Towards an Effective Context-Aware
Proactive Asset Search and Retrieval Tool

Eduardo C. R. Santos², Frederico A. Durão¹, Alexandre Martins²,
Rodrigo Mendes¹,², Cásio Melo¹,², Bruno Melo³, Vinícius C. García¹,²,
Eduardo S. de Almeida¹,², Sílvio R. L. Meira¹

¹Informatic Center – Federal University of Pernambuco (UFPE)
²C.E.S.A.R. - Recife Center for Advanced Studies and Systems
³University of Pernambuco (UPE)
{ecrs, acm2}@cesar.org.br, {fad, rm, cam, vcg, esa2, srlm}@cin.ufpe.br

Abstract. “by improving the computers access to context, we increase the richness of communication in human-computer interaction and make it possible to produce more useful computational services” [7]. In this paper we present an implementation proposal of a proactive context-aware asset search and retrieval tool focusing in reducing the problem of ”no attempt to use” identified by [9] by reducing the user effort of query formulation.

1. Introduction

The RiSE¹ (Reuse in software Engineering) framework [4] has approached the software reuse area, proposing a framework for systematic software reuse adoption. This framework has been validated in many areas like software component certification [1], software reuse process [5], software reuse metrics [19] [18], domain analysis [6], software architecture evaluation [3] and software component search and retrieval [17][12].

On software component search and retrieval we recently made a survey of the area [17]. This work identified some directions and the state-of-the-art. Based on these directions, we specified and implemented a software component search engine called MARACATU [11] involving features such as keyword and facets search. In addition the tool was improved and an evaluation experiment was performed [12]. Moreover, a new folksonomy module was implemented and also evaluated [23]. However, all the previous versions have considered the search as a passive mechanism which the software engineer should perform specific queries.

On the other hand, as stated in [17], an efficient search engine should consider among other requirements the active search or proactive one. In this context, a first proactive search approach was proposed in [18] but this work did not focused in an important requirement as context information. Moreover the MARACATU search engine evolved into an industrial project called B.A.R.T. (Basic Asset Retrieval Tool), and its architecture was revised and evaluated using a systematic architecture evaluation process [3].

These previous works have successfully achieved its goals. However, none of them focused in context information to improve the search. Thus, as part of the RiSE

¹http://www.rise.com.br
framework, in this paper we propose a proactive approach adding context information to the B.A.R.T. project to reduce the problem of “no attempt to use” identified by [9], removing the user effort of query formulation.

In the following sections we present a brief background on reuse and component search and retrieval. Next, we describe the new approach with its requirements and its related architecture. At the end, we discuss some related works and present the conclusion and future directions.

2. Background
According to [15] "Software reuse is the process of using existing software artifacts rather than building them from scratch". The main motivation for that is to improve software quality and productivity [10] and reduce the required effort and cost [15] to build new software. The idea is not new, since it has been discussed since [20], but the problems related to develop a mass production software industry were not resolved yet.

Although, systematic reuse is not achieved only with tools. [10] presents some critical factors as management, measurement, legal and economic factors related to successful systematic reuse. However, tools are still important to support the process and currently there is a lack of efficient ones to properly locate potential code for reuse [8]. With the increase of software development projects and efforts in information technology the problems of information management and information retrieval, have been applied to software industry. Software repositories are now used and in order to deal with the huge amount of information search and retrieval tools must focus on solving the problem of component search and retrieval such as users who are not able to exactly understand the problem, or encode it in the query language to retrieve the exact components they need or want [21] [17].

3. Towards an Effective Context-Aware Proactive Asset Search and Retrieval Tool
In the early nineties, [16] proposed a work on information retrieval based on user activities. Their project was developed in the Xerox EuroPARK and the problems that they identified like understanding the users needs for retrieval tools and safeguarding the users privacy have not been resolved yet. Much of the researches about context-aware application are related to mobile application and ubiquitous computing. We propose the use of these techniques so improve the component search and retrieval experience.

3.1. Terminology
Some concepts might be defined to clarify some components of this work. One of them is the concept of a software asset. In order to give a broader semantic view of what could be searched and retrieved by the proposed tool we exchanged the term software component, which in many works is usually focused on source code units, by the term software asset whose concept includes not only source code classes or packages but also others artifacts of the software development cycle like requirements documents or use case diagrams.

Every asset will have its information content, which will be used to solve the user problem, but it might also have data descriptors or meta-data related to it. Thus, we will call it fields.
Another conceptual component of the solution is the **user**. He will be very important when we related information as history or personal queries. The user, here, is the person who is developing new artifacts of the software development cycle and might be granted with existing assets to help him to improve his work, be more productive, make things with better quality, improving software quality and productivity [10] or even reducing the required effort and cost [15].

A search application might be **interactive** or **proactive** [2]. On the first, the user must interact with the system and directly issue a request to retrieve information. The second, which is the focus of this work, presents information automatically to the user, using context information that is explained next.

Dey [7] says that while people tacitly understand the definition of **context**, they usually find it hard to elucidate. Context in this work is the information that might represent the state (related properties on time), activities (related actions in time) and situation (state and activities related) of a given user.

### 3.2. Requirements

The basic requirements for a search engine were previously describe in [11] and as an improvement of the this work, we started from those requirements: High recall and precision, security, query formulation, performance, interoperability, performance and others. Additionally, we propose new requirements focused on context-aware application and proactive search and retrieval:

1. The most basic requirement for a context-aware application is the concept of a **sensor**. The sensor, which might be physical or logical, is responsible for grabbing the information that may be interpreted as contextual data.
2. Historical information must be stored. This information will be used to predict future actions or intentions of users [7].
3. Context information must be gathered and stored to allow subsequent information retrieval [22]. This can be achieved by analyzing the current situation the user is in, comparing it with available information, and provide the user with information that is of most value in this situation [13].
4. To be more effective, the system must consider not only the context information of the consumer, but also the production side [13].
5. Context information must be explicitly treated, so it can be compared, used in queries ranked or to have a similarity measure applied to it. Additional retrieval strategies can be provided that are based on combinations of content-based and context-based queries [13]. This may lead to a multi-stage retrieval process and proposed by [2], where partial context information may act as a filter to remove documents for further consideration.
6. The context data might be automatically grabbed by system sensors, but the user should be allowed to explicitly provide this information [13].
7. In order to improve context matching, and thereafter its retrieval, the system must be able to compare partial parts of the contexts [13].
8. Historical information must be stored. This information will be used to predict future actions or intentions of users [7].
9. A context interpreter must be considered [7]. It will read one or more context in order to produce a unified view of information. i.e. the system will get information from local activities and merge with remote security activities to define the actual context.

10. A context aggregator [7] should be responsible for the manipulation of multiple contexts. It is the source of information for the context interpreter.

11. Not only technical issues must be considered. [13] proposes that the modeling effort for it and maintaining context models should clearly pay off in terms of improved access to information and increased working efficiency.

12. Finally, the information must be retrieved while the user is in the specific context searched and matched or the information retrieved might become irrelevant or obsolete. [13].

13. Brown [2] proposes that the criteria for proactive retrieval request will be any change in the user’s context, or in the collection of stored documents. For performance reasons, we will only consider the first option. Additionally, our focus is only information retrieval, not information filtering.

3.3. B.A.R.T. Architecture

In this section we describe our architecture previously seen in [3] and present how this solution must evolve to achieve the requirements listed in the previous section.

![Figure 1. B.A.R.T. Architecture](image)

- **Administration UI** - In order to provide administration access to the server, the architecture now supports an administration user interface, which may be implemented as a web layer or using the Java Management eXtension (JMX) specification;
- **B.A.R.T. Server** - The B.A.R.T. Server might be used in cluster and the Message Bus (described below) works as a message router with load balancing purposes routing messages for the multiple B.A.R.T. Server instances, which improves the search capability of dealing with heavy requests;
- **Embedded Database** - An embedded database to keep data like the user information and search log, will be used for reports. Although, the idea of using an embedded database is to reduce the solution installation complexity;
- **Log File** - A system log file for information like server startup and shutdown will be also developed;
- **Message Bus** - The message bus is responsible for the message routing in the system and to give support to synchronous and asynchronous messages and for the connectivity using multiple clients like Java, RMI, C, and web services. This module works with a rules engine to aid in the flexibility of defining new routing rules for the module;
- **Repository Manager** - It is responsible for the connection to remote asset repositories like Subversion and CVS to extract and index the information. B.A.R.T. Server might have multiple instances to work as a cluster to improve the search capability. Despite that, the repository manager has a single instance because downloading assets from remote repositories and indexing them does not need distributed instances of the Repository Manager;
- **Sandbox** - This is the local folder where the repository manager downloads the assets from the remote repositories;
- **Scheduler** - The scheduler executes scheduled jobs, like periodically downloading and indexing repository content;
- **Search Index** - The index is used by the B.A.R.T. server to search for assets. Multiple instances of the server accesses one centralized index;
- **Search UI** - A module embedded in the user environment to make search capable. From its usual work environment like Microsoft Windows or a development platform like Eclipse;
- **Security Server** - This new version now deals with security issues like authentication and authorization and limits access to users depending on specific groups created by an administrator;
- **Software Configuration Management Client** - A configuration management system client like WinCVS or Eclipse This element is here to represent external applications that might input information in the configuration management system;
- **Software Configuration Management System** - The remote repository with assets saved like CVS or Subversion. This is the source of information where the for the B.A.R.T. system.

This architecture is the basis for the search engine, it might be used in a interactive search engine, but to support proactive search, it needs more components, which are shown in the next section.

### 3.4. Adding Context

According to [2] to extract useful data from the user environment the system might use two different ways. In first one the information might be set by the user, while in the second option the information might be composed of data from one or multiple sensors.

- **Listener** - Referred by [22] as the activity widget this client side sensor is used to identify the relevant information to be used by the query formulator. A group of client listeners might be used to get information the user project, user activities, language or tool he/she is using at the moment and one listener for changes in the
historic information stored about that specific user or about the enterprise. Server
listeners are out of the scope of this work;

- **Query Formulator** - The query formulator is responsible for structuring the context
  information and data to request data from the server. It is also responsible for
  performance issues, determining the right moment to make those requests. This is
  the key component to minimize the user effort of query formulation.

- An **context aggregator** - Should be responsible for the manipulation of multiple
  contexts. This component should be capable of getting information from multiple
  sensors. This is important because some rules might be defined where the
  information from one sensor might be used as a filter for activation or inactivation
  of another sensor. i.e. the user environment sensor must identify that the user is
  working with use case documents and source code sensors will not be useful.

- **Context Fields** - The context fields are the contextual information added manually
  or automatically to the assets in order to improve its semantic representation i.e.
  information about the user who created the asset, about its dependencies, related
  technologies and activities performed by the user at the moment the query was
  formulated.

- **Context Enriched Asset** - The asset it self. They might be related to specific context
  fields, but may also be related to pre-defined contexts i.e. a asset must be related
  to a pre-defined

- **Presenter** - In order to avoid interrupting the user activity with many search results
  every moment, the presenter will identify the best momento to do so i.e. an user
  writing a specific Java method signature. The presenter would wait until the user
  finishes the writing the signature, before it starts a new search requisition.

- **Context Matcher** - At a given moment the system will need to match the con-
  text of the user with the contexts or assets stored. The context matcher will take
  care of this task making the needed matches i.e. compare the actual user context
  "user writing a non-functional security requirement" with existing assets related
  to security, or with non-functional requirements.

- **User History Base** - This component is responsible for storing user activities,
  searches and also references to assets or groups of assets previously known by
  the user. Therefore, as proposed by [25] well known assets the user usually deals
  with, will not be displayed every time i.e. the system may save in the user history
  base that he usually ignores the search results related to assets of a given project
  even if they are related to his actual. In this case the user may know well those
  components.

These are the basic components needed for a first proactive context-aware version
of the B.A.R.T. project. They will be implemented and inserted in the architecture
previously architecture presented.

#### 4. Related Work

- **CodeBroker [24]** - The first project to propose a proactive approach to search and
  retrieval focusing not only in technical issues but also cognitive ones.

- **Strathcona [14]** - Strathcona is not a proactive search engine, but Holmes used the
  concept of structural context, based on java source code relations and dependen-
  cies to define relationship between the user activity and the source content stored
  in a repository.
• **ADMIRE** [18] - Another important contribution by a RiSE group member was the Asset Development Metrics-based Integrated Reuse Environment (ADMIRE). This work proposed software reuse metrics as the Reuse potential and also validated these metrics implementing our first proactive search and retrieval project. ADMIRE focused in helping developers and managers.

5. Conclusion

We presented an implementation proposal of a proactive context-aware asset search and retrieval tool. This approach is an evolution of previous works [11][12][3] and part of the RiSE framework [4] and different from other related projects described, it works not only with java source code, but with the concept of generic assets from different parts of the software development lifecycle. We believe that this solution will improve the user experience because "by improving the computers access to context, we increase the richness of communication in human-computer interaction and make it possible to produce more useful computational services" [7]. Moreover, the context information we use proactive search to reduce the problem of "no attempt to use" identified by [9], in order to reduce the user effort of query formulation.

6. Future Work

As a future work we intend to implement and evaluate the proposed solution and to compare it to the previous works and analyze if the context-based query formulation might increase the search recall and precision.

References


