Using Ontologies For Software Documentation

Werner Klieber[1], Vedran Sabol[1], Roman Kern[1],
Markus Muhr[1], Michael Granitzer[1]

[1]Know-Center, Inffeldgasse 21a
8010 Graz, Austria
{wklieber, vsabol, rkern, mmuhr, mgrani}@know-center.at

Abstract. High quality software documentation is a substantial issue to understand software systems. Shorter time-to-market software cycles increase the importance of automatism to keep the documentation up to date. In this paper, we describe the automatic support of the software documentation process using a social semantic software approach. Therefore, we introduce a software documentation ontology as underlying knowledge base. The defined ontology is populated automatically through analyzing source code, documentation and code executions and made accessible to developers via a wiki. The wiki enforces collaboration and allows correcting analysis errors. We demonstrate, that the use of social semantic systems can support software documentation processes efficiently.

Keywords: Documentation, Ontologies

1 Introduction

Software development of large project is a complex task. High quality documentation is essential to understand the intention of a software system. Good documentation provides multiple complementary views on the software system [7]. The documentation has to reflect the developer's view on how the system works and provides logical views on the system for users how to consume the provided functionality.

In the last decades a lot of effort has been done in formalizing the documentation process. However there is still a lack of coherent documentation [5]. An ongoing process in software engineering is to reduce development cycles to shorten time-to-market. Agile software development methods like SCRUM [10] enforces software evolution in "sprint" periods taking typically one till four weeks. This dynamic process requires updating software documentation frequently. Hence, automatic support in software documentation increase accuracy and reduces workload on developers.

Software documentation is closely related to knowledge management and requires collecting knowledge from various sources and representing it in a formalized way. Having a standardized and flexible data environment of the gathered information enables an unproblematic communication among each tool involved in the documentation process. Ontologies are well investigated and an accepted approach in formalizing shared concepts making them understandable by both, humans and computers. Several approaches exist which use ontologies for supporting the software engineering process. Cortellessa [4] introduces a software performance ontology to formalize performance indicators like response time or utilization. The ontology provides alternate views on a software system beside common used engineering documentation methodologies like UML. Further, formalizing information in standardized formats like RDF and ontologies standardises the exchange of information among information systems.

Wikis came up recently as another effective collaboration and knowledge management tool due to their simple usage. Wikis are used in companies in many cases for software documentation activities [8]. The survey reported three types of benefits of wikis: enhanced reputation for users, making work easier and helping organizations improving their processes. However, in contrast to ontologies wikis are hardly machine readable and formalize knowledge on a rather low degree.

In the presented work we combine the use of ontologies to model software documentation activities with wikis to support developer collaboration. We focus on automatic tools to fill the ontologies and provide content for a wiki system to minimize manual effort in maintaining the documentation. Therefore, we developed a software ontology to model the structure of program code up to a detail level of single operations. Program executions are monitored to populate the ontology. The populated ontology is further analyzed via Process mining to detect meaningful, potential useful process patterns [1]. Using ontologies as communication base allow an overall transboundery concept sharing between tool filling the data store and tools accessing the data.

The paper is structured as following. Chapter 2 introduces the software ontology, the ontology modeling process we use and discusses the design in detail. In Chapter 3 we describe our test scenario to prove the practicability of ontologies for an automated documentation process. In our domain of knowledge discovery we developed a knowledge discovery framework to reuse algorithms within projects. We describe the usage of the ontology based documentation process introducing some documentation scenarios for our knowledge discovery framework.

2 Design of a software ontology

A documentation process consists of three steps: gathering documentation data, resolving correlations and generating reports [5]. In our approach we focus on automatic tools to support this process via formal, well defined ontologies. Ontologies are well suited for defining vocabularies and concepts of any domain to avoid communication obscurities. Using ontologies for specifying data and data structures instead of data description languages like XML facilitate a more comprehensive system specification. The system intention and relationship of involved components can be expressed explicitly. In this section we introduce the ontology modeling approach used, and introduce the main aspects of the ontology.

2.1 Ontology modeling process

The ontology has been created considering the seven steps proposed by Noy and McGuiness [9]. The domain and scope of the ontology has been defined in a combination of a bottom up and top down approach. A bottom up modeling phase investigate existing system characteristics and identifies useful pattern for a given domain model. A top down modeling phase investigates application scenarios and extracts common patterns for the given domain.

In the bottom up phase the underlying technical characteristics has been considered like the structure of source code or the nature of event logs in the process mining domain. The top down approach was used to model concepts useful for generating documentation reports by querying needed information using SPARQL queries.

To keep the vocabularies and concepts of the ontology consistent, the ontology has been developed from scratch. In a final step an extra ontology has been derived from the base ontology to fill it with specific instances applying the environment of our company, for instance all employees.

2.2 Overview

Figure 1 shows the main concepts of the ontology[1]. In the ontology all aspects are grouped together using a central *project* concept. To reflect dependencies among projects, the *project* concept itself can have dependencies to other projects or modules. This *project* part of the ontology is based on the project model approach from the maven tool[2]. Maven is a project management tool from the Apache Software foundation organizing projects in a project object model (POM). In Maven, software projects are defined by a set of these xml configuration files. Dependencies among projects are defined within these POM files using a parent-child relationship. Software projects are typically maintained by using a bulk of specialized frameworks like revision control systems, bug tracking systems, automated build and release management systems, reporting tools and so on. The concept of *external resources* is used to link to this resources within the ontology. The concept of a *Change Log* is used to provide an ontology data change history affected by tools writing into the ontology. This provides a simple mechanism for tools verifying pre and post conditions.

[1] The ontology is available for downloaded from our web site:
http://www.know-center.at/ontologies/2009/2/software-project.owl.

[2] Maven project management tool: http://maven.apache.org/

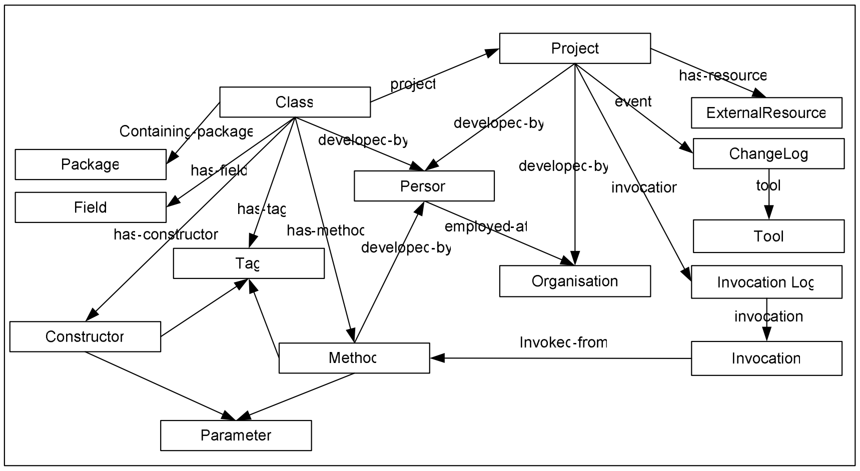


Figure 1 Overview figure of the software ontology showing the most relevant concepts and properties.

2.3 Detailed description of the ontology

Beside common metadata concepts introduced in the overview section, the ontology is structured into three aspects useful for generating various views on software. The views are motivated by recurring documentation needs from developers. These needs are:

- Static source code information is modeled to import data on a source code level into the ontology.
- Dynamic program information is modeled to import runtime information like invocation sequences of program operations into the ontology.
- Human actors linking the technical information to developers.

Static Source Code: The static source code information forms the main part for analyzing. Selective classes, operations and source code comments can be used to represent the current state of an API. The ontology models the source code API specifications reflecting characteristics of object oriented programming languages. In particular, this ontology part encapsulates the relationship of classes, methods, variables and package structure of source code. Additional code metrics like lines of code of a method can be added here in future. All of these source code elements can be tagged. The concept of "tagging" resources can be used by developers to add further information about intentions of relevant source code pieces directly in the source code. Querying these tags later on during report generation allows more specialized views on the source code. For instance a tag "workflow" is used to mark useful test code to generate invocation events described more detailed in the usage scenario section. In most state of the art generated source code approaches for marking input and output parameter descriptions are used to provide this information for reports. For instance in the Java programming language the “@” prefix is used to mark tags in program code. Using these tags in queries allows filtering for useful criteria in reports like authors or date.

The Ontology distinguishes between productive source code and test code. The productive source code contains all source code dealing with the program execution logic. The test code contains all tests written by developers. Besides using test code as indicator for test coverage of program functionality, selective tests are good usage examples for application developers. Using the test code itself as usage examples ensures the code examples are up to date and working.

Inspecting test code allows a more indebt knowledge about a program. Besides testing functional behavior of a program several tests are written by developers as usage examples.

Dynamic Program Information: Concepts in the ontology for dynamic program information are used to gather software usage information. Dynamic program information is gathered during program execution runtime by logging operation calls. Analyzing this information allows more in depth knowledge about

- In which order operations needs to be executed to reach certain goals.
- Analyzing the operation execution time is a useful indicator for performance
- Analyzing operation errors is a useful stability indicator.
- Analyzing the operations usage count: the more frequently an operation is called might be an indicator about its importance.
- Analyzing the amount of times an operation is called within tests is a simple indicator for test coverage.

The ontology models this information as operation call invocation events. Approaches in the business processes mining research can be used for analyzing these events. In future versions of the ontology further types of events may be introduced, for instance logging program API changes over time. The concepts belonging to the ontology’s dynamic part are based on the data format used by the business process mining tool ProM[5]. The ProM framework provides a large set of state of the art process mining algorithms integrated as plug-in that can be used by an intuitive graphical user interface. The ontology groups several operation invocations by workflow usage cases. Three type of operation calls are distinguished

- Begin: this event occurs when initiating an operation call
- End: this event indicates finishing an operation call.
- Error: This event indicates operation calls causing exceptions.

Each invocation event contains a timestamp when it occurred, the memory consumption, the originator of the event and a process instance name.

Technical Information: The ontology allows to assigns human actors to operations and projects. In an organizational context it is crucial knowing which employees are involved in projects or are responsible for certain source code pieces. The ontology allows assigning persons to relevant topics of the ontology.

3 Usage scenarios

The applicability of the ontology has been tested by importing all source code projects our knowledge discovery framework “KnowMiner” consists of. Knowledge discovery is the non trivial process of finding previously unknown and potentially useful knowledge in unstructured data. It is an interdisciplinary research area covering topics from information retrieval, machine learning, logic/inference and visualization [11]. We chose a service oriented architecture for our framework enabling different teams – each experts on their research area – to work on isolated components. Whereas special effort has been undertaken in providing a simple to use API to application developers, an open issue for new developers is how to assemble their workflows [12].

The KnowMiner framework uses the Maven project management tool from the Apache software foundation to define all of its depended modules and third party libraries. These Maven project files are parsed, converted to the RDF data in the ontology schema and imported to the triple store. The conversion is done using an adapted version of the java2rdf tool[3]. The KnowMiner Framework contains about 400.000 lines of code, separated in 74 modules and 32 test modules and 65 third party libraries. Next the invocation events are imported using a separate program converting usage logs of the main API into RDF. For our experiment we used the TDB jena triple store[4]. Jena TDB is a file based storage backend for the jena semantic web backend. The next step normalizes the author names. This may become a non-trivial task [3], however in our test scenario only 21 persons involved in the software development process have to be considered. Firstly, all persons under consideration are imported as basic data. A simple lookup mechanism is used to unmask all alias names developers are using during documentation of their source code. Typically mail addresses or users login names are used as author names. Finally the database consists of about 4.5 million triples. The next sections introduce some usages scenarios finding relevant information.

3.1 Process mining using invocation logs

When developing our knowledge discovery framework we took care in providing a simple to use API with focus on limiting the amount of operations. When application developers first come into contact with the API it is a challenge to figuring out how to assemble workflows using API methods. State diagrams showing the allowed transitions of operation calls help in getting an overview about the operating mode of a system. However, especially in agile software development processes, developers do not draw state diagrams manually.

The invocation logs of API calls can be used as input for process mining tools to generate automatic state diagrams. In our experiment we used the invocation logs generated by executing some unit tests from our knowledge discovery framework. The intention is to get useful workflows from persons who know best how to use the

[3] Java RDF documentation tool: http://simile.mit.edu/wiki/Java_RDFizer
[4] TDB – A SPARQL database for Jena: http://jena.sourceforge.net/TDB/

system. Furthermore, no extra effort is needed to get this invocation information. These logs are stored in a triples store using the software ontology. First, a SPARQL query is used to get the invocation logs from the RDF database. The result is converted into the MXML format used by the process mining tool ProM[5].

First results showed that using the complete invocation events from all unit tests contain too much noise to detect meaningful workflows. One reason is that many tests perform no usable workflows with regard of meaning full API usage. For example some tests call an operation several times to ensure the correct results are delivered each time. Furthermore it is difficult to detect the start and end sequences of a workflow correctly. Some developers reuse test data among their tests for performance reasons. Using each unit test isolated as workflow will therefore result in partially invalid workflows. Therefore we chose only tests marked as usage examples by developers. The event log for the ProM tool contains 130 events grouped in 5 processes with 46 cases in 15 operations. Figure 2 shows a state diagram of our knowledge discovery framework generated with alpha++ algorithm of the PromM framework [2]. The generated diagram provides an overview of our preprocessing workflow, illustrating the main pattern of how to use it. For instance, first a configuration must be added to specify domain specific settings so the framework can work. Next the importData operation can be used as data source to bring some operational data into the system. After performing algorithmic tasks at the end of a workflow the system needs to be clean up be freeing no longer needed resources.

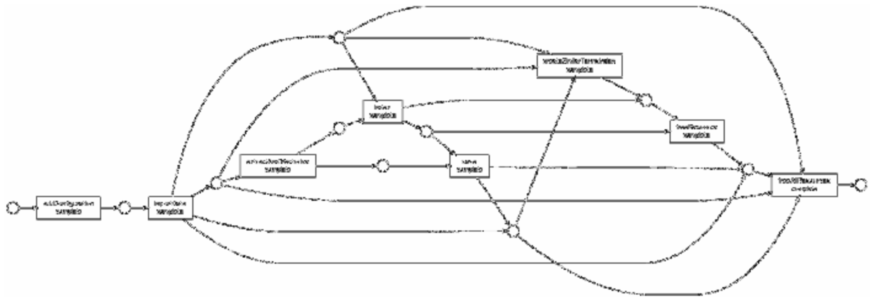


Figure 2 Sample state diagram created using the alpha algorithm++ from the ProM Framework

3.2 Overview charts

A further example illustrating the usage of the ontology is generating simple overview charts. When developers start working with the KnowMiner framework, they start with simple workflows, make them running and next add some further functionality. When adding a further functionality a common question is the effect on the execution time of the whole workflow. Figure 3a shows an overview diagram of

[5] The ProM Framework: http://prom.win.tue.nl/tools/prom/

the average execution time of each method recorded in the invocation events. This diagram helps developers to estimate performance issues. Mostly they start firstly with a simple information retrieval task: import some data and store it in an index to make the data searchable. Next some extraction tasks are included, for instance generating vectors so the data becomes comparable, for instance to calculate similarities between text documents. According to the diagram in figure 3a this extraction task takes much more time than the other tasks. Sometimes application developers are insecure about the long execution time when using the extract operation the first time. A look on the diagram helps them to understand that this is a technical issue.

Such overview diagrams are furthermore useful to find out which operations are good candidates for optimizing.

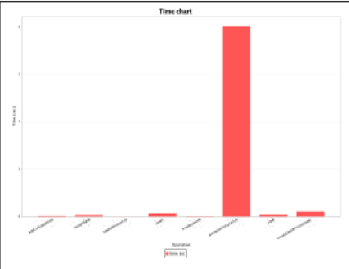
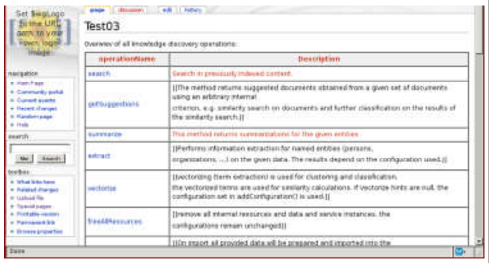


Figure a



Test03

Overview of all knowledge discovery operations:

operationName	Description
search	Search in previously indexed content.
getSuggestions	[[The method returns suggested documents obtained from a given set of documents using an arbitrary internal criterion, e.g. similarity search on documents and further classification on the results of the similarity search.]]
summarize	This method returns summarizations for the given entities.
extract	[[Performs information extraction for named entities (persons, organizations, ...) on the given data. The results depend on the configuration used.]]
vectorize	[[vectorizing (term extraction) is used for clustering and classification. the vectorized terms are used for similarity calculations. If vectorize hints are null, the configuration set in addConfiguration() is used.]]
freeAllResources	[[remove all internal resources and data and service instances. the configurations remain unchanged]]
	[[On import all provided data will be prepared and imported into the

Figure b

Figure 3 Figure 3a illustrates sample diagram showing the average execution time of an operation in seconds. Figure 3b illustrates presenting RDF query results in a wiki environment.

3.3 Web front-end

A further example shows how to integrate the information in a wiki environment. Wikis are used frequently to provide information on web pages that are easily producible and maintainable. Presenting some software project information gathered from various sources allows one single point of access for developers to get needed information. Using automatic generated content to show in a wiki simplifies the report generation and reusing the data on other pages. Changes occuring when a software project evolves will show up automatically in the wiki pages. Figure 3b shows a screenshot of the API from our knowledge discovery framework. It lists all operations of the api and the source code documentation provided by the developers. In the technical realization the media wiki framework is used. An RDF plugin has been developed allowing to send SPARQL queries embedded in a wiki page to the RDF backend.

Source code documentation tends to be very technical and hard to understand without the local context in the source code where it was written. Showing the documentation in a web front-end where users can give feedback to developers about the understandability and accuracy of their documentation can help to improve the source code documentation quality.

4. Conclusion and outlook

In this paper we presented a software ontology to model concepts occurring in the software documentation process. Using a unified, ontology based data format for all tools involved in the automatic documentation process helps avoiding impreciseness in workflows and enables machine understandable specification of data and operations. In our usage examples we demonstrated the feasibility of using an RDF triple store, ontologies and SPARQL queries to support an automatic documentation generation process. Using generic data formats like RDF triples provides a flexible data characteristic adjustment on software evolvements processes. Further, those standardized formats allow in depth analysis of code relationships and their presentation in collaboration enhancing tools like Wikis.

However SPARQL still misses some needful features complicating the usage. For instance aggregator functions to generate sums on query results commonly needed for report generation are missing. In a next step we plan to import larger source code projects from the open source community for analyzing documentation cultures in opern source software development environments.

Acknowledgment. The Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Federal Ministry of Transport, Innovation and Technology, the Austrian Federal Ministry of Economy, Family and Youth and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

5. References

1. A. Rozinat, R.S. Mans, M. Song, and W.M.P. van der Aalst, Discovering Simulation Models, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
2. A.K.A de Medeiros, B.F. van Dongen, W. v. d. A. & Weijters, A. (2004), Process Mining: Extending the alpha-algorithm to Mine Short Loops, *in* BETA Working Paper Series, WP 113, Eindhoven.
3. Bird, C.; Gourley, A.; Devanbu, P.; Gertz, M. & Swaminathan, A. (2006), Mining email social networks, in 'MSR '06: Proceedings of the 2006 international workshop on Mining software repositories', ACM, New York, NY, USA, pp. 137--143
4. Cortellessa, V. (2005), How far are we from the definition of a common software performance ontology?, *in* 'WOSP '05: Proceedings of the 5th international workshop on Software and performance', ACM, New York, NY, USA, pp. 195--204.
5. Hartmann, J.; Huang, S. & Tilley, S. (2001), Documenting software systems with views II: an integrated approach based on XML, *in* 'SIGDOC '01: Proceedings of the 19th annual international conference on Computer documentation', ACM, New York, NY, USA, pp. 237--246.
6. http://jena.hpl.hp.com/wiki/TDB, visited April 2009.
7. Huang, S. & Tilley, S. (2003), Towards a documentation maturity model, *in* 'SIGDOC '03: Proceedings of the 21st annual international conference on Documentation', ACM, New York, NY, USA, pp. 93--99.
8. Majchrzak Ann, Wagner C. & Ystes Dave (2006), Corporate wiki users: results of a survey, in 'WikiSym '06: Proceedings of the 2006 international symposium on Wikis', ACM, New York, NY, USA, pp. 99--104
9. N.F. Noy, D.L. McGuinness, Ontology Development 101: A Guide to Creating Your First Ontology, Stanford University, 2000.
10. Schwaber, K (2004) Agile Project Management with
Scrum. Book: Microsoft Press ISBN 9780735619937

11. Weiss, S.; Indurkhya, N.; Zhang, T. & Damerau, F. Text Mining: Predictive Methods for Analyzing Unstructured Information Springer, 2004
12. Werner Klieber et al., Knowledge discovery using the KnowMiner framework, Proceedings of the IADIS International Conference Information Systems 2009, Barcelona, Spain, February 2009