Software Knowledge Representation to Understand Software Systems*

Victoria Torres[0000-0002-2039-2174] and Miriam Gil[0000-0002-2987-1825] and Vicente Pelechano[0000-0003-1090-230X]

Universitat Politècnica de València, València 46022, Spain
{vtorres, mgil, pele}@pros.upv.es

Abstract. A software development process involves numerous persons, including customers, domain experts, software engineers, managers, evaluators and certifiers. Together, they produce some software that satisfies its requirements and its quality criteria at a certain point in time. This software contains faults and flaws of different levels of severity and at different phases of its production (specification, design, etc.) so maintenance is needed in order to correct it. Perfective and adaptive maintenance is also needed to cope with changes in the environment or with new requirements, e.g. new functionalities. In this work, we introduce the Persistent Knowledge Monitor (PKM), which is being developed within the DECODER H2020 project for handling (i.e. storing, retrieving, merging and checking for consistency) all kinds of knowledge and information related to a software project. The PKM will be part of a platform capable of taking advantage of all the artefacts available in a software ecosystem, not only the source code, but also its version control system, abstract specifications, informal documents or reports, etc. for representing the software knowledge and improving the workflow of software developers.

Keywords: Persistent Knowledge Monitor, Software engineering, traceability.

1 Introduction

Software maintenance and improvement are very costly and consuming tasks especially when there is an intense use of legacy code or third-party libraries, which usually lack of documentation or when available, it is out-of-date from the current version of the associated piece of software. However, properly performing these maintenance and improvement tasks requires a deep understanding not just of the source code but also of the critical information bound to the code and the process that led to its production.

A key aspect to achieve such deep understanding is to discover knowledge by analyzing all the available artefacts of a given software project. Then, based on the obtained

* This work has been developed with the financial support of the European Union's Horizon 2020 research and innovation programme under grant agreement No. 824231 and the Spanish State Research Agency under the project TIN2017-84094-R and co-financed with ERDF

knowledge, stakeholders can be provided with different views of the system at different levels of abstraction that may be more appropriate to achieve the understanding of the underlying system. However, prior to the creation of such system views, knowledge has to be properly represented according to a well-defined schema or meta-model. Such meta-model must represent, in the most accurate way, all the elements that conform to a software system and all the existing relationships between them. Regarding these relationships, it is important to have a clear understanding at the most fine-grain level, where specific sections or portions of a given artefact (e.g., class x implementation in a java source file) may relate to a different one (e.g., class x definition in a uml class diagram).

In the literature we can find different meta-models targeted to represent the knowledge that can be extracted from software artefacts. These include the Knowledge Discovery Meta-model (KDM) [1] and Abstract Syntax Tree Metamodeling (ASTM) [2] (specifications developed by the OMG ADM task force [3]), FAMIX [4], the Pattern and Abstract-level Description Language (PADL) [5], or the OASIS Static Analysis Results Interchange Format (SARIF) [6]. All these meta-models put their focus on artefacts such as source code, models, and specifications to extract knowledge from the software project. However, in addition to these artefacts, there are other less formal sources that are not usually considered and that can be processed and analyzed to get some extra knowledge about the software project being maintained or improved. These include forum discussions, issue tracker items, reports, etc.

Therefore, taking as reference these meta-models, and considering these less formal sources, in this work we present an overview of the meta-model of the Persistent Knowledge Monitor (PKM), a central infrastructure to store, access, and trace all the persistent data, information and knowledge related to a given software or ecosystem. This PKM is being developed within the DECODER H2020 project[1], whose major objective is to provide powerful tools for developers to get thorough understanding of a given piece of software.

The remainder of the paper is organized as follows. Section 2 identifies the type of sources considered in DECODER to populate the PKM. Section 3 provides an overview over the existing literature found regarding meta-models representing software artefacts. Then, section 4 provides an overview over the PKM meta-model, describing its main components and the relationships among them. Finally, section 5 provides some conclusions and outlines future work.

2 Knowledge Sources to Populate the PKM

One of the major functionalities of the PKM is storing the knowledge generated by the DECODER toolset, toolset targeted to process/analyze the different software project artefacts. Besides this storage functionality, the PKM should also provide the capabilities to allow the DECODER toolset to query, update, and reason over the stored knowledge. Specifically, the information that will be stored in the PKM includes:

[1] https://www.decoder-project.eu/

- Some form of the abstract syntax tree (and concrete trees) related to the source code and the libraries used.
- Some derived or normalized form of the code (after pre-processing, GIMPLE or Generic/Tree internal representations provided by GCC, or CIL representation for Frama-C [7]).
- Some generated or manually written annotations (e.g. in ACSL/ACSL++ for C or C++ code, in JML for Java code).
- Natural language documentation or comments, related to some particular chunk of source code or of a global nature.
- Historical information, extracted from version control systems and bugzillas.
- Information produced by static source code analysis, by optimization passes of compilers, by natural language processing and machine learning techniques.
- Any other relevant information that contributes to enrich the system representation.

Examples of processing/analyzing activities performed by the DECODER toolset are extracting features from source code, annotating code comments and issues with entities, predicates, arguments, etc. Therefore, as Fig. 1 shows, the PKM is expected to interact with several tools, some targeted to process different artefacts to generate knowledge and populate the PKM and others to consume such knowledge and assist stakeholders in their respective tasks within the process lifecycle.

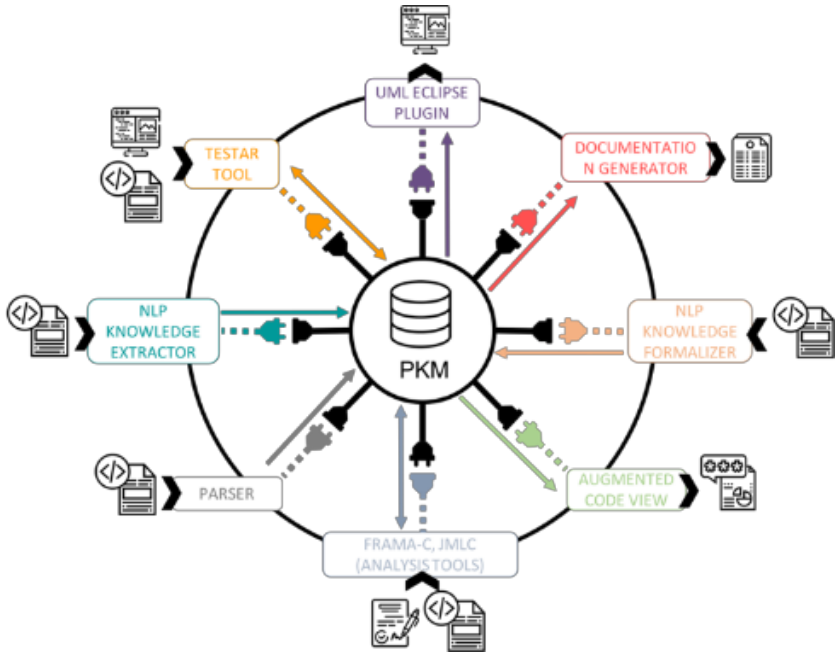

Fig. 1. Interaction between the PKM and tools that generate and/or consume knowledge

3 Meta-models for Software Knowledge Representation

The knowledge extraction process refers to one of the major tasks of the reverse engineering, which was defined by Chikofsky and Cross II in [10] as "the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction". Big efforts have been made in the area of Model-Driven software modernization where several works have been proposed in order to create a common repository structure for representing information about existing software assets. The OMG's Architecture Driven Modernization (ADM) initiative [3] defines a set of standard meta-models which represent the information normally managed in modernization tasks. Specifically, the Knowledge-Discovery Metamodel (KDM) [1] provides the ability to document existing systems, discover reusable components in existing software, support transformations to other languages and MDA, or enable other potential transformations. KDM is partitioned into several packages, each one representing different kinds of software artifacts as entities (e.g., code entities, data entities, UI entities, environment entities). An implementation of this meta-model is provided by MoDisco [11], an Eclipse-based framework that was developed to provide support to the software modernization process. In addition, to better support source code analysis activities, ADM also defined the Abstract Syntax Tree Metamodel (ASTM) metamodel [2], to represent the Abstract Syntax Tree (AST) of any programming language. This model defines a Generic ASTM (GASTM) with definitions that apply to ASTs of most programing languages, and Specialized ASTM (SASTM) with features specific to a single programming language. More recently, other meta-models have been defined to support structured metrics (SSM) [8], or software patterns (SPMS) [9]. Other meta-models focused specifically on the object-oriented languages are FAMIX [4], which also allows representing procedural languages, and the Pattern and Abstract-level Description Language (PADL) [5], which also focus on patterns, allowing the description of motifs. Mainly conceived to detect software defects and vulnerabilities, the OASIS Static Analysis Results Interchange Format (SARIF) [6] defines a standard specification to capture the range of data produced by commonly used static analysis tools.

In DECODER, for the definition of the PKM meta-model we will make use/reference all those existing meta-models when possible. For example, GASTM and FAMIX will be used to define the part of the PKM meta-model where the AST is kept. However, in the PKM we consider other less formal sources of knowledge that are poorly structured, incomplete, and sometimes incorrect. After a process of knowledge extraction, this information will be stored in the PKM.

4 The PKM Meta-model

The PKM provides the representation of a general and specific knowledge about the artefacts of a software project. In order to manage the complexity of the PKM, it is defined by a collection of meta-models according to the categories of the artefacts and

a core package that defines the general knowledge of them. The defined packages are the following (see Fig. 2):

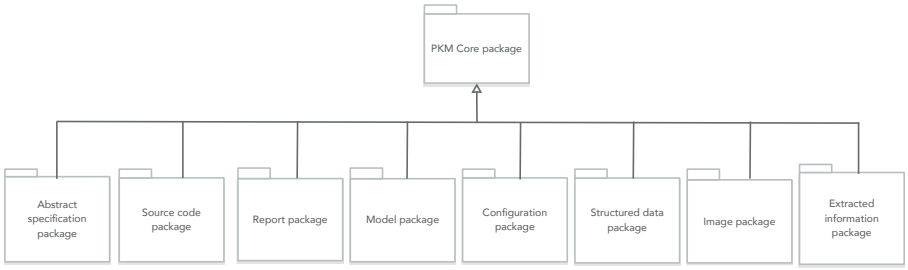

Fig. 2. Organization of the PKM Packages

- *Core package*: it defines the core part of the PKM representing the concept of artefact and its related concepts such as the project use case in which the artefact belongs to, the tools that can manage the artefacts (specification and management tools), the development phases in which artefacts are used during the development process, and the stakeholders that are involved.
- *Abstract specification package*: it defines the meta-model elements of the formal specification describing, by means of pre, post and invariants, the behavior of an associated source code. This abstract specification can be automatically generated or manually written by means of annotations (e.g., in ACSL, ACSL++, JML, etc.).
- *Source code package*: it defines the part of the meta-model that refers to the artefacts that list human-readable instructions written by a programmer with the objective of being executed in a computing device. A source code artefact belongs to one programming language, it relates to a set of referenced libraries and with history data extracted from version control systems and bugzillas.
- *Report package*: it defines the part of the meta-model that represents the artefacts containing a structured content in natural language, related to some particular chunk of source code or of a global nature.
- *Model package*: it defines the part of the meta-model that represents abstract representations of a specific aspect from a given domain (e.g., a uml class model describes the structure – concepts, properties of the concepts, relationships between concepts- of a specific domain).
- *Configuration package*: it defines the meta-model that represents artefacts describing, in plain text, the parameters that define or execute a specific software program.
- *Structured data package*: it defines the meta-model that represents artefacts that store data structures and that are usually used as interchange format.
- *Image package:* it defines the meta-model that describes binary representation of visual information such as drawings, pictures, graphs, etc.
- *Extracted information package*: it defines the meta-model that represents information produced by static source code analysis, by optimization passes of compilers, by natural language processing or by machine learning techniques.

4.1 The PKM Core Package Overview

The PKM Core package, as shown in Fig. 3, is built around the *artefact* concept, which is specialized into the different types of artefacts considered in DECODER use cases, which are *abstract specifications, source code, reports, models, configuration artefacts, structured data,* and *images*.

Artefacts are digital products or documents created during the software development process. It can be presented in different formats (plain text, key-value structures, markup documents), and levels of abstraction (high, medium, and low). Moreover, artefacts can be related to other artefacts with the same (or similar) semantic intention (e.g., a java file may be related to a uml diagram describing a class from a given domain).

An artefact belongs to a *project use case*, which defines a set of artefacts of different nature (source code, documents written in natural language, configuration files, etc.) organized (or not) according to a logical structure (e.g., directories) and provided (or not) as a compressed file. These artefacts are consumed or created during the project development and maintenance process.

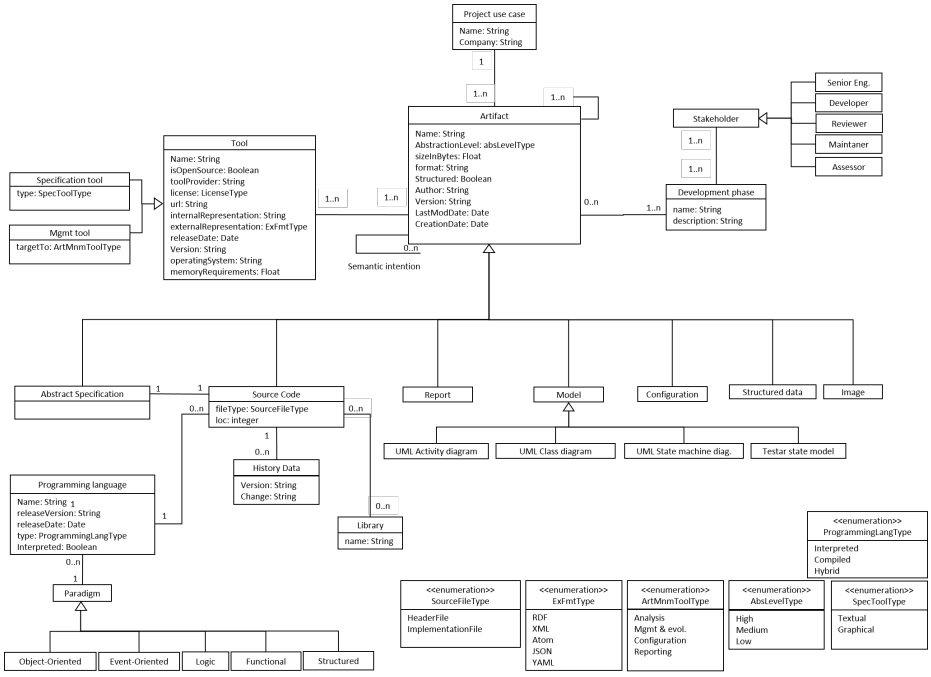

Fig. 3. PKM Core Metamodel Package

Artefacts are managed by *tools* that are used by any stakeholder to analyze, transform, refine, etc. them and produce new or modified artefacts. Tools can be categorized into *specification tools*, which are tools that allow to create, modify, and refine artefacts, and *management tools*, which are tools that assist/guide the stakeholder in the

task of analyzing, managing, evolving, and configuring a specific tool as well as tools that act as back-ends to the previous tools categories to generate various kinds of reports.

Artefacts are related to *development phases* in which they are used during the development process, i.e., requirements, design, implementation, testing, deployment, maintenance. Finally, in each development phase, different *stakeholders* take part to develop a specific task within the project. These stakeholders can be senior engineers, developers, reviewers, maintainers, or assessors.

5 Conclusions and future work

As we have pointed out, the PKM has been built within the DECODER project with the goal of store, access, and trace all the persistent data, information and knowledge related to a given software project or ecosystem. This knowledge will be useful for the different actors involved during the life span of a software, especially new persons, to keep project information and knowledge in the most accessible and unambiguous way. This living repository can be queried and enriched by the actors involved in the project, in order to maintain consistency and keep the most updated and precise information about it.

This work constitutes a first step in the formalization process of the PKM meta-model, which will be in charge of gathering all the data, information and knowledge that can be extracted from a given software project. As future work such meta-model will be implemented as a database having in mind that the potential and diverse processing tools that may interact with the PKM demands for a dynamic and flexible data schema that could be modified according to the new interaction needs. Such flexibility would allow extending the schema with new types based on the processing results produced by new interacting tools. For this reason, we are planning to use JSON as the interchange mechanism between tools and the PKM. Once complete and implemented, the PKM will be validated empirically with four different use cases proposed in the DECODER H2020 project. These refer to OS drivers provided by SYSGO (https://www.sysgo.com/), the openCV library commonly used by Tree Technology (http://www.treetk.com/es/index.html) in its developments, general purpose Java code hosted in the OW2 (https://www.ow2.org) open-source software community, and My-Thai-Star showcase application, developed by CAPGEMINI (https://www.capgemini.com/es-es/service/agile-delivery-center-valencia/).

In addition, the knowledge gathered in the PKM should be also used along the different stages of the software lifecycle to improve and assist stakeholders in their respective tasks. Fig. 4 provides an overview over the different roles involved in DECODER as well as their interaction with the PKM. First, developers will feed the PKM with the bulk code and documentation of the use cases where they are involved. Second, reviewers will write correct properties in ACSL, ACSL++ (the extension of ACSL for C++) or JML with invariants and behaviors implicitly connected to a model based on abstract state machines. Finally, maintainers will do the work of reviewing and taking decisions on how to resolve inconsistencies. An online traceability matrix will be used to control

the consistency of these elements and to help deciding when the software becomes ready for manufacturing and for being reused.

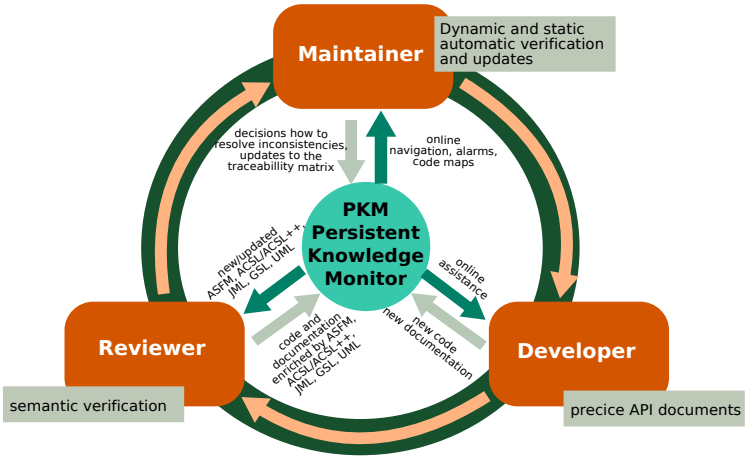

Fig. 4. An overview of the development life cycle

References

1. Object Management Group, Inc. (2012) Knowledge Discovery Meta- model (KDM). [Online]. Available: http://www.omg.org/technology/ kdm/index.htm
2. Architecture-Driven Modernization: Abstract Syntax Tree Metamodel (ASTM), OMG document formal/2011-01-05, OMG, Jan. 2011. [Online]. Available: http://www.omg.org/spec/ASTM
3. ADM initiative website. http://adm.omg.org. Accessed 5 July 2019
4. S. Tichelaar, S. Ducasse and S. Demeyer, "FAMIX and XMI," Proceedings Seventh Working Conference on Reverse Engineering, Brisbane, Queensland, Australia, 2000, pp. 296-298. doi: 10.1109/WCRE.2000.891485
5. Y.G. Guéhéneuc, “Ptidej: promoting patterns with patterns”, in 1st ECOOP Workshop on Building Systems using Patterns, pp. 1-9. Springer, Heidelberg (2005)
6. Static Analysis Results Interchange Format (SARIF) website. https://www.oasis-open.org/committees/sarif. Accessed 9 July 2019
7. Frama-C software analyzer website. https://frama-c.com/. Accessed 9 July 2019
8. Structured Metrics Meta-model (SMM), OMG document formal/2016- 04-04, OMG, Apr. 2016. [Online]. Available: http://www.omg.org/spec/ SMM/
9. Structured Patterns Metamodel Standard (SPMS), OMG document ptc/16-03-13, OMG, Mar. 2016. [Online]. Available: http://www. omg.org/spec/SPMS/1.1
10. Elliot J. Chikofsky, James H. Cross II: Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software 7(1): 13-17 (1990)
11. H. Brunelière, J. Cabot, G. Dupé, F. Madiot, “MoDisco: A model driven reverse engineering framework”, in Information & Software Technology 56(8): 1012-1032 (2014)