A Unified Ontology-Based Process Model for Software Maintenance and Comprehension

Juergen Rilling[1], Yonggang Zhang[1], Wen Jun Meng[1], René Witte[1], Volker Haarslev[1], Philippe Charland[2]

Department of Computer Science[1] and Software Engineering
Concordia University, Montreal, Canada
{rilling, yongg_zh, w_meng, rwitte, haarslev}@cse.concordia.ca

System of Systems Section[2]
Defence R&D Canada Valcartier,
Val-Bélair, Canada
philippe.charland@drdc-rddc.gc.ca

Abstract. In this paper, we present a formal process model to support the comprehension and maintenance of software systems. The model provides a formal ontological representation that supports the use of reasoning services across different knowledge resources. In the presented approach, we employ our Description Logic knowledge base to support the maintenance process management, as well as detailed analyses among resources, e.g., the traceability between various software artifacts. The resulting unified process model provides users with active guidance in selecting and utilizing these resources that are context-sensitive to a particular comprehension task. We illustrate both, the technical foundation based on our existing SOUND environment, as well as the general objectives and goals of our process model.

Keywords: Software maintenance, process modeling, ontological reasoning, software comprehension, traceability, text mining.

1. Introduction and Motivation

Software maintenance is a multi-dimensional problem space that creates an ongoing challenge for both the research community and tool developers. These maintenance challenges are caused in particular by the variations and interrelationships among software artifacts, knowledge resources, and maintenance tasks [3,20,22]. Existing solutions [10,20] that address aspects of these challenges are commonly not integrated with each other, due to a lack of integration standards or difficulties to share services and/or knowledge among them. The situation is further complicated by the non-existence of formal process models to create a representation that describes the interactions and relationships among these artifacts and resources.

There has been little work in examining how these resources work together for end users [13,20] and how they can collaboratively support a specific program maintenance task. Maintainers are often left with no guidance on how to complete a particu-

lar task within a given context. Our research addresses this lack of context sensitivity by introducing a formal process model that stresses an active approach to guide software maintainers during maintenance tasks. The process model, its basic elements and their major inter-relations are all formally modeled by an ontology based on Description Logic (DL) [2]. The process behavior is modeled by an interactive process metaphor. Our approach differs from existing work on comprehension models [3], tool integration [11, 20] and task-specific process models [8,19,20] in several aspects:

1. A formal software maintenance process model based on an ontological representation to integrate different knowledge resources and artifacts.
2. An open environment to support the introduction of new concepts and their relationships, as well as enriching the existing ontology with newly gained knowledge or resources.
3. The ability to reason about information in the ontological representation to allow for an active and context-sensitive guidance during the maintenance process.
4. Analysis of relationships among resources, e.g., the traceability between artifacts.

The process model itself is motivated by approaches used in other application domains, like Internet search engines (e.g., Google[1]) or online shopping sites (e.g., Amazon[2]). Common to these applications is that they utilize different information resources to provide an active, typically context-sensitive user feedback that identifies resources and information relevant to a user's specific needs. The challenge in applying similar models in software maintenance goes beyond the synthesis of information and knowledge resources. There is a need to provide a formal meta-model to enable reasoning about the potential steps and resources involved in a maintenance process.

For example, a maintainer, while performing a comprehension task, often utilizes and interacts with various tools (parsers, debuggers, source code analyzers, etc.). These tool interactions are a result of both, the interrelationships among artifacts required/delivered by these tools and the specific techniques needed to complete a particular task. Identifying these often transitive relationships among information resources becomes a major challenge. Within our approach, we support automated reasoning across these different information resources (e.g., domain knowledge, documents, user expertise, software, etc.) to resolve transitive relationships. Furthermore, our model can be applied to analyze and re-establish traceability links among the various resources in the knowledge base [1].

From a more pragmatic viewpoint, process models have to be able to adapt to ever changing environments and information resources to be used as part of the process itself. In our approach, we address this problem by providing a uniform ontological representation that can be both extended and enriched to represent any newly gained knowledge or change in the information resource(s). This knowledge will also become an integrated part of the process that can be further utilized and reasoned on.

The remainder of the article is organized as follows. The relevant research background is introduced in Section 2. Section 3 describes in detail the context-driven program comprehension process model, followed in Section 4 by its implementation and validation. Discussions and future work are presented in Section 5.

---

[1] www.google.com

[2] www.amazon.com

2. Background

Historically, software lifecycle models and processes have focused on the software development cycle. However, with much of a system's operational lifetime cost occurring during the maintenance phase, this should be reflected in both the development practices and process models supporting maintenance activities. It is generally accepted that even for more specific maintenance task instances (e.g., program comprehension, architectural recovery), a fully-automated process is not feasible [11]. Furthermore, existing models share the following common challenges:

- Existing knowledge resources (e.g., user expertise, source code artifacts, tools) are used to construct mental models. However, without a formal representation, these process models lack uniform resource integration and the ability to infer additional knowledge.
- Limited knowledge management that allows the extension and integration of newly gained resource and knowledge.
- These models provide typically only general descriptions of the steps involved in a process and lack guidelines on how to complete these steps within a given context (concrete software maintenance task and available knowledge resources).

In our approach, we provide a formal representation that integrates these information resources and allows reasoning and knowledge management across them. Furthermore, we address the issue of context-sensitive support, i.e., providing the maintainer with guidance on the use of the different information resources while accomplishing a particular task.

Research in cognitive science suggests that mental models may take many forms, but the content normally constitutes an ontology [8]. Ontologies are often used as a formal, explicit way of specifying the concepts and relationships in a domain of understanding [2]. They are typically specified using the standard ontology language, Description Logics (DL), as a knowledge representation formalism.

DL is also a major foundation of the recently introduced Web Ontology Language (OWL) recommended by the W3C[3]. DL represents domain knowledge by first defining relevant concepts (sometimes called classes or TBox) of the domain and then using these concepts to specify properties of individuals (also called instances or ABox) occurring in the domain. Basic elements of DL are atomic concepts and atomic roles, which correspond to unary predicates and binary predicates in First Order Logic. Complex concepts are then defined by combining basic elements with several concept constructors.

Having DL as the specification language for a formal ontology enables the use of reasoning services provided by DL-based knowledge representation systems. The Racer system [7] is an ontology reasoner that has been highly optimized to support very expressive DLs. Typical services provided by Racer include terminology inferences (e.g., concept consistency, subsumption, classification, and ontology consistency) and instances reasoning (e.g., instance checking, instance retrieval, tuple retrieval, and instance realization). For a more detailed coverage of DLs and Racer, we refer the reader to [2,7].

[3] available online at http://www.w3.org/TR/owl-ref

3. Modeling a Software Maintenance Process

A model is essentially an abstraction of a real and conceptually complex system that is designed to display significant features and characteristics of the system, which one wishes to study, predict, modify or control [13]. In our approach, the software maintenance process model is a formal description that represents the relevant information resources and their interactions (Fig 1).

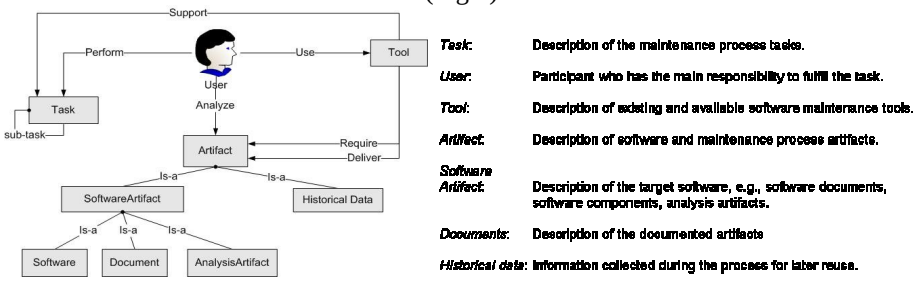

Fig. 1. Comprehension Process Meta-Model

In what follows, we describe in general: (1) the ontological representation used to model the information resources, (2) the ontology population and traceability among ontologies, and (3) the maintenance process and its management.

3.1 An Ontological Software Maintenance Process Model

Through the use of ontologies and DL, we formally model the major information resources used in software maintenance. The benefits of using a DL-based ontology as a means to model the structure of our process model are as follows:

Knowledge acquisition and management. As mentioned previously, program comprehension is a multifaceted and dynamic activity involving different resources to enhance the current knowledge about a system. Consequently, any comprehension process model has to reflect and model both the knowledge acquisition and use of the newly gained knowledge. The ontological representation provides us with the ability to add newly learned concepts and relationships, as well as new instances of these to the ontological representation. This extendibility enables our process model not only to be constructed in an incremental way, but also to reflect more closely the iterative knowledge acquisition behavior used to create a mental model as part of human cognition of a software system [3,8]. It is not realistic to expect all these sources to share a single, consistent view within a comprehension task. Rather, we expect disagreements between individual users and tools during an analysis. In our approach, we explicitly model those different views using a representational model that attributes information to (nested) contexts using so-called *viewpoints.*

An elegant model for managing (possibly conflicting) information from different sources has been proposed by [18]: Knowledge is structured into *viewpoints* and *topics*. Viewpoints are environments that represent a particular point of view (e.g., information stemming from a particular *tool* or entered by a *user*). Topics are environ-

ments that contain knowledge that is relevant to a given subject (e.g., design patterns, architectural recovery). These environments are *nested* within each other: viewpoints can contain either other viewpoints or topics. A topic can contain knowledge pertaining to its subject, but also other viewpoints, e.g., when the subject is another user. These viewpoints create *spaces* that allow consistency to be maintained within a topic or a viewpoint, but at the same time, conflicting information about the same topic can be stored in another viewpoint. Therefore, knowledge can be collected while attributing it to its source, without having to decide on a "correct" set of information, thereby avoiding losing information prematurely. Viewpoints can be *constructed* as well as *destructed* through the processes of *ascription* and *percolation*. Ascription allows incorporating knowledge from other viewpoints (users, tools) unless there is already conflicting information on the same topic. Percolation is introduced for the deconstruction of nested knowledge.

Reasoning. Having DL as a specification language for a formal ontology enables the use of reasoning services provided by DL-based knowledge representation systems, by inferring knowledge through transitive closure across different ontologies. The DL-based ontology and reasoning services form the basis for both, the knowledge integration and retrieval used in our process model.

Building a formal ontology for software maintenance requires an analysis of the concepts and relations of the discourse domain. In particular, the outlined process model must be supported by the structure and content of the ontological knowledge base. Our approach here is twofold: We (1) created sub-ontologies for each of the discourse domains, like tasks, software, documents, and tools (Fig. 1); and (2) link them via a number of shared high-level concepts, like *artifact*, *task*, *or tool*, which have been modeled akin to a (simple) upper level ontology [16].

Having different knowledge resources modeled as ontologies allows us to link instances from these knowledge resources using existing approaches from the field of ontology alignment [17]. Ontology alignment techniques try to align ontological information from different sources on conceptual and/or instance levels. Since our subontologies share many concepts from the programming language domain, such as `Class` or `Method`, the problem of conceptual alignment has been minimized. This research therefore focuses more on matching instances that have been discovered both from source code analysis and text mining.

3.2 Ontological Representation for Software Artifacts

Software artifacts such as source code and documentation typically contain rich structural and semantic information. Providing uniform ontological representations for various software artifacts enables us to utilize semantic information conveyed by them and to establish their traceability links at a semantic level (Fig. 2b). In this section, we introduce our SOUND program comprehension environment [22], which was developed to establish the technical foundation for our ontological software maintenance process model.

The SOUND environment facilitates software maintainers in both discovering (new) concepts and relations within a software system, as well as automatically inferring implicit relations among different artifacts (Fig. 2a and 2b).

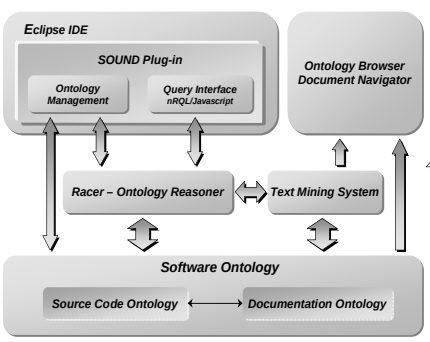

Fig. 2a Overview of SOUND Environment

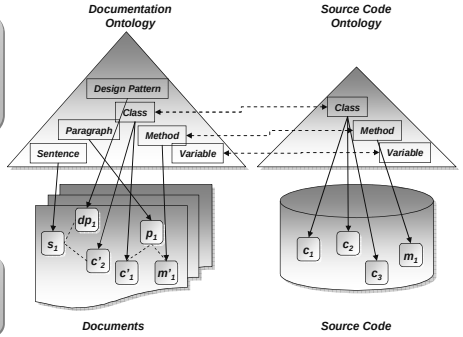

Fig.2b Linking Code and Documentation

Instances of concepts and roles in the software ontology can be populated by either our Eclipse plug-in or text mining system. The discovered instances from different sources can be automatically linked through ontology alignment [17]. Based on the software ontology, users can define new concepts/instances for particular software maintenance tasks through an ontology management interface. Text Mining (TM) is commonly known as a knowledge discovery process that aims to extract non-trivial information or knowledge from unstructured text. Unlike Information Retrieval (IR) systems, TM does not simply return documents pertaining to a query, but rather attempts to obtain semantic information from the documents themselves, using techniques from Natural Language Processing (NLP).We implemented our TM subsystem based on the GATE (General Architecture for Text Engineering) framework [4], one of the most widely used NLP tools [22].

The ontological reasoning services within the SOUND environment are provided by the ontology reasoner, Racer [7]. Racer's query language nRQL can be used to retrieve instances of concepts and roles in the ontology. An nRQL query uses arbitrary concept names and role names in the ontology to specify properties of the result. In a query, variables can be used to store instances that satisfy it. However, the use of nRQL queries is still largely restricted to users with a good mathematical/logical background due to nRQL's syntax, which, although comparatively straightforward, is still difficult for programmers to understand and apply. To bridge this conceptual gap between practitioners and Racer, we have introduced a set of built-in functions and classes in the JavaScript interpreter, Rhino[4], to simplify querying the ontology for users. The scriptable query language allows users to benefit from both the declarative semantics of Description Logics as well as the fine-grained control abilities of procedural languages.

In our previous work, we have already demonstrated an ontological model of source code and documentation supporting various reverse engineering tasks, such as program comprehension, architectural analysis, security analysis and traceability links [22]. We currently investigate its integration with work examining the requirements for software reverse engineering repositories [15] that deals with incomplete and inconsistent knowledge on software artifacts obtained from different sources (e.g., conflicting information delivered by source code and document analysis).

[4] available online at http://www.mozilla.org/rhino/

3.3 Process Management

The interaction among users and the knowledge resources plays a dominant role in any software maintenance process model. As part of this interaction, users should become immersed in the program maintenance process, while the different phases of a particular maintenance task unfold. The user itself is active and interacts with different resources (e.g., support from tools, techniques, documents and expertise) and other users (e.g., system or historic user data) to complete a particular task. In this research, we introduce a process management approach that establishes the communication and interaction between users, the process and the underlying ontology manager [15]. A typical usage scenario of our maintenance process model is illustrated in Fig. 3. with the iterative nature of the process being reflected by the loop (messages 2-22). A user is completing a comprehension task and the process manager, ontology manager, reasoner and available resources are all working together to assist the user during the different phases of the software maintenance process.

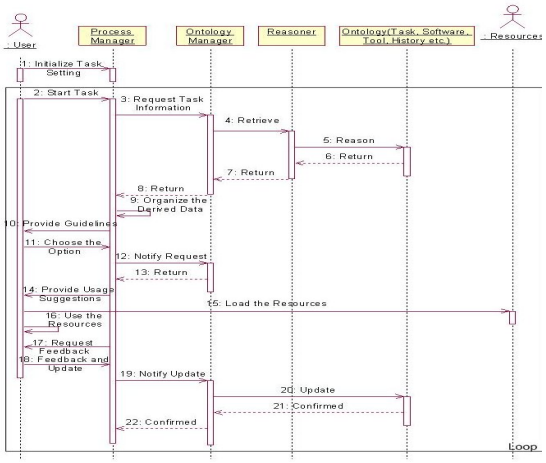

Fig. 3. Process Sequence Diagram

After each iteration, users will provide feedback and annotate briefly their experience with the resources and their success towards problem solving (Messages 18-22). The resulting feedback is used to further enrich the historical data stored in the ontology, as well as trigger the next step in the maintenance process.

In this research, we introduce an iterative process management approach that guides the communication and interaction between users, the process and the underlying ontology manager. A user is completing a comprehension task and the process manager, ontology manager, reasoner and available resources are all working together to assist the user during the different phases of the software maintenance process.

After each iteration, users will provide feedback and annotate briefly their experience with the resources and their success towards problem solving. The resulting feedback is used to further enrich the historical data stored in the ontology, as well as trigger the next step in the maintenance process. A more detailed description of the process manger can be found in [15].

4. System Implementation and Evaluation

In this section, we provide a general system overview of the implementation of our process model and a general overview of the ontological implementation used to model the knowledge base.

4.1. System Overview

The process itself is based on two main components, the process and the ontology manager.

Ontology Manager is used to manage the infrastructure of the process model, where the basic elements of the program comprehension process and their inter-relationships are formally represented by DL-based ontology. Our approach supports the addition of new concepts and their relations in a given sub-ontology, coordinates the reasoner with the ontologies, and controls querying and reasoning services across the sub-ontologies. A user can perform both pre-defined and user-defined queries. The ontology manager is an extension to our SOUND tool [22], an Eclipse plug-in that provides both ontology management (software ontology and document ontology have been developed) and inferences service integration using the Racer [7] reasoner. So far, the ontology management interface provides the following services: adding/defining new concepts/relationships, specifying instances, browsing the ontology, and a Java Script based query interface.

Process Manager is built on top of the ontology manager and provides users with both the context and the interactive guidance during the comprehension process. The process context is established by the process manager, depending on the user process interactions, the current state of the knowledge base and the resulting information inferred by the reasoner. For interactive guidance, the process manager utilizes different visual metaphors to establish a representation that allows users to immerse in the process context and, at the same time, provides an approach to analyze and utilize the inferred knowledge to provide guidance during the comprehension process itself.

4.2 Initial Evaluation

At the current stage, we have successfully implemented and used our SOUND ontology management and query tool to perform comprehension tasks such as impact analysis, design pattern recovery, and component identification [9]. In addition, we have defined an initial set of concepts and relations for the remaining sub-ontologies as the foundation for our process model. A more detailed description of the ontology implementation can be found in [15].

A set of frequently used queries has been defined in the system, e.g., identifying the coupling among classes, recovering the design pattern in a system. We are currently in the process of conducting a larger case study in collaboration with Defence Research and Development Canada (DRDC) Valcartier to explore and validate the ap-

plicability of our software maintenance process model. The system used for the case study is an open source software for the analysis and reporting of maritime exercises – Debrief [5]. As part of the ongoing Debrief case study, we are performing a specific component substitution task, in which a non-secure file access will be substituted by a client specific encrypted version. Feedback from the process and information resource usage will be collected for further refinement and enrichment of both the process model and the knowledge base.

5. Related work

There exists only very limited research in applying Description Logics or formal ontologies in software engineering. The two major projects that are closely related to our ontological approach are the LaSSIE [6] and CBMS [21] systems. However, these systems are much more restricted by the expressiveness of their underlying ontology languages and they lack the support for an optimized DL reasoner, such as Racer in our case.

Current research in modeling software maintenance processes [10,19,20] typically describe only very generally the process and lack formal representations. Thus, they are unable to utilize any type of reasoning services across the different knowledge sources involved in the comprehension process. To the best of our knowledge, there exists no previous work that focuses on developing a formal process model to describe the program comprehension process.

Existing work on comprehension tool integration focuses either on data interoperability using a common data exchange format [20] or on service integration among different reverse engineering and software comprehension tools [11]. Our approach can be seen complementary to these ongoing tool integration efforts. Improving the overall capabilities and applicability of reverse engineering tools will help to enrich our tool ontology and therefore, directly/indirectly benefit the comprehension process model. However, our approach goes beyond just mere tool integration. It is the formal ontological representation that supports both reasoning across different knowledge sources (including tools) and context support during the comprehension process itself. Furthermore, our approach provides flexibility and extensibility required to support the evolution of the process model itself.

6. Conclusions

Our work promotes the use of both formal ontology and automated reasoning in software maintenance research, by providing a DL-based formal and uniform ontological representation of different information resources involved in a typical software maintenance process.

As part of our future work, we will conduct several case studies to enrich our current ontology and optimize the software maintenance process model for different maintenance tasks. We are currently in the process of developing a new visual process metaphor to improve the context-sensitive guidance during typical maintenance tasks.

Acknowledgement

This research was partially funded by Defence Research and Development Canada (DRDC) Valcartier (contract no. W7701-052936/001/QCL).

References

1. G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, “Information retrieval models for recovering traceability links between code and documentation”. In Proceedings of IEEE International Conference on Software Maintenance, San Jose, CA, 2000.
2. F.Baader, D. Calvanese, D. McGuinness, D.Nardi, P.P.-Schneider, “The Description Logic Handbook”. Cambridge University Press, 2003.
3. R. Brooks, “Towards a Theory of the Comprehension of Computer Programs”. Int. J. of Man-Machine Studies, pp. 543-554, 1963.
4. H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan. “GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications.” Proceedings of the 40th Anniversary Meeting of the ACL (ACL'02). Philadelphia, July 2002.
5. Debrief, www.debrief.info, last accessed 25/10/2006.
6. P.Devanbu, R.J.Brachman, P.G.Selfridge, and B.W.Ballard, “LaSSIE: a Knowledge-based Software Information System”, Com. of the ACM, 34(5):36–49, 1991.
7. V. Haarslev and R. Möller, “RACER System Description”, In Proc. of International Joint Conference on Automated Reasoning, IJCAR'2001, Italy, Springer-Verlag, pp. 701-705.
8. P. N. Johnson-Laird, “Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness”. Harvard University, Cambridge, Mass., 1983.
9. A. V. Mayhauser, A. M. Vans, “Program Comprehension During Software Maintenance and Evolution”. IEEE Computer, pp. 44-55, Aug.,1995.
10. IEEE Standard for Software Maintenance, IEEE 1219-1998.
11. D. Jin and J. R. Cordy. "Ontology-Based Software Analysis and Reengineering Tool Integration: The OASIS Service-Sharing Methodology". 21st IEEE ICSM, 2005.
12. P. N. Johnson-Laird, “Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness”. Harvard University, Cambridge, Mass., 1983.
13. M. I. Keller, R. J. Madachy, and D. M.Raffo, “Software Process Simulation Modeling: Why? What? How?”. Journal of Systems and Software, Vol.46, No.2/3, 1999.
14. U. Kölsch and R. Witte, “Fuzzy Extensions for Reverse Engineering Repository Models”. 10th Working Conference on Reverse Engineering (WCRE), Canada, 2003.
15. W. Meng, J. Rilling, Y. Zhang, R. Witte, P. Charland, “An Ontological Software Comprehension Process Model”, 3rd Int. Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006), Genoa, October 1st, 2006, pp. 28-35.
16. Niles and A. Pease. “Towards a Standard Upper Ontology”. Proc. of the 2nd Int. Conf. on Formal Ontology in Information System (FOIS), Maine, 2001.
17. N. F. Noy and H. Stuckenschmidt, “Ontology Alignment: An annotated Bibliography – Semantic Interoperability and Integration” Schloss Dagstuhl, Germany, 2005.
18. A. Ballim, Wilks, “Artificial Believers: The Ascription of Belief”,Lawrence Erlbaum,1991.
19. C. Riva, "Reverse Architecting:An Industrial Experience Report", 7th IEEE WCRE, pp.42-52, 2000.
20. M. -A. Storey, S. E. Sim, K. Wong, “A Collaborative Demonstration of Reverse Engineering tools”, ACM SIGAPP Applied Computing Review, Vol. 10(1), pp18-25, 2002.
21. C.Welty, “Augmenting Abstract Syntax Trees for Program Understanding”, Proc. of Int. Conf. on Automated Software Engineering. IEEE Computer Soc .Press. 1997, pp. 126-133.
22. Y. Zhang, R. Witte, J. Rilling, V. Haarslev, *“An Ontology-based Approach for Traceability Recovery”*, 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006), Genoa, October 1st, 2006, pp. 36-43.