muñoz, R., Montoyo, A., Métais, E. (eds.) NLDB 2011. LNCS, vol. 6716, pp. 153–160. Springer, Heidelberg (2011)