Supporting Collaborative Software Development by Context-Aware Information Retrieval Facilities

Andreas Henrich Karlheinz Morgenroth

University of Bayreuth
Department of Applied Computer Science
D-95440 Bayreuth, Germany
Email: {Henrich|Morgenroth}@uni-bayreuth.de

Abstract

The development of large software systems is a typical example for collaborative development efforts. Moreover, software development becomes more and more component-oriented. The source for these components can be either a component repository inside an organization or one of the emerging open source development networks available on the Internet. In this paper we describe an approach to search for potentially useful components during software development. Our approach is by no means restricted to components in an implementation oriented sense, but covers all types of artifacts created within a software development process. It is based on retrieval techniques for structured documents and uses contextual information about the user and his or her current work to refine the queries.

1 Introduction

Today everyone would agree that the reuse of previously developed artifacts can significantly improve productivity during software development. An open question in this respect is how to find well-suited previously developed artifacts in a concrete situation. In our opinion, customized information retrieval facilities can be useful for this purpose. The idea is that software developers — e.g. the software developers in a global company — open their local repositories for each other. In this way a software developer searching for a class can benefit from the artifacts developed by all software developers in the community. Another opportunity would be to exploit further sources for software artifacts like open source development networks.

The main problem in this scenario is to find appropriate artifacts, which are useful in the concrete situation. To this end, the structure and the relationships between the artifacts in the repositories and the working context of the software developer can be used. For example, the dependencies between requirements documents, analysis documents, design documents, code modules and test protocols will be represented in a repository. Given a software developer in charge for the design of a system, we can look for existing design documents developed for similar analysis documents and requirements. Furthermore, we can increase the precision of this similarity search enriching the similarity condition by knowledge about the current working context.

To become more concrete, let us consider the scenario sketched in figure 1. This figure depicts the current working situation of a software architect within a software development process, e.g. the unified software development process [5]. The software architect is currently working on a design document which implements the system defined in an analysis document which in turn depends on two requirements documents. On the other hand, there are the software artifacts from earlier projects given on the right side of figure 1. These artifacts are part of a document network representing the dependencies between the documents.

In this situation we want to assist the software architect as follows: Based on the documents building the specification for the design document under work (the three upper documents on the left side of figure 1), we perform a similarity query searching for similar requirements and analysis documents in the pool of documents from finished or proceeding projects (on the right side). From the requirements and analysis documents found in this way we can determine associated design and implementation documents. As a consequence, we can provide the software architect with documents ranging from requirements documents to implementation documents which might be helpful with his or her current task.

Furthermore, this scenario can be augmented in two directions: First, the pool containing the artifacts of finished

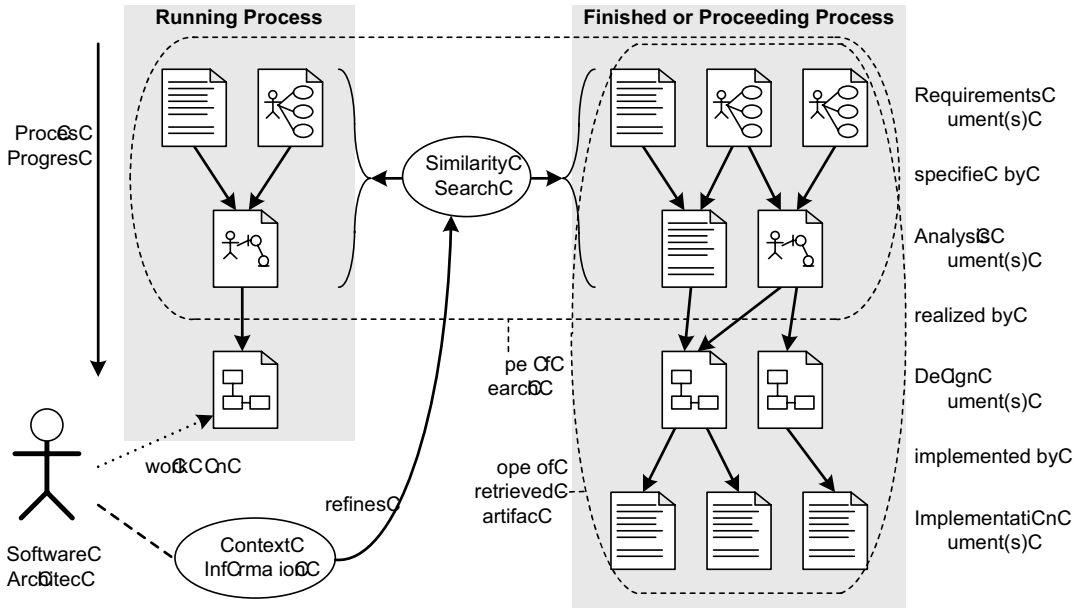

Figure 1. The scenario of searching reusable artifacts

or proceeding software projects has to be realized as a distributed pool containing artifacts of various users sharing the results of their work. This leads to a loosely coupled human cooperation over the Web and forms a platform supporting the remote cooperation of software developers. Second, it seems to be promising to improve the similarity query in our scenario by the additional consideration of information about the working context of our software architect. For example, information about his qualification or his role in the project team might give helpful hints for a more precise similarity search.

In the rest of this paper we will address the various aspects of the system sketched above in more detail. An important open point in this respect is similarity queries on structured documents (cf. section 2). Thereafter in section 3 we will elaborate on the integration of context knowledge into the similarity search and in section 4 we will sketch our prototype implementation. A discussion of related approaches (section 5) and an outlook on future research aspects (section 6) conclude the paper.

2 Information Retrieval for Structured Documents

As mentioned in the introduction, our approach to support the reuse of artifacts is based on retrieval facilities for structured documents and on the use of context information. The latter aspect will be discussed in section 3. The retrieval facilities for structured documents are based on the approach presented in [3]. The principle behind this approach can be explained by an example. Let us assume that we are searching for a reusable design document. In this case the analysis documents defining the functionality of the desired design document do already exist. These documents can be used as query documents and we can search for design documents in the pool with the artifacts of finished and ongoing projects which are connected to similar analysis documents. The search starts with the search for analysis documents in the pool which are similar to the query documents. This similarity search can e.g. be based on the textual similarity defined according to the vector space model [9]. Then the design documents connected to the best matching analysis documents can be presented as potential reuse candidates.

3 Exploiting the Context

As mentioned earlier, we use information about the context of a user to trigger and refine the search for similar artifacts. As a starting point for our context-aware retrieval engine we use a unified user model covering several dimensions from the user, his or her working context and the interaction with applications he or she is using [2]. Figure 2 illustrates our context model with its different context dimensions.

The *user context* comprises the physical and organizational context of the user as well as his or her personal profile. Considering a software development process the physical context contains e.g. geographical information about a user's working office. The organizational context represents

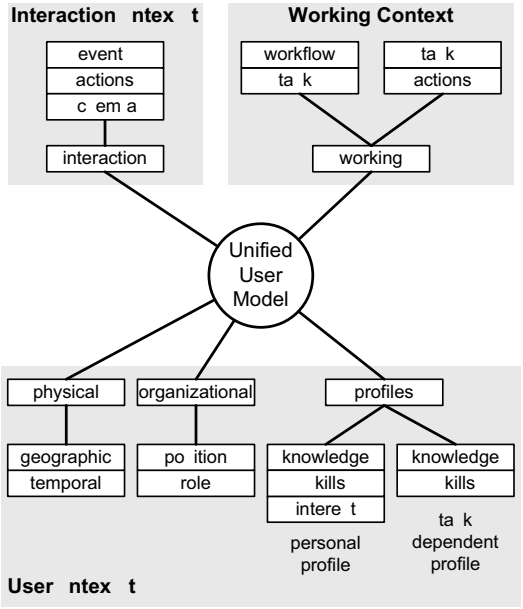

Figure 2. A unified user model for context information

the position a person holds within the organization as well as the roles he or she holds within the software development process. The roles a person fulfills within a process flow are an essential information about his or her responsibilities. Finally, a user profile can give information about a user's knowledge and skills.

The *working context* characterizes the current activity a person performs. During a software development process we can assume that activities are mostly planned and well described. Furthermore, there will be tasks that are unplanned and only vague information might be available.

Finally, a person's current interactions with the application systems supporting his or her present activities are reflected in the *interaction context*. Typically, these are events from menu and dialog interactions on a lower system level, e.g. creating a class diagram or editing a file of source code.

3.1 Representing the context information

To represent the context information of a user we distinguish the activities he or she performs or has performed and the artifacts that result from these activities. Both are represented as structured documents based on the extensible markup language (XML).

The dimensions within our user model that cover the user and working context as well as the interaction with applications are represented by a network of statements using the resource description framework (RDF). In this respect we can create a network of statements expressing relationships between different statements. Due to the definition of an RDF Schema with a naturally spoken language it is possible for us to transform statements given in RDF to term vectors used in the vector space model [9]. Applying an ontology to the vocabulary used in RDF statements allows to bring the concepts and hence the statements within the different dimensions of the user model in a semantic relationship to each other. Furthermore, applying an ontology is an appropriate way to expand the queries with related concepts (cf. [7]).

The resulting artifacts from activities, e.g. use case or class diagrams, are innately structured documents. Also textual descriptions enclose a structure, that separates each requirement and its description within a requirements documentation. Artifacts are stored in a wide range of document formats that comprise text processing formats as well as formats for diagrams and program code. Our approach transforms the different types of artifacts into XML, providing an XML Schema for each artifact type. Thus a formulation of queries for structured documents over different types of artifacts is feasible.

3.2 Obtaining context information from client applications

A modern software development process is unthinkable without the support of tools [5]. The functionality of these tools can typically be extended by plug-ins.

For example, using an integrated software development environment that supports UML modeling as well as implementing source code and testing, we can use a plug-in to infer the *User Context*, the *Working Context*, and the *User Interaction* (cf. section 4).

3.3 Obtaining context information from server applications

Besides the client application, valuable sources for obtaining information about the working context are data stored within server applications. This includes the repositories in which the model data for requirements, analysis and design as well as the source code of the implementation are stored. Within such repositories metadata including the date, the creator, and the contributors of each artifact is recorded. Retrieving the content from such repositories it is possible to reconstruct the responsibilities of people to their artifacts in finished or proceeding projects.

Using the data stored in an organizational directory, e.g. through LDAP, we can additionally gain information for the user context about the positions and roles a person holds within an organization. Combining this information with data from a groupware or project management server, we

can further gain detailed information about the activities a person was and is still involved in.

In our approach, we use this information in two ways. The data within the repositories is transformed into XML and stored in an index repository. The metadata from repositories, or a project management server is transformed into RDF statements and incorporated into the working context dimension of the user model.

In the prototype described in the next section the context information is used to restrict the query artifacts and to refine user queries. For the latter aspect query terms typical for the current user context are given more weight in the queries refined by means of the user context.

4 Prototype Implementation

Figure 3 illustrates the prototype for our approach to support collaborative software development by context-aware information retrieval. This prototype is still under construction but has already implemented the fundamental points of our approach.

As indicated in section 3.2, we use an integrated development environment (*Together Control Center*) that supports the requirements, design and analysis workflows with UML modeling as well as the implementation and testing workflows. This IDE allows the integration of plug-ins. Our plug-in monitors the following data and interaction between the user and the IDE: (1) The *project* a person is currently working on. From the metadata associated with this project the repository for model and code data as well as the user identification are extracted. (2) The *type of the activities* a person is currently performing. This includes the opened windows and diagram types allowing to conclude the type of workflow and activities a person is currently involved in. (3) The *interaction* between a person and the IDE comprising, e.g., the creation of new elements in a diagram or source code. This information is augmented by the data embedded in the elements a user is working on. For example, when a user is editing a class diagram the class documentation and the relationships between the classes are included.

To refine the assumptions about the users current activity a rule based system is used. For example, opened windows within the IDE containing analysis collaboration diagrams and the simultaneous work of the user on a class diagram lead to the assumption that he or she works within an activity of the design workflow. This contextual information is formulated as a series of RDF statements expressing the current working and interaction context. These statements are transmitted by the plug-in to the index and search server that incorporates them into the user model.

Within the index and search server there are two main repositories, holding the user model and the indexed artifacts retrieved from external repositories. These external repositories are connected by an indexer and connector component. The indexer component has the responsibility for retrieving and transforming data into XML according to a given schema or into RDF. A connector supplies a common access mechanism for the indexer to the data stored within the external repositories. For example, retrieving data from a CVS needs a different access strategy than querying an organizational directory through LDAP.

Within the query engine we use the approach for querying structured documents sketched in section 2. In our approach, the current working context is used to construct a query for the index repository finding similar artifacts resulting from similar working contexts according to the working context of the user and artifacts created by the user. In contrast to permanent queries for related artifacts the query engine starts only a query procedure when the working context has significantly changed.

Query results including short textual summarizations of an artifact are sent back to the plug-in within the IDE that displays them within a message pane at the bottom of the IDE main window.

5 Related Work

Our approach for supporting software developers with their work encompasses ideas from two main research areas.

First, there are different approaches to use information from the current working context of a user to gain information that might be useful.

The Remembrance Agent [8] represents a category of systems that use the text of a document or passages from the document a user is currently editing or reading as the context for the continuous retrieval of related documents from locally maintained document and email archives. Results will be presented to the user in a non-intrusive manner. The Lumière Project [4] provides an assistant for users working with an office software suite. The use of a Bayesian user model allows Lumière to derive a user's need from the actions a user performed within an application and to infer suggestions or help texts.

The second research area comprises the search for relevant components in repositories containing software components. The proposed approaches and systems differ in the utilized features extracted from the components. Text- and structure-based systems can be distinguished. Text-based approaches use either the description included within the component [10] or from an external documentation [6]. Structure-based systems, like [1], create a knowledge representation from the components within a repository.

Finally, a system named CodeBroker combining both research areas has been presented by Fischer and Ye [10]. This approach extracts documentation texts and method sig-

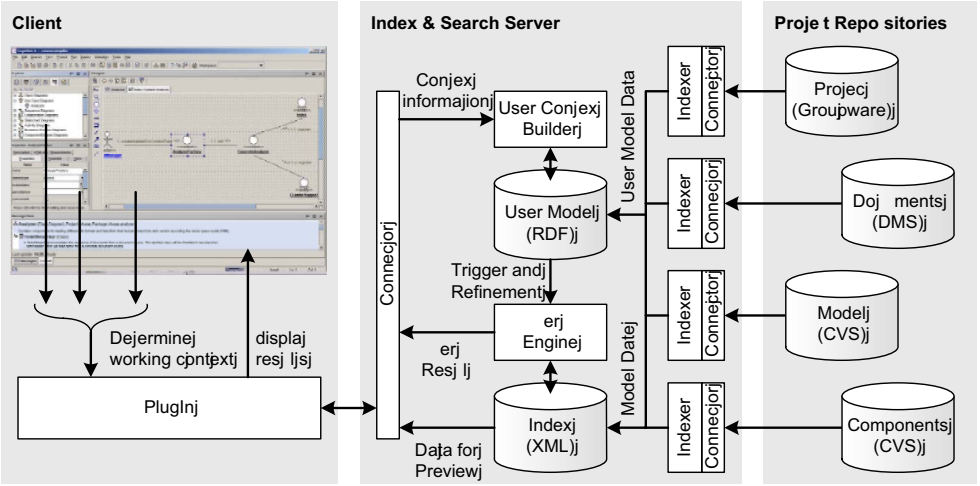

Figure 3. Prototype realization of a context-aware information retrieval system supporting collaborative software development

natures from the source code a programmer edits. Therewith, similar components from a component repository are searched.

The main differences between these approaches and systems and our approach can be seen in the following aspects: (1) The consideration of all artifacts created within a software development process. (2) The use of a wide range of user activities and interactions from an IDE. (3) The integration of different repositories and server applications to combine model and code data as well as information about the historical and current context of the user. (4) A user model using different contextual dimensions that will directly influence the retrieval of similar artifacts.

6 Future Work

In the present paper we have described our approach for a context-aware information retrieval system supporting collaborative software development. At present the integration of context information is based on retrieval techniques for structured text. In the future we will extend the approach to the complete range of UML diagrams. Furthermore, sophisticated techniques for the transformation of RDF-statements into similarity queries will be addressed and recall/precision tests will be performed to assess the benefits of a context-aware retrieval system.

References

[1] S. Henninger. An evolutionary approach to constructing effective software reuse repositories. *ACM Transactions on Software Engineering and Methodology*, 6(2):111–140, 1997.

[2] A. Henrich and K. Morgenroth. On the integration of context-aware information retrieval in portal systems (in german). In *Management der Mitarbeiter-Expertise in IT-Beratungsunternehmen (MKWI 2002)*, Nuremberg, Germany, 2002.

[3] A. Henrich and G. Robbert. An approach to transfer rankings during the search in structured documents (in german). In *Proc. 10. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web, BTW'03*, Leipzig, Germany, 2003.

[4] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence*, pages 256–265, Madison, WI, July 1998.

[5] I. Jacobson, G. Booch, and J. Rumbauch. *The Unified Software Development Process*. Addison-Wesley Object Technology Series, Reading, Massachusetts, 1999.

[6] Y. S. Maarek. An information retrieval approach for automatically constructing software library. *IEEE Transactions on Software Engineering*, 17(8):800–813, 1991.

[7] A. Maedche, M. Ehrig, S. Handschuh, R. Volz, and L. Stojanovic. Ontology-focused crawling of documents and relational metadata. In *Proceedings of the Eleventh International World Wide Web Conference WWW-2002*, Hawaii, USA, May 30 2002.

[8] B. J. Rhodes. *Just-In-Time Information Retrieval*. PhD thesis, MIT Media Laboratory, Cambridge, MA, May 2000.

[9] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Computer Science Series, New York, 1983.

[10] Y. Ye and G. Fischer. Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of 2002 International Conference on Software Engineering (ICSE'02)*, Orlando, Florida, USA, May 19–25 2002.