ARARA – A Collaborative Tool to Requirement Change Awareness

Ester J. C. de Lima¹, José A. Rodrigues Nt.¹, Geraldo B. Xexéo¹,², Jano M. de Souza¹,²
¹COPPE/UFRJ, Graduate School of Engineering
²DCC-IM Computer Science Department - Institute of Mathematics
Federal University of Rio de Janeiro, Brazil
{esterlima, rneto, xexeo, jano}@cos.ufrj.br

Abstract—This paper describes a collaborative tool, ARARA (Artifacts and Requirements Awareness Reinforcement Agents), that provides awareness in Software Development Process (SDP). Ontologies support this application for they are used by project team members when preparing the artifacts, propagating the concepts through them, and used by agents as knowledge base to retrieve information, tag the artifacts, detect changes, and search for linked artifacts that may be affected by a change on a previous artifact. This collaborative tool can improve team members awareness automatically no matter where they are geographically working, once agents are responsible for tracking and notifying requirements, or any other changes, to team members, including consequent candidates for modifications.

Keywords—tools for CSCW; awareness; agents; ontology; traceability

I. INTRODUCTION

The lack of traceability and communication demands that project members have full knowledge about artifact relations (be them explicit or implicit) and knowledge about other members’ work, i.e., what they are doing and when. Both types of knowledge are necessary for correctly introducing changes to the project, without compromising its integrity. If members are not aware of what others are doing, it is difficult to detect changes and their resulting implications. In nowadays, since project team members can be geographically distributed over a large area, this problem is even more relevant.

We have been working on a three-year project for an Oil Company, modeling and developing a system that assesses and reports the company reserves to the company board, regulatory agencies and stock markets. Since the beginning of the project, we knew that team members would change frequently and team size would increase throughout the development. These facts could lead to some weakness on member interaction, compromising the detection of changes and their resulting implications, e.g., the propagation of changes among distinct artifacts. In addition, since SDP is a cooperative work, communication is recognized as a critical task [5]. However, although SDP artifacts referred by the documents spawned by the Software Requirements Specification are all related, forward traceability is still a problem, as the relation among them are not always explicit and proper communication to team members is not always effective.

Based on these facts, we realized the need for a tool that could improve members awareness, reducing the impact of project artifacts changes. The size and complexity of the application, in terms of business coverage and capillarity, led us to design a collaborative tool to automate traceability in software engineering projects.

When the project started, we created a Business Ontology in parallel with the Business Process Model, to support and facilitate the development of the latter, and the System Design Model. Some of the concepts where new to a great part of the development team, and some terms had different meaning for different stakeholders, which justified the creation of an appropriate ontology. Cappelli [7] proposes ontology as a complement of business-process models, for domain understanding, to explicit and better understand the business, with the purpose of identifying software requirements through business requirements. Taking this into consideration, we extended the use of the ontology to all artifacts of the SDP. This way, all software artifacts were related to the proper business ontology concepts, to facilitate understanding, to ease communication among team members and stakeholders, and to promote requirements traceability.

Additionally, most SDP artifacts can be related using an adequate software engineering ontology or metamodel.

Considering all of the above, we decided to develop a SDP support tool, using ontologies, to promote the discovery of software artifacts relationships, mixing the knowledge offered by the business ontology and the software engineering ontology. Therefore, if an artifact is modified, the tool can improve awareness, providing project team members proper information about the set of artifacts that may have been affected by another artifact modification.

ARARA is based on agents that can track changes in artifacts and report to team members those changes and their effects on other artifacts. To allow for traceability information recovery, we use the available ontologies to prepare the artifacts [6]. We believe this is an effective way to link all the artifacts and to promote traceability among them.

The rationale is to prepare the system model using the associated business concepts extracted from the business ontology, tagging all UML artifacts produced (activity diagrams, use-case diagrams, class diagrams, etc). Later on, the
tool can benefit from the business ontology, to retrieve the artifacts that are connected to the one that was changed, navigating it and finding related business concepts, as well as using the software engineering ontology to find related artifacts.

The paper is organized as follows. In section II we introduce traceability in SDP. In section III we provide a basic explanation of ARARA. In section IV we discuss the principles and mechanisms of ARARA and in Section V we present the results we achieved. Finally, section VI presents the state of our work, next steps and offers some conclusions.

II. TRACEABILITY IN SOFTWARE DEVELOPMENT PROCESS

Traceability is one of the IEEE Recommended Practice for Software Requirements Specifications (SRS) [8]. Gotel [9] defines requirement traceability as the ability to describe and follow the life of a requirement, in both forward and backward directions, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases.

Edwards [10] defines traceability as a technique used to provide a relationship between the requirements, the design, and the final implementation of a system. During design, traceability allows designers and maintainers to keep track of what happens when a change request is implemented, throwing light on what shall be redesigned.

Palmer [11] states that traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design, and implementation. Watkins [12] found that traceability aids project managers in verification, cost reduction, accountability, and change management. For each change, it is easy to determine which related elements of the design are affected. This helps to keep documentation up to date as the implementation progresses.

Several issues make artifacts’ links recovery difficult. The connection between artifacts is rarely represented explicitly and the artifacts themselves are represented at different abstraction levels and using diverse formalisms. Once SDP produces a large set of artifacts, which is composed by different types of documents, and as one artifact is derived from others, it is important to link them to ease the traceability process, and it is, at least, desirable to adopt automatic techniques to recover those links. Manual recovery can be prohibitiv for a large set of artifacts.

There are other works focusing on the recovery of traceability links between software documentation and source code. In [13], “the methodology is based on the extraction, analysis, and mathematical representation of the comments and identifiers from the source code.” In [14], “the method uses the identifiers extracted from source code component as a query to retrieve the documents relevant to the component, assuming that programmers use meaningful names for their identifiers and consequently “words” are chosen as indexing feature.

ARARA uses ontology to provide a vocabulary to represent a Business domain and to serve as identifiers to be used in all artifacts. Object-Oriented design of software systems is somewhat similar to domain ontology, due to existence of objects, their attributes, and relations among them. It mirrors aspects of the domain that are relevant to the application [2]. In this case, using the ontology (vocabulary, concepts, relations, etc) it is possible to link artifacts of an SDP and better recover traceability between them.

III. ARARA CONCEPTION

We implemented ARARA to be deployed on a project we have been working, where requirements constantly change and lack of awareness is a fact, as members of the development team are all dispersed geographically within the company and team changes in size and constituents during the project, with professionals coming and going.

As artifacts are always in constant change in SDP, team members must be aware of these changes in order to complete their tasks and keep project artifacts complete and consistent.

ARARA communicates to team members the SDP artifacts that may be affected by the change done by someone in another specific artifact. As team members are notified by ARARA about artifacts that shall be verified, we reduce the chance of inconsistencies.

Figure 1 illustrates the behavior of ARARA’s agents. As a collaborative tool, ARARA’s basic tasks are:

- Detect changes on artifacts;
- Identify other artifacts that shall be changed;
- Identify to whom a change should be reported to; and
- Notify proper project team members.

In our setting, all artifacts of the project are stored on a version control repository. Therefore, whenever an artifact is written on the repository a set of Event-Detection Agents are responsible to detect this event and identify the change. These agents then communicate with another set of agents, called Information Extraction Agents, which are composed by three subsets of agents: Tag Agents, Change Agents and Responsibility Agents. They are responsible to analyze the changes, track the artifacts that may be affected due to the change initially detected and find whom to notify.

Project Knowledge Base is composed by references to all the artifacts of the project together with tags regarding their relation to the concepts defined on the Business Knowledge Base.

After the Information Extraction Agents have done their job, they communicate with the Awareness Agents that shall report team members about the detected change and the artifacts that may be also affected.
IV. PRINCIPLES AND MECHANISMS

A. Ontology Supporting Traceability

In this work we propose to apply the ontology, created during the business modeling process, to the artifacts produced in the system development. The idea is to link the artifacts by using ontology and to use ontology as a knowledge base to provide ARARA with automatic tracing capabilities.

One of the best practices for traceability recommended by [1] is to create traceable artifacts using a well defined project glossary to construct a meaningful hierarchy of information. All these can be achieved using ontology. Oliveira [3] uses ontology to promote software-tool integration. Whenever we have an ontology that is derived from a business process model, the concepts mapped onto the ontology can be used in all the artifacts.

Edgington [4] points out that one goal of ontology is to provide a faithful specification of a knowledge unit, i.e., that the resulting conceptual model adequately represents the domain or contextual knowledge of interest. Thereby, ontology is one way to clarify knowledge structure and to formalize it, to represent adequately the domain, and to enable better communication between humans and machines.

Contextualizing, as we orient system development to business process models, all the concepts mapped onto the business ontology were used on the artifacts. We have tried not to abbreviate the ontology terms and not to suppress the use of them. In model building, we follow standard conventions of uppercase and lowercase usage to facilitate the algorithm that identifies tokens and compare them with ontology terms, e.g., an interface of a component of the system architecture, “ICreateDissociatedField” that allows for the identification of the ontology Class “Field” and the Data Type Property “Dissociated”.

With all the artifacts prepared using ontology terms and based on those concepts, we support traceability through tagging the artifacts and searching for artifacts that are related, by having some similar tags and tags that are linked by ontology, and consequently may have been affected.

B. Tagging the Artifact – the Project Knowledge Base

Each artifact, after it has been written on the repository, is retrieved by a Tag Agent to create a representation of the artifact on the Project Knowledge Base. The process consists of:

- Searching ontology terms on it, creating Business Tags set that best represents this artifact;
- Defining which type of UML diagram or element the artifact represents; and
- Identifying the team member that committed the artifact’s change to the repository.

Such process uses the concepts mapped on the Business Ontology to create the Business Tags:

- Ontology Classes – used to match the artifact and form the conjunction of Business Tags that represent it (example of Ontology Classes: Field, Project).
- Object Properties – do not appear explicitly in the artifacts. Represent the association between ontology’s classes, linking an individual to another individual.1

1 Individuals can be referred to as being instances of classes.
Object Properties are used by ARARA to relate concepts in artifacts that are not explicitly linked but have an implicit link provided by ontology (e.g., Field has Projects or Project has EconomicValues). So, artifacts that deal with Field are in some way related to the ones that deal with Project.

- Datatype Properties – are the attributes of Ontology’s Classes. As Datatype Properties link an individual to an XML Schema Datatype value or an RDF literal [15], it contributes to find the Classes that can represent that artifact (e.g., Field has exactly 1 partner and can be dissociated or Project has exactly 1 description and can be dissociated).

During the retrieving process, the Tag Agents remove stop words on the artifacts and stem the rest of the words to their root form to get the ontology term.

As we are using a CASE tool – RSA (Rational Software Architect) [16] – to model the system, ARARA has to parse its XMI files (XML Model Interchange) [17] to get elements that compose the artifacts and to search them for Business Tags.

The XMI files themselves provide us the type of UML Diagram being described. We limited our scope to Activity diagrams, Use-case diagrams, Class diagrams, and Component diagrams. They also provide a mechanism for specifying references within and across documents, but we did not use it in our setting. This mechanism uses the XML href attribute to locate XML elements in another XML document by its XMI id. The value of href is a URI reference in the form URI#id_value, where URI locates the XML file containing the XML element to link to, and id_value is the value of the XML element’s XMI id [17].

This mechanism helps link artifacts, but it does not link artifacts that are in the same context and do not have a reference within and across documents. On the other hand, with ontology, we can retrieve all artifacts that represent the same concepts or are strongly related to it.

In the Version Control Tool Repository that we use in the project – SubVersion [18] –, the svn log command provides records of who made changes to a file or directory, in which revision it was changed, the time and date of the revision, and – if it was provided – the log message that accompanied the version commit. We infer the responsible for an artifact as the one that made the last change on the artifact.

Before the Tag Agent substitutes the new representation created for an artifact changed on the Project Knowledge Base, the agent communicates with the Change Agents.

C. Searching for Artifacts that may be Affected

Change Agents compare two representations of an artifact to find the differences between them. It also uses the information that SubVersion makes available to discover if the artifact committed on the repository is an inclusion, a deletion, or a modification. Change Agents treat each action in a different way.

- Inclusion – as the artifact is new it does not have a previous representation of it on the Project Knowledge Base. Therefore, the agents need only to find the group of artifacts that are linked to it by the same business context (artifacts that represent the same concepts and are strongly related to it).
- Deletion – as the artifact is not anymore part of the project repository, the agents use the last representation of this artifact to find the group of artifacts that was linked to it by the same business context.
- Modification – in this case, the agents need to compare the two representations of the artifact on the Project Knowledge Base (the previous and the new) to check: 1) if the set of Business Tags continues the same; or 2) if there are new Business Tags or if some are not anymore Business Tags that represent the artifact. In the first case, the agents need to find the group of other artifacts that are linked by the same business context to the changed artifact. In the second case, the agents need to find the new group of artifacts that are linked to it and find the other group that was linked to it before.

To find the artifacts that are linked by the same business context to a changed one is to search for artifacts that may be affected by the change. To get these artifacts we use the Jaccard Similarity Coefficient [19] to measure similarity between the artifact that has been changed and all others that compose the Project Knowledge Base. Jaccard Similarity Coefficient definition is:

\[
JS(\text{ArtfTag}_1, \text{ArtfTag}_2) = \frac{|\text{ArtfTag}_1 \cap \text{ArtfTag}_2|}{|\text{ArtfTag}_1 \cup \text{ArtfTag}_2|},
\]

\(JS\) represents the Jaccard’s similarity between artifacts; \(\text{ArtfTag}_i\) – stands for the set of business tags of an artifact \(T(\text{Artf})\).

\[\text{ArtfTag}_T = \{BT_1, BT_2, ..., BT_n\}; \text{ and} \]

\(BT_i\) – stands for the \(i^{th}\) business tag that represents an artifact on the Project Knowledge Base.

We rank the results, based on artifacts’ similarity, starting with the artifacts that are more similar to the changed one.

D. Agents Supporting ARARA

ARARA is a P2P application that is being implemented on the COPPEER framework, which is a multi-Agent System (MAS) framework, i.e., an environment for developing and running agent-based collaborative P2P applications [20], and that has successfully been used on the implementation of other collaborative tools like FoxPeer [21] and BPCE [22].

Although COPPEER technology supports any type of strategy for implementing agents, our approach is to implement single function and specialized agents that cooperate through entry exchange, which should be uncoupled.

ARARA agents can directly communicate with the artifacts’ repository through the SubVersion API, after being...
started by a repository hook. A repository hook is a program triggered by the occurrence of an appropriate repository event, e.g., a commit event.

The hook used to start ARARA process is called post-commit hook, because it runs after the completion of a commit event and it hands enough information about the repository changes [17].

V. ARARA RESULTS

We have been also experimenting the best way to list the artifacts to notify team members. Table 1 presents the ranking of the artifacts that may be affected by changes on the artifacts Artf1 and Artf2.

The column named $JS(\text{ArtfTag}_1, \text{ArtfTag}_x)$ shows the Jaccard’s Similarity Coefficient between Artf1 and the artifacts listed on column named RelatedArtifacts, which shows the artifacts related to the one changed. The next two columns present the same, but for Artf2.

We can see in the results presented above that a propagation of the change on Artf1 to Artf2 leads us to rank another set of artifacts candidate to be modified. Some of the artifacts listed on the second rank are also listed in the first rank, e.g., the set of artifacts (Artf5, Artf6, Artf7 and Artf8).

Comparing with the first rank these artifacts are less similar to Artf1 then to Artf2. In this particular case, maybe it’s not necessary to propagate changes on Artf1 to Artf5, Artf6, Artf7 and Artf8. It’s possible to wait till propagation changes on Artf1 occur on Artf2. Since Artf2 are consistent after the modification reflecting changes on Artf1, changes on Artf5, Artf6, Artf7 and Artf8 will reflect changes on Artf1 and Artf2.

Considering this specific case, it is not necessary to rank all artifacts related to the one that has been changed. The process of propagating modifications covers all the artifacts and we avoid information overload. Suppressing from the first ranking the artifacts Artf1 to Artf10 would not impact on the process of propagating changes.

In this work, the feedback of team members is necessary to evaluate the implemented algorithm. We are now conducting experiments with team members to get their view of:

- The artifacts they consider shall be prioritized when propagating change;
- The ones that they consider necessary to propagate change, but in a second moment (after changes are made to the ones initially prioritized);
- And the ones that they consider not necessary to propagate change;

VI. CONCLUSION

ARARA is a system to be used in software development projects where requirements constantly change; the development team is dispersed geographically, new team members arrive during the project; and awareness improvement is desired.

It provides awareness of artifact and requirement changes, based on knowledge bases, constructed as ontologies. A Business ontology is useful in this context, as it provides a business vocabulary and the relations among the represented concepts. This business vocabulary can be used by all artifacts produced in a software development process, linking them to each other and supporting traceability.

We intend to deploy ARARA in this large development project, where actually the need for a collaborative tool to notify requirement changes exists. This project will allow us to evaluate the effectiveness of ARARA.

The first phase of this project was conducted without the support of ARARA, and it is a fact that propagation of an artifact change was delayed more than expected or even suppressed, time is lost gathering information, and despite the team efforts, it is usual to find inconsistent artifacts, which demands rework. Delays on the schedule are also common.

The second phase of this project will be supported by ARARA and a comparison in response time between these two phases will measure improvement in work productivity, since we expect not have artifacts inconsistent though the constantly changes during SDP. This second phase will also provide us feedback about information overload. Although ARARA provides awareness on requirements/design changes, it is about the team members the responsibility to analyze the artifacts suggested to be changed and to do the necessary changes on the artifacts that need to be changed, ensuring the consistence of the requirements. This feedback will allow for the improvement of ARARA.

Although ARARA is a collaborative tool, we focused on the implementation of the algorithm to rank the artifacts that are more susceptible to have been impacted by the detected change. We are presently implementing the agents that automate the process.

The first prototype that was implemented was to evaluate the use of ontology in support of information retrieval techniques. ARARA development efforts are currently focused on ranking the artifacts that may be affected. We still need more time to refine the mechanism and define the actual algorithm that is to be used.
REFERENCES


