A Story Driven Approach to Software Evolution

Juergen Rilling[1], Wen Jun Meng[1], René Witte[1], Philippe Charland[2]

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

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

Abstract

From a maintenance perspective, only software that is well understood can evolve in a controlled and high quality manner. Software evolution itself is a knowledge-driven process that requires the use and integration of different knowledge resources. In this paper, we present a formal representation of an existing process model to support the evolution of software systems by representing knowledge resources and the process model using a common representation based on ontologies and description logics. This formal representation supports the use of reasoning services across different knowledge resources, allowing for the inference of explicit and implicit relations among them. Furthermore, we introduce an interactive story metaphor to guide maintainers during their software evolution activities and to model the interactions between users, knowledge resources, and the process model.

Keywords: Software evolution, software maintenance, process modeling, story metaphor, ontological reasoning

1. Introduction

Software evolution is a knowledge-driven process, with knowledge being continually integrated from different sources (including source code repositories, documentation, test case results) and at different levels of abstraction (from single variables to complete system architectures). For maintainers to perform and complete a particular maintenance task, they typically need to use and interact with various tools and techniques (e.g., parsers, debuggers, source code analyzers, visualization tools). Identifying knowledge resources that are applicable in a given maintenance context can become a major challenge for software maintainers. Furthermore, there often exist non-obvious direct or indirect dependencies among these knowledge resources that require maintainers to follow a certain sequence of steps in order to accomplish a particular evolution task. As a result, both maintainers and organizations are facing the challenge of deriving and applying procedures that provide for guidance to utilize these existing resources more efficiently [47, 48, 49].

Various process models [5, 6, 10, 19, 49] supporting the evolution of software have been introduced. Common to most software and, more specifically, software evolution process models, is that they share a generality in abstracting and describing activities to be performed and resources to be used as part of the process. This generality can become, from an organizational viewpoint, a major challenge in adopting them. Organizations often have to adapt their own internal processes to support a particular process model, and they also need to modify and customize this process model so that it can be supported or sustained as part of a larger organizational context. Organizations are often left alone in establishing context-awareness and customization of process models. From an organizational perspective, it is possible to define processes in such a detail that the resulting model not only describes the required maintenance activities, but also resources that must be employed. However, these well defined process models are typically based on the as-

sumption that resources and the knowledge provided by these resources are known at the process specification time, resulting in a static (closed world) resource and knowledge allocation. This closed world assumption however restricts the ability to integrate newly gained knowledge and resources within these process models. There has been little work in examining how these resources can be applied and used collaboratively to support a specific software maintenance task [43]. Maintainers are often left with no guidance on how to complete a particular task within a given context, using a set of available resources (e.g., tools, artifacts). Current research in software evolution has lead to a set of task specific tools and techniques that are not integrated, due to a lack of integration standards and frameworks to allow sharing services among them.

The problem of knowledge and process integration is not unique to software maintenance. Other application domains, including Internet search engines (e.g., Google[1]) or online shopping sites (e.g., Amazon[2]) are facing similar challenges. Common to these application domains is that they use different information resources to support users in a given context. For example, during an online shopping session, users will typically find related information on other relevant products, customer reviews, product summaries, etc. Tools and techniques are utilized to enhance the shopping experience and facilitate the online shopping process. The two major challenges in applying similar approaches to support software evolution are: (1) The lack of formal models to represent and link relevant knowledge resources and process activities; and (2) the resulting lack of a suitable metaphor to model the interaction between users, the model, and the relevant resources.

In this research, we focus in particular on modeling the interactions between maintainers, a process model, and its relevant knowledge resources. The presented research is a continuation of our previous work on modeling program comprehension [27] by introducing a software evolution ontology that models both software evolution process specific aspects and knowledge resources relevant to software evolution.

1 www.google.com
2 www.amazon.com

We introduce a novel formal ontological representation that models and integrates software process specific information with software and other knowledge resources (e.g., domain knowledge, documents, user expertise, historical data, etc.). The goal of this common representation is to reduce the conceptual gap caused by the type of abstractions and languages found in these artifacts. Having a common ontological representation allows maintainers not only to explore knowledge relevant to their given task across the different modeled artifacts, but also to enrich their current understanding of a system and share their knowledge. Furthermore, traceability between software processes and knowledge resources can be established by providing non-trivial relationships among these artifacts. Ontologies also provide support for extending an existing knowledge base to reflect more closely an "open" world assumption [3], by allowing for both the modeling of incomplete knowledge and the enrichment of the knowledge base with newly gained knowledge. Moreover, having a formal ontological representation also allows us to take advantage of automated reasoning services provided by ontology reasoners to infer implicit relations (links) between processes and knowledge resources. Figure 1 shows a simplified example that restricts the available information resources to tools and tasks. Different strategies (top-down, bottom-up or integrated approach) are provided to support maintainers in the use of the knowledge resources and to answer questions such as:

(1) Which tools might directly or indirectly be required to perform a particular comprehension task following a top-down approach?
(2) Given a current knowledge level acquired using a bottom-up strategy, what are the potential (directly/indirectly) related tasks that can be performed?

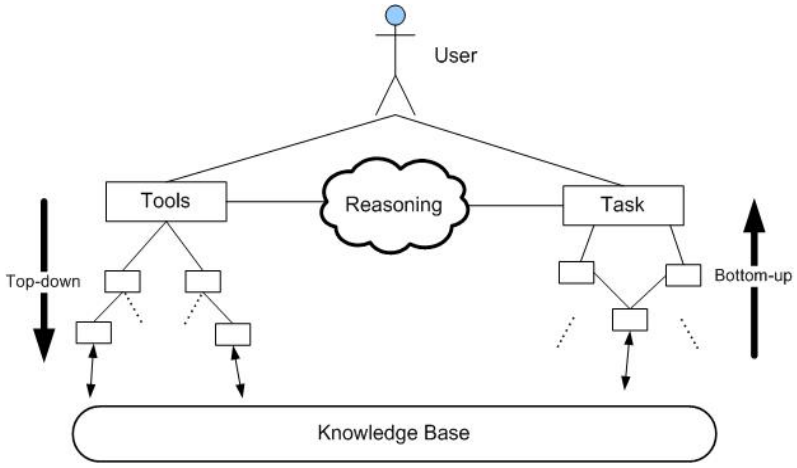

Fig. 1 Conceptual model

Our research is not only motivated by this need to synthesize these different information and knowledge resources used within our formal framework, but also by the ability to provide maintainers with a context directly related to a specific process activity. The acceptance of any new technology or approach is directly dependent on delivering added benefits to the users or management of an organization. For a process environment to be accepted by maintainers, it is essential that it provides a supporting framework that captures the context in which the process model is applied and that it also links resources to the process. For the contextual representation, an intuitive visual metaphor is needed to establish an interaction model between the user and the process model. We introduce a *story* metaphor that addresses the following three major issues:

(1) a metaphor that is familiar to users;
(2) a metaphor that can be mapped closely to the activities of a software evolution process and its activities;
(3) a metaphor that can be expressed at different levels of abstraction to reflect the different granularity of knowledge and context that need to be represented.

The story metaphor not only models the interactions between users and the ontological representation in terms of process context and resources, but it also provides the basis for developing a supporting environment.

The overall goal of our research is not to provide a concrete solution to a specific software evolution activity, but rather make use of explicit and implicit knowledge found in the software maintenance domain. We focus on the integration of this knowledge within the maintenance process by establishing traceability links between process activities and applicable resources at the knowledge level, and modeling the interaction between users and the process.

Our approach differs from existing work on comprehension models [25, 26, 38], tool integration [20, 43], and software evolution related process models [5, 6, 10, 19,36] in several important aspects:

1. We introduce a formal ontological representation based on Description Logics (DL) to support the modeling, integration and linking of processes (process activities), and knowledge resources relevant to these processes.
2. The ability to reason about knowledge modeled using this ontological representation to infer explicit and implicit knowledge, in order to provide an *active and context sensitive* guidance during a software evolution process. This includes knowledge inference across multiple sub-ontologies and the ability to model incomplete knowledge.
3. An interactive story metaphor was adopted to model the dynamic interaction aspects between users and the comprehension process.
4. An environment utilizing the modeled knowledge, process, and user interaction to guide maintainers during software maintenance tasks.

The remainder of the paper is organized as follows: Section 2 provides a brief review of existing software evolution process models and their limitations. Section 3 introduces

background related to OWL ontologies and inference services provided by ontological reasoners. Section 4 introduces the conceptualization of software evolution and its corresponding ontological model. Section 5 introduces the story metaphor used to model the interaction between users and the process. Section 6 discusses the implementation and validation issues, followed by Section 7 with the related work. Conclusions and future work are outlined in Section 8.

2. Software Evolution Process Models

Historically, software lifecycle and process models 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. One approach to model software maintenance is to include *software maintenance* aspects as part of the total system life cycle/process model, as suggested for example in [5, 19]. Other approaches to model software maintenance include deriving maintenance specific process models. Among these maintenance specific models are the quick-fix, iterative enhancement, full-reuse model [5], the staged model [6], the SEI CMMI model [10], and the IEEE Std 14764-2006 – Software Life Cycle Processes – Maintenance [19].

One of the challenges in applying these models is that various aspects can affect software evolution [26, 30, 42], making it an inherently complex and difficult problem to address. Some of these aspects influencing software evolution include a user's comprehension ability (e.g., experience, knowledge background), the characteristics of a software system to be maintained (e.g., its application domain, size, and complexity), the maintenance task itself to be performed (e.g., adaptive, corrective, or perfective maintenance), as well as the tools and software artifacts available to support the evolution.

Common to most existing process models, including evolution models, is that they do not specify *how* any available supporting resources (tool, system, user expertise, software

artifacts, etc.) should be integrated within the process in a given context. Current research in software maintenance focuses mostly on providing conceptual guidance (such as the documented standards) or on developing tool support to address some specific aspects of a software maintenance task. An example of such a process model is the IEEE Std 14764-2006 [19] that lists and describes activities and their sub-activities, referred to as task-steps, as part of the process model. However, the standard provides only limited or no details on how to implement or perform the activities and task-steps specified in it.

Common observations that can be made from most existing software evolution process models are:

- They tend to provide only general descriptions of the steps involved in a process, lacking details or guidelines on how to complete these steps in a given task and/or organizational context.
- They are limited in their ability to provide a cohesive integration of existing knowledge resources (e.g., user expertise, source code information, tools) and newly gained knowledge (e.g. experienced gained from previously performed maintenance tasks) within the process.
- Lack of tool support that can provide users with an active, context-driven guidance during the comprehension process, to help reduce their cognitive load.

3. Ontologies and Reasoning

Intuitively, maintainers perform various activities while maintaining a software product. These activities include understanding, conceptualizing, and reasoning about the software to be maintained. Therefore, support for developing a user's mental model during software evolution is needed to assist and improve human thinking processes. Research in cognitive science has suggested that mental models may take many forms, but the content normally constitutes an ontology [21].

Ontologies have their origins in philosophy, where ontologies correspond to a theory about the nature of existence and the categories of things that exist [45]. In computer science, ontology is "an explicit specification of conceptualization" [13] to provide a formal model of a certain domain. There exists a variety of ontology languages with different degrees of formality. Some of these languages are based on graphical notations such as semantic network, UML, or RDF. Others are based on logic such as Description Logics (DL) (e.g., OIL, DAML+OIL, OWL), Rules (e.g., Prolog), or First Order Logic (e.g., KIF) [3, 18, 40].

DLs are a family of logic-based knowledge representation formalisms [2] that can be distinguished by their formal semantics and inference services. DLs describe domains in terms of TBox (also known as concepts or classes), roles (also known as relationships or properties) and ABox (also known as individuals or instances). Particular DLs can be characterized by the set of constructors they support for building complex concepts and roles from simpler ones and the set of axioms available for asserting facts about concepts, roles, and individuals [18]. Although DLs have many applications (e.g., databases, configuration, software engineering and natural language processing [2]), DLs are currently best known as the basis for formal ontological languages such as OIL, DAML+OIL and OWL [18].

3.1 Web Ontology Language (OWL)

The Web Ontology Language (OWL) [18, 44] is the foundation of the Semantic Web [7], providing machine understandable Web information to enable automatic processing and integration of Web resources. OWL was created based on Web standards such as XML and RDF(s) and it exploits SHOIN(D) DL for supporting greater machine interpretability of Web content [18]. OWL is built on earlier DL-based ontology languages OIL and DAML+OIL and is the current recommendation of the World Wide Web Consortium

(W3C)[3] [44]. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL-DL, and OWL Full. We adopted OWL-DL as it provides the best tradeoff between expressiveness and reasoning power.

An OWL ontology can be seen as a DL knowledge base that consists of DL TBox and ABox. OWL describes a domain in terms of classes, properties and individuals. OWL ontologies can be constructed incrementally by first specifying elementary descriptions, such as simple classes and properties, and then defining more complex class and property descriptions inductively through a set of OWL-DL constructors and axioms. A class description can also be used to define queries to describe sets of individuals [2].

OWL ontologies can simply be used as a data storage medium, similar to traditional databases. However, the use of DL to define the ontological model allows for a more precise and expressive representation than traditional data semantics [2] and is also the foundation for automated reasoning support. Reasoning is a mechanism to infer implicitly represented knowledge from the knowledge that is explicitly represented in a knowledge base [2]. DL has been designed to support both TBox (concept) reasoning and ABox (individual) reasoning. Typical reasonings for TBox are satisfiability check to determine whether a description is satisfiable (can have individuals) or subsumption test, to examine whether one description is more general than another one [2]. Concepts can be organized into a terminology hierarchy according to their generality through the use of subsumption reasoning [2]. Other TBox reasoning includes classification and consistency check. Basic reasoning for the ABox includes instance checking, i.e., whether a given individual is an instance of a certain concept. Additional ABox instance reasonings can be derived from instance checking, such as instance retrieval, tuple retrieval, and instance realization. For a more detailed coverage of DL and reasoning services, we refer the reader to [2, 14].

[3] http://www.w3.org/

3.2. Process Ontology Example

In what follows, we briefly illustrate the use of both ontologies and reasoning services on a simplified subset of our ontological model. Figure 2 presents a simplified ontology that consists of four atomic concepts (Person, Team, Organization, and Project) and three atomic roles (manage, program, and work). The ontology is populated with four individuals, Mike (Person), InfoGlue_001, uDig_001 and Debrief_001 (Project).

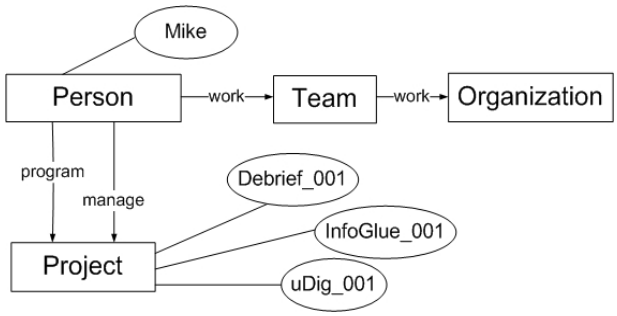

Fig. 2 A simplified partial maintenance ontology

Defining New Concepts

Two new concepts, `Manager` and `Programmer`, can be defined to enrich the existing ontology (Figure 2). In the following two expressions, `Manager` is first defined as a person who has managed some project; likewise, `Programmer` can be defined as a person who has performed some programming as part of a project. Based on these concept definitions, one can now verify the following assumptions about a person `Mike`. If the assumption "`Mike` has programmed in a certain project such as `Debrief_001`" is true, the assumption that "Mike is a programmer" will also automatically hold.

$$\texttt{Manager} \equiv \texttt{Person} \sqcap \exists \texttt{manage.Project}.$$

$$\texttt{Programmer} \equiv \texttt{Person} \sqcap \exists \texttt{program.Project}.$$

TBox Reasoning

OWL-DL allows for the specification of a role hierarchy through the use of the subproperty axiom. The following example illustrates a TBox classification through the use of such a subPropertyOf axiom. A new role `participate` can be defined as a super role of `manage` and `program`, denoted as:

```
manage ⊑ participate,
program ⊑ participate.
```

Having such a `participate` role specified, a new concept `participant` can now be defined as a person who has participated previously in some projects:

```
Participant ≡ Person ⊓ ∃participate.Project.
```

Given the above knowledge, the reasoner can now infer that `Manager` and `Programmer` are both subconcepts of `Participant` and therefore, all instances of `Manager` and `Programmer` become instances of `Participant`.

OWL-DL also supports transitive roles. Given the transitive roles `P(x,y)` and `P(y,z) -> P(x,z)`. In Figure 2, `work` corresponds to such a transitive role, where a person works in a team and a team itself works on some projects. The person works in an organization relationship is not asserted, but based on the transitive `work` role, the reasoner can now infer that all persons who work in a team also work for the organization the team belongs to.

ABox Reasoning

In addition to TBox reasoning, ABox reasoning can also be applied to perform instance checking. For example, `Mike` is a Person and `Debrief_001`, `uDig_001`, and `InfoGlue_001` are three different project instances. We can now define a new concept `ExperiencedProgrammer` as:

$$ExperiencedProgrammer \equiv Programmer \sqcap \exists \geq 3 program.Project.$$

Given that Mike has programmed in three project instances (Debrief_001, uDig_001 and InfoGlue), the reasoner can now automatically infer that Mike is a project participant and also an experienced programmer.

Summary

Based on the above discussion, we adopted OWL-DL as the ontological modeling language for our research. OWL-DL representation is applicable to model the dynamically evolving software maintenance domain, due to its support for domain knowledge evolution (concept, role and instance creation and population). In addition, it not only provides a more precise and expressive representation of the domain, but also enables automated reasoning to process the content of the represented domain. For a more detailed discussion on OWL ontologies, DL and reasoning, we refer the reader to [2, 14, 44].

4. A Unified Ontological Software Evolution Process Model

Models are essentially an abstraction of real and conceptually complex systems to represent their significant features and characteristics [22]. For any model to provide added benefits, it is essential that the model is being accepted and used by the expected users. In what follows, we introduce two criteria we applied for modeling software evolution processes and their related knowledge resources.

- *Support and adaptability:* For a process model to be adopted by an organization and/or end-user, the model has to be adaptable to a given organizational structure. It also has to provide additional benefits to the user in the form of providing guidance in applying the model towards a specific maintenance task.
- *Flexibility and extendibility:* Process models have to be able to adapt to ever

changing maintenance contexts and knowledge resources, therefore requiring the ability to evolve over time to reflect these new requirements.

4.1 Conceptualizing Software Evolution

Building a formal ontology for software evolution requires the analysis of concepts and relations in this discourse domain. From a software practitioner's perspective, it is therefore essential that the ontological Knowledge Base (KB) includes and models concepts and roles critical to software evolution processes. Indeed, our conceptualization work has been influenced by other works in modeling software maintenance [11, 23, 37] and the observations of best software maintenance practices. Our approach to construct this KB is twofold: (1) We created sub-ontologies for each of the key discourse domains, such as tasks, software, documents, and tools; and (2) we linked 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 [28]. Figure 3 provides a simplified view of the resulting meta-model, focusing only on the major concepts and their roles. In what follows, we briefly summarize these key sub-ontologies.

Task: Describes a unit of work that is triggered by an emerging modification request (MR) or problem report (PR). Information about MRs or PRs, task assignment, tool log (tool used in the task solving process), activity log, etc. is modeled in this sub-ontology. For example, the instances in this sub-ontology might be "Debrief XML decrypt/encrypt component substitution request" (an instance of concept `Request`), "corrective" (an instance for concept `RequestType`), and "successfully finished" (an instance for concept `TaskStatus`).

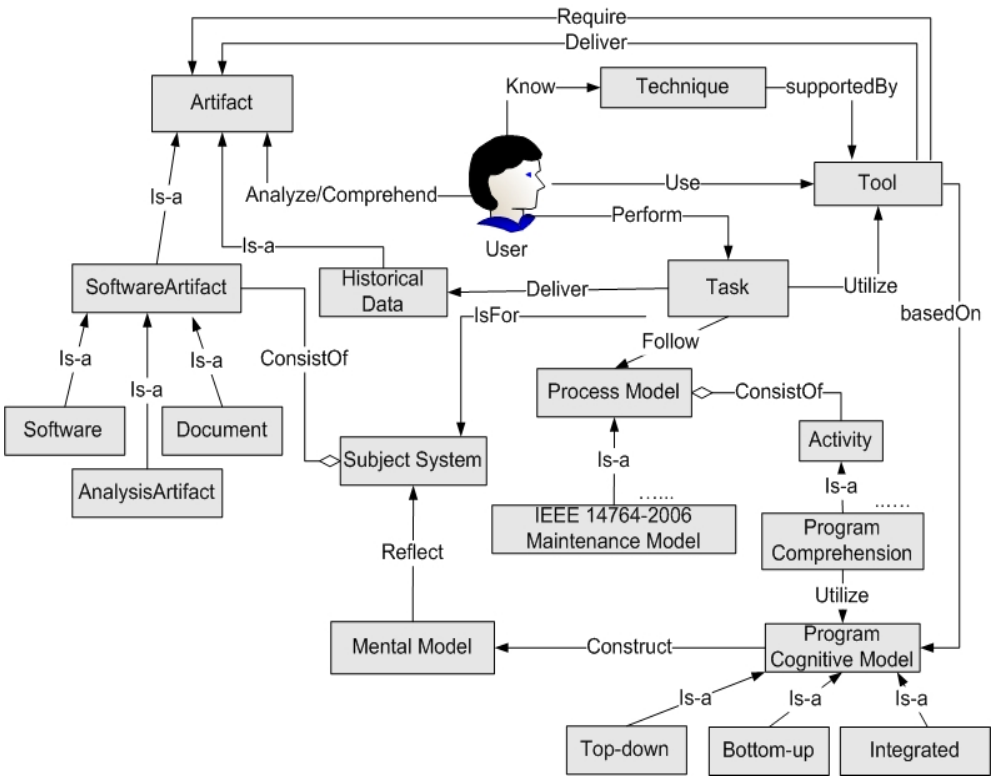

Fig. 3 Software maintenance meta-model

User: Describes maintainers involved in the software maintenance process. Information about the involved roles and their related skills, competencies and responsibilities are modeled. The users involved in the software maintenance can be an individual, an organizational team, or an entire organization. The example instances in this sub-ontology might be "Mike Smith" (an instance of concept `Person`), "SE group" (an instance of concept `Maintainer`), and "MR reporter" (an instance of concept `Role`).

Artifact: Describes artifacts associated with both the software product (source code, documentation, etc.) and the maintenance process (documents, models, etc.) that are produced as part of the maintenance process (Figure 4). At the current stage, we consider

artifacts to be either related to the software system or to the maintenance task. Figure 4 shows a partial view part of the taxonomy of the artifact sub-ontology. Artifact itself is a super class of the `Artifact` sub-ontology, which has two subclasses: `SoftwareArtifact` and `TaskArtifact`. The software system related artifacts are further classified into software documents, software components and libraries, and artifacts made available by supporting tools. Task related process artifacts correspond to artifacts, like guidelines, reports, etc.

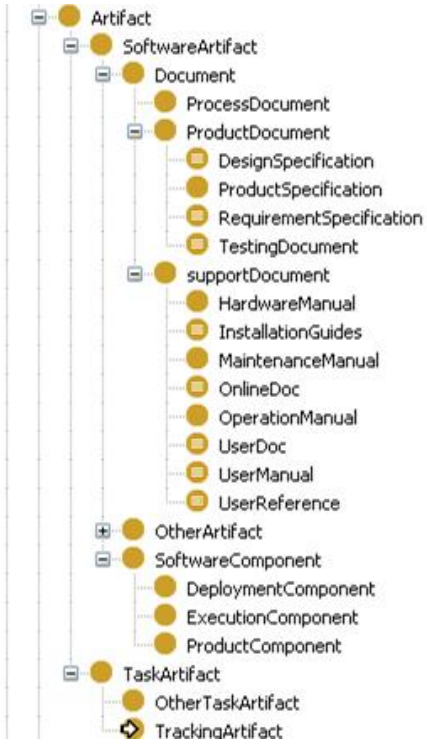

Fig. 4 Partial view of the `Artifact` sub-ontology

Subject system: Describes the software system to be maintained and includes information about the programming language(s) used, application domain, etc.

Process: Describes the interactions and relationships among different sub-activities within a maintenance process model. The example instances in this sub-ontology can be "IEEE Std 14764 – Software Life Cycle Processes – Maintenance" (an instance of concept `ProcessModel`), "5.3 Modification Implementation" (an instance of concept `Activity`), "5.3.2.1 Analysis" (an instance of concept `Task`), and "5.3.2.1 a" (an instance of concept `TaskStep`).

Technique: Describes the software techniques that can be used for supporting software maintenance (program comprehension technique, source code analysis technique, impact analysis technique, etc.). Based on the previous work by Dias et al. [55] and Pressman [57], the following techniques supporting the following activities can be identified: requirement elicitation, maintenance support, programming related, testing, configuration management, documentation, and modeling technique.

Tool: Describes the software tools used to carry out a function or service with the goal to simplify maintenance tasks [56]. At the current stage, our Tool sub-ontology provides general categories for commonly used maintenance tools and describes their key features for supporting software maintenance activities. Tools can be classified by their functionality, their role in supporting managers and maintainers, their particular use during the various steps of a software engineering process, their supported environment (hardware and software), or even by their origin or cost. Pressman [57] provides a comprehensive classification of tools by their functions, which includes, among others, analysis, design, programming, software configuration management, testing and documentation tools. In our approach, tools and techniques in the KB are automatically classified through the use of reasoning. For example, programming tools can be classified based on their support for a particular programming language, operating system, applicability for a specific task, etc.

As discussed previously, software evolution is a multifaceted and dynamic activity involving different resources, which utilizes these resources to enhance the current knowledge about a system. Existing work on ontological modeling of the software engineering domain, including its processes, has focused on conceptualizing the domain [50, 51] to establish a common terminology or to model specific aspects of software engineering processes [50, 51, 52]. In the context of our research, we adopt core parts of these ontologies and further enrich them with new concepts and relationships to more closely reflect (1) the particular needs of our particular modeling goal – establishing traceability links between the process and various knowledge process related knowledge resources. (2) provide a design that fully supports and utilizes optimized DL reasoners, such as Racer [14] in our case, to infer additional knowledge.

An essential part of our model is the ability to support both the knowledge acquisition and use of the newly gained knowledge. The unified ontological representation provides the ability to dynamically add new concepts and relationships, as well as new instances of these to the KB. This extendibility enables our model to be constructed in an incremental way, closely modeling the same iterative knowledge acquisition behavior used to create a mental model as part of human cognition of a software system. Having these different sub-domains modeled as sub-ontologies also allows for the automated processing (e.g., reasoning) and integration of this knowledge at a concept and/or instance level. However, it would not be realistic to expect that all these sources share a single, consistent view within a maintenance task. Rather, we expect disagreements between individual users and sub-ontologies during an analysis. We explicitly capture those different views using a representational model that attributes information to (nested) contexts using so-called viewpoints [4]. The detailed knowledge management strategies have been discussed already in our previous paper [27]. Having this knowledge management mechanism in place, our ontological model can evolve over time to reflect new maintenance contexts and knowledge resources, addressing the flexibility and extendibility acceptance criteria introduced earlier.

4.2 An Ontology Based Software Evolution Process Model

For the foundation of our evolution model, we have adopted the most recent IEEE software maintenance standard [19]. The maintenance standard describes activities for both managing and executing software maintenance tasks. The starting point for the maintenance process is either a maintenance request (MR) or problem report (PR). The maintenance process itself is detailed within the standard by six major activities: process implementation, problem and modification analysis, modification implementation, maintenance review/acceptance, migration, and retirement. Each of these activities is again described by a set of tasks, which are further refined by a list of task-steps. These task-steps correspond to the most fine-grained activities described by the standards document. However, the IEEE standard, like most process models, describes only a general software maintenance process without specifying any details on how to adopt or perform these specific activities and tasks in a given maintenance context. Furthermore, the IEEE standard also lacks support on how existing knowledge and resources within an organization can be adopted to support these process activities.

Presently, we are limiting our modeling scope to a subset of these activities described within the IEEE standard, namely the problem modification analysis, modification implementation, and acceptance phase, corresponding respectively to Section 5.2, 5.3, and 5.4 in the document. These activities are closely related to performing the actual maintenance task rather than addressing more global organizational issues related to software maintenance (Section 5.1, 5.5, and 5.6 [19]). The major concepts (e.g., maintenance request, activities, tasks, task-steps, and their required input and output) and their relationships (e.g., follows, requires, delivers) documented in the maintenance standard have been modeled as part of our process sub-ontology. Figure 5 shows a partial view of the ontological representation of the IEEE software maintenance process.

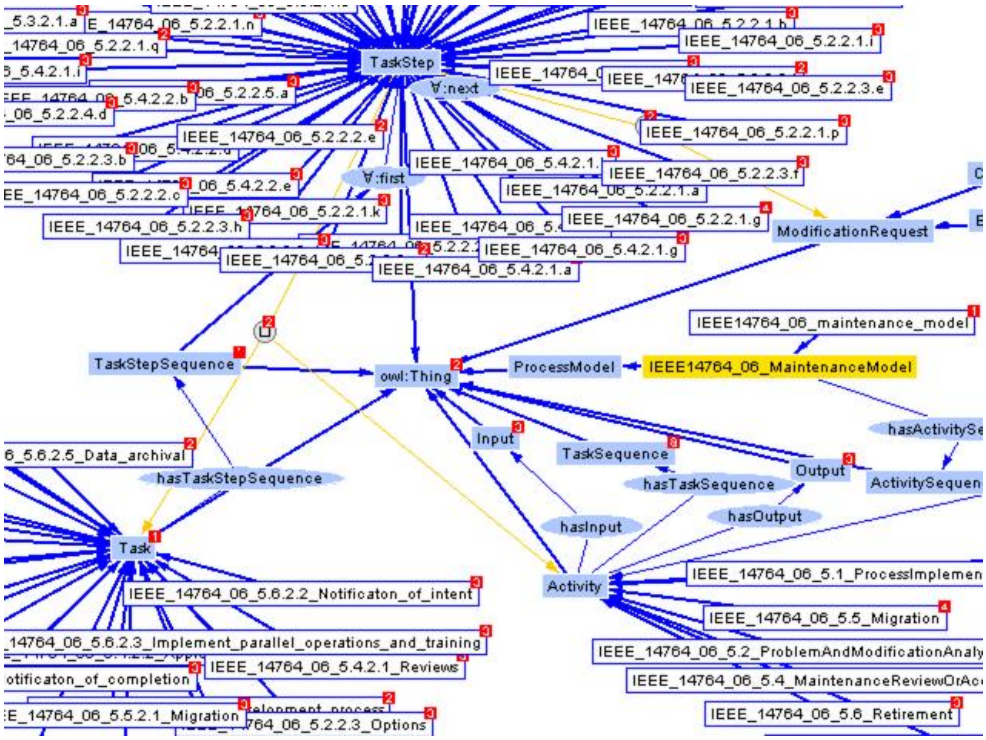

Fig. 5 A partial view of the process sub-ontology

The IEEE software maintenance model is formalized as part of our process sub-ontology, allowing for both the provision of a unified representation of process and knowledge resources and the ability to establish links between knowledge resources and the process activities in the existing maintenance standard. In our approach, these traceability links are established through either design level links via OWL-DL axioms and relations, or at the logical level through a set of predefined queries. In terms of the design level links, OWL-DL axioms or concepts can be used to establish the links between the process model and other knowledge resources modeled. Relationships/properties specified as OWL axioms can be used to establish implicit links as well. For example, in task-step "5.3.2.1 a) Identify the elements to be modified in the existing system" [19], the `supportedBy` property can be utilized to link automatically the task-step with supporting

tool resources. The `supportedBy` relation in Figure 6 is defined as transitive property used to link Task Step with technique (1) as well as for linking Technique with Tool (2). Then the links between TaskStep and tools can be automatically derived through the inferred link (3). As a result, each task-step in the process ontology can automatically be linked through this transitive `supportedBy` property to the subontologies of Technique and Tool. Through further traversing of the transitive closures across the subontologies, other knowledge resources can be identified (Figure 6).

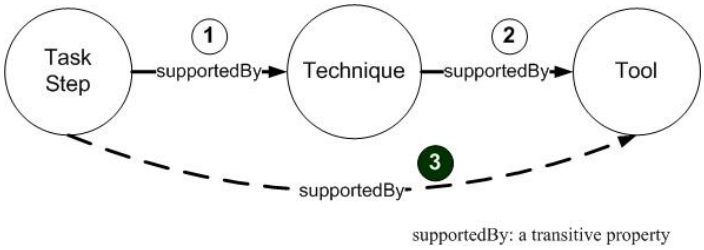

Figure 6. Transitive design level links

In addition to these design level links, logical links between the process and the knowledge resources can be established through predefined queries for each task-step.

Figure 7 illustrates the use of a set of predefined queries to support different process task steps. Users can also define their own supporting queries based on their experience to extract, explore and reason upon the information stored in the KB.

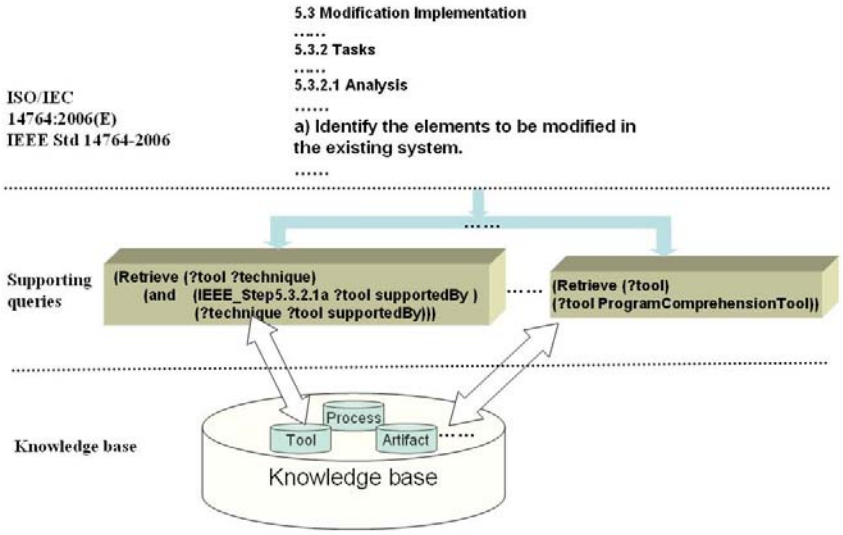

Fig. 7 Task supporting queries

These predefined queries supporting the different task-steps allow users to retrieve explicitly modeled knowledge, as well as inferring implicit knowledge across the KB to support a particular task-step (Figure 7). The query below shows such an implicit knowledge support through a predefined nRQL query [16]:

```
(retrieve (?tool ?technique)
  (and  (IEEE_Step5.3.2.1a ?tool supportedBy )
        (?technique ?tool supportedBy)))
```

In this query, all tools providing techniques applicable to task step 5.3.2.1a in the IEEE standard will be identified. In this query, several sub-ontologies (`process`, `tools`,`techniques`) are queried to retrieve the task step relevant information.

As discussed, our unified ontological representation allows for conceptualizing, storing, linking and retrieving software evolution processes and their relevant knowledge resources. Providing both pre-defined and user-defined queries addresses our first acceptance criterion for the process model, related to support and adaptability. Through the use of the pre-defined and user-defined queries and the underlying reasoning services, our model allows customization of the process support and also provides users with contextual information relevant to their given task. However, a metaphor is needed to represent the interaction between users and the ontological model. In the following section, we introduce a story [31, 35] based metaphor to model this interaction between users and our unified ontology.

5. A Software Evolution Story Model

Stories are ubiquitous, universal, and readily implanted and recalled in human minds [15]. They are widely used to convey information, cultural values, and experience [61]. It is suggested that humans build cognitive structures that represent real events in their lives using models similar to the ones used for narrative stories in order to better understand the world around them [9]. People usually find it is easier to understand information integrated within a story context instead of serial lists [61]. Learners constantly adjust their understanding in accordance with their exposure to conventional narratives, making the construction of narrative a central cognitive goal [32].

5.1. Story Models in Software Engineering

A story itself is a narrative, which, in its simplest form, provides a temporally ordered sequence of events (Table 1) [34]. Storytelling can be applied by scripting a story of the complete system at design-time or by generating stories dynamically on a per-session basis [35]. A system that can generate stories dynamically is capable of adapting the story

narrative to the user's preferences and abilities. Stories themselves can be represented in various ways, like using text, image or animation.

Table 1 Story Model [31]

Story Model		**Description**
Theme		Overall message, concept, or essence of a story; ties every structure and dynamic element.
Genre		Establishes an author's overall attitude, which casts a background on all other considerations.
Characters	Protagonist	Main story character - driver of the story: the one who forces the action to resolve the problem and reach the ultimate goal.
	Impact Character	Might be an antagonist (desire to let the problem grow), guardian (functions as a helper/teacher, a protective character who eliminates obstacles and illuminates the path ahead), or contagonist (works to place obstacles in the path of the Protagonist, and to lure it away from success) etc.
Settings		Place, time, weather conditions, social conditions, or mood or atmosphere etc.
Plot		A series of logic events that develop a story; details the order in which the elements must occur within that story.
Throughline		The point of view and growth of the main character within the story world.
Interaction		What occurs between characters (or even ideas) is presented. Interaction can occur as connections and disconnections.
Conflict		Everything that prevents or gets in the way of the protagonist in achieving the goal.
Climax /Resolution		At this point the obstacles are no longer a threat and any conflict disappears.

The story notation has already been applied in different domains. In Extreme Programming (XP), stories are used as a communication media to elicit requirements. In UML, a scenario diagram is a descriptive story of a use case, detailing what happens between users and the system for a specific function. Sequence diagrams can also be viewed as a more formal story representation of a system behavior, delineating how operations are

carried out - what messages are sent and when, by emphasizing the time ordering of message passing among objects [60].

5.2. A Story-Driven Interaction Model

In this section, we introduce our interactive story-driven software evolution model. The major motivation for applying a story metaphor is to provide us with a metaphor that allows us to model (map) user interactions between process models and knowledge resources to a visual metaphor. The story driven metaphor not only supports the mapping of user interactions with the underlying process model and knowledge resources. It also provides the basis for developing a tool environment to support user interactions with the process model.

From a maintenance perspective, maintainers typically become immersed in the setting in which the particular maintenance task (story) unfolds [8, 48] and the user (maintainer) is considered as an active character in the story, able to interact with different elements, including relevant knowledge resources [46, 49] (e.g., tools, techniques, maintainers and their expertise) and other characters (e.g., system or historic user data) while following the process (through line).

From a story perspective, the evolution process can be viewed as authoring an interactive narrative between users and knowledge resources towards completing a specific goal (task). The similarities among a story model and a maintenance process model allow for a direct mapping of the formal (ontological) representation of the process and resources with the story metaphor.

Furthermore, given the story model, this model can be easily mapped to different metaphors, like text, image, animation, depending on the abstraction level needed. Table 2 illustrates this mapping between the story model and the evolution process.

Table 2 Mapping between the story metaphor and the software evolution process model

Story Model		Software Evolution Model
Theme		Evolution task
Story		A specific maintenance request or problem report
Genre		Type of maintenance activity, e.g., perfective, adaptive, corrective maintenance
Characters	Protagonist	The maintainer or user who performs the evolution task
	Impact/other characters	Historical user experience, process manager, ontology manager and reasoner.
Settings		Organizational settings, available knowledge resources: software, tools, techniques, maintainers, etc.
Plot		Process management
Throughline		Interactive communication between user and plot based on the underlying process model (activities and task-steps)
Interaction		Inter-relationships between the characters and the knowledge resources and process activities
Conflict		Lack of necessary resources, unsuccessful use of resources
Climax/Resolution		Final outcome of the process (success/failure)

A typical usage scenario for our story based evolution process model is illustrated in Figure 8, reflecting the iterative nature of the knowledge acquisition and its use by the loop (messages 2-22). A user (maintainer) corresponds to the protagonist in the story who has to complete a particular evolution task. The story manager, ontology manager (process manager), and reasoner are all examples of impact characters that might impact the user and, therefore, the story interaction. Knowledge resources are the available elements in the story environment (setting), and users can use them during the task solving process. The *story manager* is the main character with whom the user interacts during the task solving process.

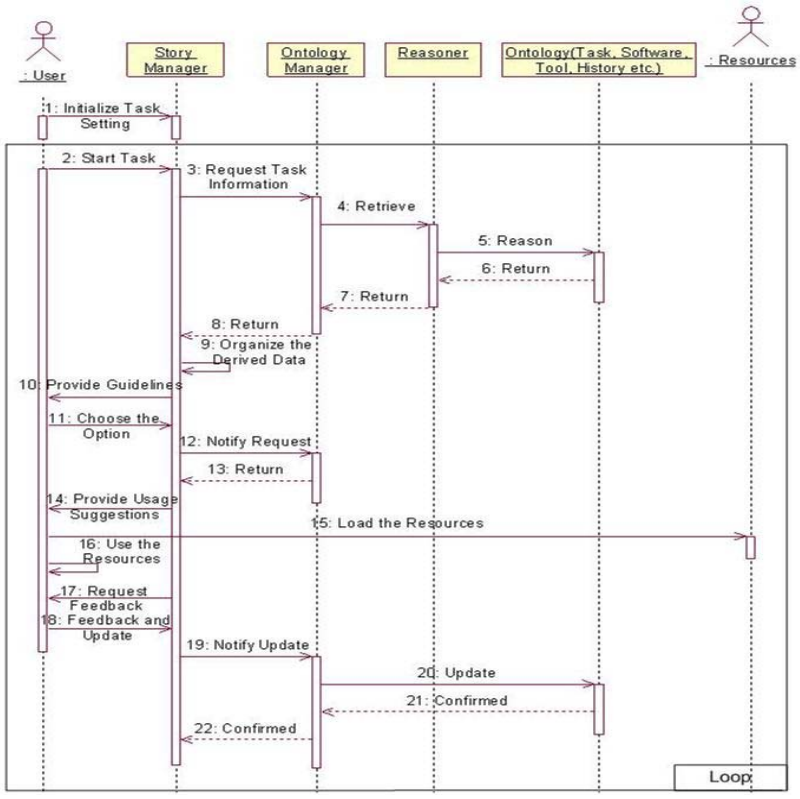

Fig. 8 Resources usage sequence diagram

Based on a given story (task) setting (Message 1), the story manager applies a set of predefined queries to support the different process activities, by identifying potential knowledge and other resources that might be applicable to the current activities (Message 2-14). The user interacts with the process through the story metaphor, providing the current process context and the ability to trigger new events and actions. One of the major advantages of our ontological representation is not only the ability to reason across the different information resources, but also the ability for users to add newly gained knowledge (concepts, relations, and instances) to the system and make these available for further processing. For instance, Message 9 may return a list of techniques that support a

particular maintenance activity, like impact analysis or reverse engineering. The resulting set of tools will be further analyzed by the process manager and a potential applicability ranking of the tools and techniques is provided. At this point, the protagonist has the choice between three different options: (1) accept one of the suggestions (shown in the scenario in Figure 3 – Messages 15-18), (2) explore all available knowledge and other resources stored in the ontologies, or (3) create customized queries to search and filter information for specific settings, tools, or historical data.

After the completion of a task-step, the protagonist will provide feedback and annotate briefly the applicability of the knowledge resource towards solving a particular task-step (Messages 18-22). The feedback is used to further enrich the ontology, as well as triggering the next task-step or activity in the evolution process.

6. Implementation and Validation

In this section, we provide an overview of our system implementation followed by a discussion evaluating our presented approach. As stated earlier, our research focuses not on developing a new tool or technique in attempt to support a specific maintenance activity. Instead, our goal is to automatically integrate implicit and explicit knowledge resources through the use of queries and automated reasoning within a maintenance process. Within our system, maintainers can directly query the KB to explore knowledge or utilize predefined views and queries to automatically establish a task (story) specific context.

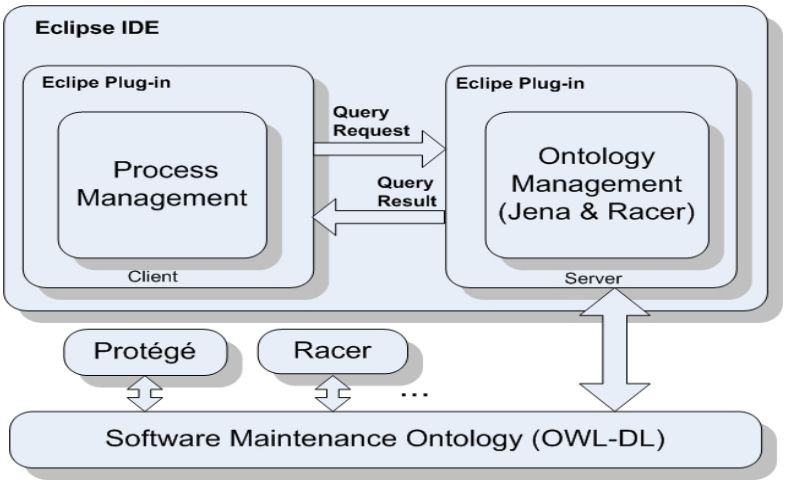

Fig. 9 System architecture

6.1 System Overview

We have developed an environment that supports our unified model built on a client/server architecture with both the server and the client being implemented as Eclipse plug-ins (Figure 9).

Ontology, persistent storage management, and reasoning services are provided through the server application. They are implemented in Java and built on top of the Jena Semantic Web Framework[4]. Jena provides the backend repository for both concepts (TBox) and instance data (ABox). Leveraging Jena's persistent storage support, the model storage can be based on files or a relational database. TBox management is centralized on the server to ensure consistency, quality of the ontology design, and standardization of the KB. Reasoning services are provided by Racer [14], which provides highly optimized TBox and ABox reasoning capabilities.

[4] http://jena.sourceforge.net/

The client provides the foundation for our visual integration, supporting the interaction and linking between the maintenance activities and the relevant resources. Similar to more traditional database applications, ABox management is provided through the client. The client also provides the story interface that is part of the local process manager. The contextual process views are established through separate process views and resource advisors. The story manager also integrates both the contextual navigation through the KB (through pre-defined queries), as well as the query interface for user defined queries.

6.2 Process Support

A successful process-centered support system is based on the premise that the right information should be made available at the right time in the right format. Achieving this premise depends greatly on the available knowledge resources and the specific task context. From a software maintainer's perspective, the challenge is specifically the need to adjust both information and context in real time to the current process task-step being performed. Our process environment supports such traceability between the involved process steps and other supporting resources (e.g., artifacts, maintainers, techniques, and tools).

The story (process) manager is built on top of the ontology manager infrastructure. The story manager provides for the user both the context and the interactive guidance during the evolution task. The story manager coordinates the protagonist (the user) to dynamically author the specific evolution task solving story. In addition, the story management is also responsible for providing an intuitive visual metaphor to vividly model the whole task solving process. The different aspects of the story metaphor can be mapped with the help of existing visualization metaphors (like static and dynamic 2D and 3D graphs, textual representations and different types of animation). During the process, the story manager is monitoring events and triggers that might affect the storyline of the evolution process, e.g., user or system activities.

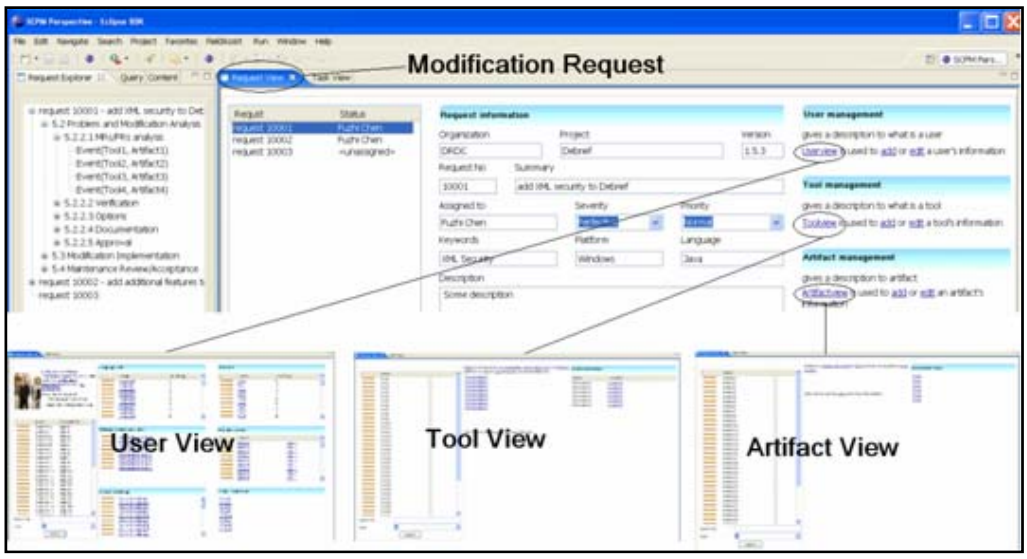

Fig. 10 Maintenance request and linked views

A software maintenance process typically originates from a MR. Similarly, our environment will initiate the maintenance process with a request view (Figure 10). The MR view also establishes the initial process context and the traceability between the process and the available resources. The context sensitivity levels are supported through customizable views and abstraction hierarchies that both support filtering and extracting details-on-demand [33]. The process supporting queries establish the link to the KB and automatically populate the relevant resource views. The process and its task-steps are supported through a workflow like view and a tree structure view, with both of them adopting similar techniques as the IBM Rational RMC [17]. Those views allow for easy process navigation by expanding/collapsing of process task-steps and their details. For a specific task-step, we also support context switching to the advisor views that provide suggestions on what relevant knowledge resources and other related task-steps might be applicable in the given context.

6.3 Initial Evaluation and Discussion

Our research contains two main contributions: The OWL-DL KB capturing both process and knowledge resources, as well as an environment utilizing the KB. The use of OWL-DL as the ontology language to formalize the KB provides additional benefits compared to less formal knowledge and information sharing approaches. Our constructed ontological representation can be used with many available OWL-DL tools such as Protégé[5] (ontology management), Pellet[6] (ontology reasoner), and SWOOP[7](ontology browsing/debugging). Our model design has been evaluated with Pellet and Racer [14] for model consistency and reasoning support. Management and evolution of the KB is supported by existing ontology management tools such as Protégé.

In addition, the visual tool environment we developed promotes the use and adaptation of process models and is not limited to maintenance process models. Our environment provides maintainers with process guidance without the need for a prior knowledge about formal ontologies or OWL. Maintainers can use and customize both the standard process as well as the traceability between the context relevant knowledge, through the different advisor views and pre-defined or customizable queries.

It has to be noted that one of the main challenges for our approach is the population of the KB. In this respect, our approach does not differ from any existing knowledge base or data mining approach. We are currently following two main strategies to address this challenge. In the first phase, we are performing several case studies in collaboration with Defence Research and Development Canada (DRDC) Valcartier. The case studies are performed in a controlled environment that restricts both the type of task performed (component substitution) and the resources to be modeled. This limited knowledge corpus is sufficient to provide an initial proof of concept that our ontological representation can capture the relevant information and model the context levels. To date, we have conducted

5 http://protege.stanford.edu/
6 http://pellet.owldl.com/
7 http://www.mindswap.org/2004/SWOOP/

two component substitution case studies on Debrief[8], a medium size open source application for the analysis and reporting of maritime exercises. Both of the maintenance tasks were related to substituting a non-secure XML data exchange component within Debrief by open source components that perform XML encryption.

Case study settings

We selected 8 graduate students from our Software Maintenance Research Group at Concordia University to participate in both case studies in a controlled environment. The technical and maintenance experience levels varied significantly among the students. Their Java programming experience ranged from 2 to 8 years and their industrial maintenance experience from no previous experience to several years as senior programmer. A commonality among all students participating in the case study was that they had no previous experience using or maintaining Debrief.

For the case studies, 30 software maintenance relevant tools were selected and installed on the computers in two software engineering labs accessible to the students. The software packages included source code analysis tools, software measurement tools, software visualization tools, and other software maintenance support tools. Information related to these tools was manually collected and populated within the KB. Since most of the students were unfamiliar with these tools, tutorials and manuals were provided to allow them to familiarize themselves with the tools prior to using them in their assigned maintenance task.

Artifacts that were made available with Debrief included source code, class files, a detailed analysis document, a system design document, user guide and tutorials.

The intent of the first case study was twofold: firstly to validate whether our unified model was capable to sufficiently and correctly capture the information required to support a software maintenance tasks. Secondly, the case study was used to collect some

[8] http://www.debrief.info/index.php

user, tool and feedback data from the case study to allow for an initial population of the ontology.

Given the results and feedbacks collected from the second case study, we were able to extend and refine our ontological model to improve its ability to support more task specific queries. The collected feedback was also used to further populate the ontology, which included after the second case study 295 concepts, 129 relationships and 765 instances.

The studies provided us with a proof of concept that our approach is capable of providing additional benefits to maintainers. It was in particular the integration of process, task and resource relevant information that was well received. Ontology population is a major concern at this point of time, since our process advisor support is driven by the quantity and quality of the information available. The collected feedback confirmed our initial hypothesis that considering associated knowledge resources without a particular maintenance context does not provide sufficient guidance to assist a maintainer (protagonist) in completing a given maintenance task.

During the second evaluation phase, we will further extended and generalize our environment by including additional tasks and resources to the KB. Currently, we are facing the challenge of ontology population and exploring different avenues to address this issue. Potential solutions for the ontology population problem include the creation of an Internet community portal and/or the concrete adoption of our tool within an industrial setting.

7. Related Work

As a knowledge representation language, OWL has already been applied in many applications of the software engineering domain such as model-driven software development [29], a CMMI-SW model representation and reasoning for classifying organizational maturity levels [41], reverse engineering tool integration [20], component reuse [16] and open source bug tracking [1]. However, there exists only limited research in modeling software maintenance and evolution using ontologies. Ruiz et al. [37] present a semi-

formal ontology based on REFSENO for managing software maintenance projects. They consider both the static [23] and dynamic aspects such as workflow in software maintenance processes. The ontology was constructed using KM-MANTIS, a knowledge management system with the goal of improving maintenance project management. However, no implementation or usage details are provided. Kitchenham et al. [23] designed a UML based ontology for software maintenance to identify and model factors that affect the results of empirical studies. Their goal differs from ours by utilizing ontologies to establish a common understanding context for empirical studies. Furthermore, the resulting ontology was not formally modeled and no reasoning services were used to infer implicit knowledge. Dias et al. [11] extended the work by Kitchenham by applying a first order logic to formalize knowledge involved in software maintenance. Although they stated that it is worthwhile to provide a knowledge base, they only identified knowledge relevant to the software maintenance domain without actually providing a concrete KB implementation. González-Pérez and Henderson-Sellers present a comprehensive ontology for software development [12] that includes a process sub-ontology modeling, among others, techniques, tasks, and workflows. The ontology is presented using UML diagrams. No implementation is given, implying that the authors also have not examined the integration of their ontology in an actual software development process.

Common to all of these approaches is that their main intent is to support in one form or another the conceptualization of knowledge, mainly by standardizing the terminology and to support knowledge sharing based on a common understanding. These approaches fall short on adopting and formalizing a process model to support traceability links between the process and the knowledge resources in the KB. They also lack the use of reasoning services to infer implicit knowledge. To the best of our knowledge, there exists no previous work that focuses on developing a formal OWL-DL software maintenance process model and utilizes automated reasoning to establish the links between process activities and involved resources.

The collaborative nature of software engineering has more recently been addressed by introducing Wiki systems into the SE process. Semantic Wiki extensions like Semantic

MediaWiki [53] or IkeWiki [54] add formal structuring and querying extensions based on RDF/OWL metadata. These works can be seen as complementary to our approach, in that they can support the creation and visualization of the developed knowledge base. However, by themselves they do not address the main concern covered by our approach, delivering active and context-sensitive guidance to an individual developer for his current task, based on knowledge both explicitly encoded and implicitly derived by actions performed by other developers.

Compared to other modeling approaches such as Model Driven Architecture (MDA) and related modeling standards (e.g., UML/MOF [58] or the Eclipse Modeling Framework (EMF) [59]) our approach differs fundamentally in its objective. (1) These modeling approaches focus on forward engineering, i.e., producing code from abstract, human-elaborated specifications, following a strict modeling process. We, on the other hand, focus on software evolution, where one has to extract, model, integrate and utilize knowledge from various and often not well defined resources. (2) MDA and EMF both focus on modeling well defined processes, in an attempt to derive complete models. Our approach on the other hand is based on an open world assumption, focusing on the integration of existing and newly gained knowledge and by making this knowledge available to the end-user.

With regard to tool support for process guidance, recent work by IBM Rational on its Method Composer (RMC) [17] and Process Advisor [39] are the closest related approaches to our research. RMC is a process management tool for authoring, configuring, and publishing development process like the Rational Unified Process (RUP). The IBM Rational Process Advisor integrates RUP process guidance within the context of software development tools [39]. Due to the required tool customization, the contextual guidance through the Process Advisor is currently only supported within a selected set of Rational tools. Our approach has similarities in process modeling and customization with the RMC,

due to the fact that we adopted parts of the Rational Method Composer, namely EPF[9] and GEF[10], to graphically model and customize the IEEE maintenance process. However, our approach differs in its overall motivation. These differences can be summarized as follows: (1) The Process Advisor focuses on the integration of RUP based process guidance into the context of several Rational development tools[11] by requiring a customization of these tools to support the Process Advisor. Our focus instead is on the general knowledge integration between process activities and resources by creating links between process activities and resources which are not limited by a set of specific tools. (2) We focus on knowledge integration rather than tool integration by providing a flexible and dynamic knowledge management. Within our approach, any newly gained knowledge (process and/or resources) automatically becomes an integrated part of our model. Furthermore, we support reasoning services that can infer implicit knowledge not explicitly modeled in the KB. In comparison, the Process Advisor is a more static approach that requires a manual adaptation of tools to be able to take advantage of the Process Advisor and its context sensitive guidance.

8. Conclusions and Future Work

Software evolution is a major part in terms of effort and cost involved in any software life cycle. Our work promotes the use of both formal ontology and automated reasoning in software evolution by providing a formal DL-based ontological representation for the modeling of both software evolution processes and resources relevant to support these processes. We have adopted the IEEE software maintenance standard to illustrate modeling of software evolution processes using a formal ontological representation. The unified ontological representation allows for the integration of these knowledge resources and

9 http://www.eclipse.org/epf/

10 http://www.eclipse.org/gef/

11 http://www-128.ibm.com/developerworks/rational/products/rup

establishes traceability links between the process and the resources. The flexibility and extensibility of the ontologies also enables the evolution and enrichment of the knowledge base. We introduce a story based software evolution process model to provide a metaphor that supports the interaction and context visualization between the process model, modeled resources, and users. The visual format for the task story model is still under development. As part of our future work, we will conduct further case studies to enrich our current ontologies and optimize our current tool in supporting particular software evolution process models.

Acknowledgement:

This research was partially funded by DRDC Valcartier (contract no. W7701-052936/001/QCL).

References

[1] Ankolekar, A., "*Supporting Online Problem –Solving Communities with the Semantic Web*", Ph.D.Thesis, Carnegie Mellon University, Pittsburgh, PA, 2005.

[2] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Schneider, P.P., "*The Description Logic Handbook",* Cambridge University Press, 2003.

[3] Baader F., Horrocks I., and Sattler U., *"Description logics as ontology languages for the semantic web*", In Dieter Hutter and Werner Stephan, editors, number 2605, Lecture Notes in Artificial Intelligence, pages 228-248. Springer, 2005.

[4] Ballim, A. and Wilks, Y., "*Artificial Believers: The Ascription of Belief*", Lawrence Erlbaum, 1991.

[5] Basili, V.R., "*Viewing Maintenance as Reuse Oriented Software Development*", IEEE Software, pp.19-25, 1990.

[6] Bennett, K.H. and Rajlich, V.T., "*Software Maintenance and Evolution: a Roadmap*", Proceedings. of the Conference on the Future of Software, 73-87, 2000.

[7] Berners-Lee T., Hendler J., and Lassila O., *"The Semantic Web",* Scientific American, 284 (5):34-43, 2001.

[8] Brooks, R., *"Towards a theory of the comprehension of computer programs",* International Journal of Man-Machine Studies, 18:543-554, 1983.

[9] Bruner, J., *"Acts of Meaning"*, Cambridge, MA: Harvard University Press, 1990.

[10] *CMMI for Development*. Version 1.2, Technical Report CMU/SEI-2006-TR-008, Carnegie Mellon, Software Engineering Institute, USA, 2006.

[11] Dias, M. G. B., Anquetil, N. and Oliveira, K. M., "Software Maintenance Ontology", chapter 5 in Caleor, C., Ruiz, F., and Piattini, M. (Eds.), Ontologies for Software Engineering and Software Maintenance, Springer, pp.153-173, 2006.

[12] Gershon, N., Page, W., *"What storytelling can do for information visualization"*, CACM 8(44), pp.31-37, 2001.

[13] Gruber, T. R.: *"A Translation Approach to Portable Ontology Specifications*", Knowledge Acquisition, 5(2):199-220, 1993.

[14] Haarslev, V. and Möller, R., "*RACER System Description*", In Proc. of Int. Joint Conference on Automated Reasoning (IJCAR'2001), Springer-Verlag, 701-705, 2001.

[15] Hartland, E.S., *"The science of fairy tales",* Walter Scott, 1891.

[16] Happel, H.-J., Korthaus, A., Seedorf, S., and P.Tomczyk, *"KOntoR: An Ontology-enabled Approach to Software Reuse*", 18th International. Conference on Software Engineering and Knowledge Engineering (SEKE), San Francisco, July, 2006.

[17] Hauner, P., "*IBM Rational Method Composer: Part 1: Key concepts*", IBM report, December, 2005.

[18] Horrocks, I., Patel-Schneider, P. F., and Harmelen, F., *" From SHIQ and RDF to OWL: The Making of a Web Ontology Language*", Journal of Web Semantics, 1(1):7-26, 2003.

[19] International Standard - ISO/IEC 14764 IEEE Std 14764-2006 Software Engineering, Software Life Cycle Processes, Maintenance, ISBN: 0-7381-4961-6, 2006.

[20] Jin, D. and Cordy, J. R., "Ontology-Based Software Analysis and Reengineering Tool Integration: The OASIS Service-Sharing Methodology", 21st IEEE International Conference on Software Engineering (ICSM), 2005.

[21] Johnson-Laird, P.N., *"Mental models: towards a cognitive science of language, inference and consciousness"*, Cambridge, Mass. Harvard University, 1983.

[22] Keller, I. M., Madachy, R. J., and Raffo, D. M., *"Software Process Simulation Modeling: Why? What? How?"*, Journal of Systems and Software, Vol.46, No.2/3, 1999.

[23] Kitchenham, B., Travassos, G.H., Mayrhauser, A.V., Niessink, F., Schneidewind, N.F., Singer, J., Takada S., Vehvilainen R., and Yang H., *"Towards an Ontology of Software Maintenance*", Journal of Software Maintenance and Practice, 11(6), 365-389, 1999.

[24] Kroll, P., Kruchten, P.,*"The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP"*,Addison Wesley, 2003.

[25] Letovsky, S., *"Cognitive processes in program comprehension"*, In Empirical Studies of Programmers, pp. 58-79, Ablex Publishing Corp. 1986.

[26] Mayhauser A.V., Vans A.M., *"Program Comprehension during Software Maintenance and Evolution"*, IEEE Computer, pp44-55, Aug.1995.

[27] Meng,W. , Rilling, J., Zhang, Y., Witte R., and Charland, P., *"An Ontological Software Comprehension Process Model*", 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM), 28-35, 2006.

[28] Niles, I. and Pease, *"A.Towards a Standard Upper Ontology*", Proc. of the 2nd Int. Conf. on Formal Ontology in Information System (FOIS), Maine, 2001.

[29] Ontological Driven Architectures and Potential Uses of the Semantic Web in Systems and SE, www.w3.org/2001/sw/BestPractices/SE/ODA/, (accessed May, 2007).

[30] Pacione, M.J., Roper, M., Wood, M., *"A Novel Software Visualisation Model to Support Software Comprehension",* 11th Working Conference on Reverse Engineering (WCRE 2004), pp.70-79, 2004.

[31] Phillips, M.A. and Huntley, C., *"Dramatic A New Theory of Story"*, 5th Edition, Screenplay Systems Inc., 2001.

[32] Plowman, L., Luckin, R., Laurillard, D., Stratfold, M., Taylor, J., *"Designing multimedia for learning: narrative guidance and narrative construction"*, Proc. of the SIGCHI conference on Human factors in computing systems: the CHI is limit, ACM Press, May, 1999.

[33] Rilling, J., Meng, W., Chen, F. and Charland, P., *"Software Visualization – A Process Perspective*", 4th IEEE International Workshop on VISSOFT, Banff, Canada, 2007.

[34] Riedl, M. O., Young R. M., *"An Intent-Driven Planner for Multi-Agent Story Generation"*, Proceedings of the 3rd International Conference on Autonomous Agents and Multi Agents Systems, July, 2004.

[35] Riedl, M.O., Yong, R.M., *"From Linear Story Generation to Branching Story Graphs"*, IEEE Computer Graphics and Applications, IEEE Computer Society, 2006.

[36] Riva C., *"Reverse Architecting: An Industrial Experience Report"*, Proceedings of 7th IEEE Working Conference on Reverse Engineering (WCRE 2000), pp. 42-52, 2000.

[37] Ruiz, F., Vizcaíno, A., Piattini, M., and García F., *"An Ontology for the Management of Software Maintenance Projects*", International Journal of Software Engineering and Knowledge Engineering, 14(3), 323-349, 2004.

[38] Shneiderman, B., *"Software Psychology: Human Factors in Computer and Information Systems"*, Winthrop Publishers, 1980.

[39] Smith, J., Popescu, D., and Bencomo, A., "*IBM Rational Process Advisor: Integrating the Software Development Process with IBM Rational Developer and Tester V7 tools*", IBM report, December, 2006.

[40] Sowa, J. F., "Knowledge Representation: Logical, Philosophical, and Computational Foundations", ISBN 0 534-94965-7, Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 2000.

[41] Soydan, G. H. and Kokar, M., *"*An OWL Ontology for Representing the CMMI-SW Model", Workshop on Semantic Web Enabled Software Engineering (SWESE), 2006.

[42] Storey, M.-A. D., *"Theories, Methods and Tools in Program Comprehension: Past, Present, and Future*", In Proceedings of 13th International Workshop on Program Comprehension (IWPC 2005), Missouri, USA, 181-191, 2005.

[43] Storey, M.-A. D., Sim, S.E., and Wong K., *"A Collaborative Demonstration of Reverse Engineering Tools*", ACM SIGAPP Applied Computing Review, Vol. 10, Issue 1, 18-25, 2002.

[44] Web Ontology Language, http://www.w3.org/2004/OWL/, (accessed: June, 2007).

[45] OWL Web Ontology Language Reference, W3C Recommendation, http://www.w3.org/TR/owl-ref (accessed June 2007).

[46] Wongthongtham, P., Chan,g E., Dillon, T.S., *"Towards "Ontology"-based Software Engineering for Multi-site Software Development"*, 2005 3rd IEEE Int. Conference on Industrial Informatics (INDIN), 2005.

[47] M.-A. D. Storey , K. Wong , H. A. Müller, How do program understanding tools affect how programmers understand programs?, Science of Computer Programming, v.36 n.2-3, p.183-207, March 2000[

[48] A.von Mayrhauser , A. Marie Vans, Program Comprehension During Software Maintenance and Evolution, Computer, v.28 n.8, p.44-55, August 1995

[49] M. M. Lehman , L. A. Belady, Program evolution: processes of software change, Academic Press Professional, Inc., San Diego, CA, 1985

[50] P. Wongthongtham, E. Chang, T.S. Dillon, I. Sommerville (2007) 'Software Engineering Ontology – Instance Knowledge Part I', International Journal of Computer Science and Network Security, USA

[51] K. Bontcheva, M. Sabou, Learning Ontologies from Software Artifacts: Exploring and Combining Multiple Sources, In *Proceedings of the 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006).*

[52] Software Engineering Coordinating Committee. SWEBOK, version 2004, http://www.swebok.org.

[53] M. Krötzsch, D. Vrandečić, and M. Völkel. Semantic MediaWiki. In I. Cruz, S. Decker, D. Allemang, C. Preist,D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, The Semantic Web – ISWC 2006, volume 4273 of LNCS, pages 935–942. Springer, 2006.

[54] S. Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In WETICE, pages 388–396. IEEE Computer Society, 2006.

[55] M. Greyck B. Dias, N. Anquetil, K. Marcal de Oliveira, "Organizing the knowledge used in Software Maintenance", Journal of Universal Computer Science, Vol.9, No.7, 2003

[56] Jens H. Jahnke and Andrew Walenstein "Reverse Engineering Tools as Media for Imperfect Knowledge,", IEEE Working Conference on Reverse Engineering, (WCRE'2000), pp 22-31.

[57] R. Pressman, " Software Engineering: A Practioner's Approach, 6th Edition, McGraw Hill College, ISBN-10:007301933X, 2005.[58] OMG – Unified Modeling Language; /www.uml.org/ (last accessed 2/2008)

[59] Eclipse and Open Development Platform, www.eclipse.org (last accessed 2/2008)

[60] Open Management Group (OMG) – Unified Modeling Language, www.uml.org (last accessed 2/2008)

[61] González-Pérez, C. and Henderson-Sellers, B., "An Ontology for Software Development Methodologies and Endeavours". Chapter 4 in Caleor, C., Ruiz, F.and Piattini, M. (Eds.), Ontologies for Software Engineering and Software Maintenance, Springer, pp. 123-151, 2006.