



IST PROJECT 35111

Tightening knowledge sharing in distributed software communities by applying semantic technologies

Project Number:	35111
Project Acronym:	TEAM
Project Title:	Tightening knowledge sharing in distributed software communities by applying semantic technologies
Instrument:	STREP
Thematic Priority:	Information Society Technologies (IST)
Start date of the project:	September 1 st , 2006
Duration:	30 months

D22: Conceptual model and specification for semantic recommendation

Lead contractor:	FZI
Editor(s):	Hans-Jörg Happel
Author(s):	Hans-Jörg Happel, Ljiljana Stojanovic, Walid Maalej, Maria Legal
Submission date:	March 2008
Dissemination level:	Public

Abstract: This document describes the conceptual model and specification of the semantic recommendation subsystem, which is part of the search and recommendation module developed in work package 4. Based on usage scenarios of the whole TEAM system and the specifications of semantic search component, an architecture and initial design for the component is derived.

Versioning and Contribution History

Version	Date	Modification reason	Modified by
0.1	16.01.2008	Initial structure	Hans-Jörg Happel
0.5	11.02.2008	Initial content	Hans-Jörg Happel
0.7	06.03.2008	Second iteration	Hans-Jörg Happel
0.8	10.03.2008	Additions and remarks	Ljiljana Stojanovic
0.9	12.03.2008	Ready for review	Hans-Jörg Happel
0.91	13.03.2008	Remarks and review comments	Walid Maalej, Maria Legal
1.0	14.03.2008	Review comments included	Hans-Jörg Happel

This document has been produced in the context of the TEAM Project. The TEAM project is part of the European Community's Sixth Framework Program for research and development and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

Table of Contents

EXECUTIVE SUMMARY	5
1 INTRODUCTION.....	6
1.1 PURPOSE OF THE DOCUMENT.....	6
1.1.1 DOCUMENT AUDIENCE.....	6
1.1.2 DOCUMENT STRUCTURE.....	7
1.2 AS-IS SITUATION.....	7
1.2.1 INFORMATION ACCESS.....	7
1.2.2 INFORMATION PROVISION.....	8
1.2.3 CONCLUSIONS AND RESEARCH CHALLENGES.....	8
2 REQUIREMENTS.....	9
2.1 PURPOSE OF THE RECOMMENDATION SUBSYSTEM.....	9
2.2 SCENARIOS	9
2.2.1 ERROR HANDLING (ASSISTANCE FOR KNOWLEDGE ACCESS AND PROVISION)....	9
2.2.2 COMPONENT REUSE (ASSISTANCE FOR KNOWLEDGE PROVISION)	10
2.3 FUNCTIONAL REQUIREMENTS	11
2.4 NON-FUNCTIONAL REQUIREMENTS	12
3 CONCEPTUAL MODEL	13
3.1 DIMENSIONS OF KNOWLEDGE ACCESS AND SHARING	13
3.1.1 THE DIMENSION OF TIME.....	13
3.1.2 THE DESCRIPTION OF INFORMATION NEEDS.....	14
3.1.3 THE DIMENSION OF PROVISION.....	16
3.2 ROLE OF ONTOLOGIES	17
3.3 CHALLENGES FOR ASSISTANCE SYSTEMS	18
3.3.1 IDENTIFY INTERESTING NEW INFORMATION (R1).....	19
3.3.2 IDENTIFY SUITABLE SITUATION FOR RECOMMENDATION (R2)	19
3.3.3 IDENTIFY STANDING INTEREST (R3)	19
3.3.4 DERIVE SUITABLE RECOMMENDATIONS (R4).....	20
3.3.5 DETERMINE IF USER NEEDS ASSISTANCE (R5)	21
3.3.6 DETERMINE ORGANIZATIONAL INFORMATION NEED (OIN) (R6).....	21
3.3.7 IDENTIFY VALUE OF CURRENT ACTIVITY (R7).....	21
3.3.8 CALCULATE EXPERTISE GAP (R8)	22
3.3.9 RELATION TO REQUIREMENTS.....	22
4 SYSTEM DESIGN.....	23
4.1 SUBSYSTEM DECOMPOSITION.....	24
4.2 RECOMMENDATION SUBSYSTEM	25
5 SUMMARY	28
REFERENCES.....	29

List of Figures

Figure 3: Time and description of information needs.....	14
Figure 4: Different modes of knowledge access.....	15
Figure 5: Time and Provision	16
Figure 6: Different modes of knowledge sharing	17
Figure 7: Elements of the abstract recommendation architecture.....	18
Figure 8: High-level architecture of the TEAM system	24
Figure 9: Subsystem decomposition	25
Figure 10: Design of the Recommendation subsystem.....	26

List of Tables

Table 1: Dependencies between use cases and requirements for the recommendation subsystem.....	12
--	----

EXECUTIVE SUMMARY

This document describes the conceptual model and specification of the semantic recommendation subsystem in the TEAM system. The main purpose of the component is to provide *proactive* knowledge assistance within the TEAM system. This can involve solution descriptions for similar problems, knowledge artefacts that were consulted by previous developers or experts that can be helpful to solve a problem.

Assistance can be twofold. Firstly, proactive recommendation can assist knowledge consumers with knowledge artefacts, which might be useful in their current situation. Secondly, proactive recommendation can assist knowledge providers to share or capture knowledge that could be useful for other team members.

The semantic recommendation subsystem works in tight collaboration with other TEAM components. While the Knowledge Desktop is responsible for providing non-intrusive visualization of recommendations, the TEAM context subsystem provides meaningful semantic interpretations of the user's current context. These interpretations can be used to provide more precise and relevant information recommendations. Furthermore, the semantic recommendation subsystem shares concepts and components with the semantic search subsystem as specified in deliverable D11. This also includes access to the TEAM metadata storage and P2P subsystems, which are responsible for storing and sharing knowledge artefacts within the TEAM system.

Based on this, the present document describes the evolution of the architecture and design of TEAM Work Package 4, involving interfaces for semantic recommendation functionality.

1 INTRODUCTION

In this document, the conceptual model and design of the semantic recommendation subsystem for the TEAM system are presented (task 4.6 in the description of work).

Therefore, we shortly describe the general problems of information access and information provision and how they are currently addressed in a Software Engineering context. Foundations for the study of this topic have already been described in deliverable D15 [20] ("State-of-the Art in Semantic Recommendation").

We then describe functional and non-functional requirements for the TEAM recommendation subsystem and develop a comprehensive model of knowledge access and provision within TEAM. Within this model, several challenges are identified and discussed. Finally, we describe the initial system design for the TEAM recommendation subsystem.

1.1 Purpose of the document

This document represents a combination of a conventional Requirement Analysis Document (RAD) and a System Design Document (SDD). On the one hand, the recommendation subsystem is described in terms of functional and non-functional requirements while on the other hand, the solution concepts of the recommendation subsystem are specified in detail.

It is important to stress that the TEAM system as well as the recommendation subsystem will be developed following an iterative and experimental process, cycling through all software engineering activities more than once. Improvements made during the iterations will not only affect the source code, but also all other models of the system, i.e., the requirements, the design models, the test cases and the documentation. These improvements, re-factoring and redesign will be summarized in the corresponding deliverables (D29, D30 and D37). For that reason, what is presented and discussed in this document is the first iteration of the conceptual model of the semantic recommendation subsystem.

Together with the semantic search subsystem, it forms the functionality developed in work package 4. Since both subsystems will be developed together and finally delivered as a search and recommendation module, future revisions might affect both subsystems.

1.1.1 Document audience

The audience of this document includes the following stakeholder groups:

- The project management team.
- The TEAM architecture task force (i.e., developers who participate in the design of the whole TEAM system).
- Developers who design and implement the recommendation subsystem.
- Developers who design other TEAM components that use services of the recommendation subsystem.
- Scientists, researchers and designers interested in generic architectural models for (semantic) recommendation systems.

1.1.2 Document structure

In the remainder of this section, we will shortly sketch the as-is situation with respect to information access and sharing in distributed team settings. The following descriptions of requirements and system design are loosely based on [2]. Section 2 describes the overall requirements of the recommendation subsystem. This includes a short overview of scenarios and functional as well as non-functional requirements. Section 3 contains the conceptual model for the implementation. Section 4 describes the initial design of the recommendation subsystem. This includes the overall architecture, its decomposition into subsystems and the detailed description of these subsystems. In section 5 we provide a conclusion and discuss future work.

1.2 As-Is situation

Knowledge sharing, as it is addressed in TEAM, can be defined as the "dual problem of searching for (looking for and identifying) and transferring (moving and incorporating) knowledge across organization subunits" [7].

It thus involves two different roles: the consumers of knowledge and providers of knowledge. We argue that proactive assistance or recommendations, as discussed in this deliverable, can be supportive for both roles. In fact, questionnaires answered by the TEAM industrial partners indicate that awareness about the existence of certain knowledge (information access) and the low level of experience sharing and capture (information provision) are two major blockers of knowledge sharing, especially in distributed scenarios.

The following two answers from the questionnaire summarize this nicely regarding information access: *"First, people are not well aware of existing knowledge and second, it takes them too much time and effort, to locate existing solutions in the existing systems."* ...and with reference to information provision: *"Developers are usually interested in resolving their problems rather than documenting their own ones, offering new knowledge or extending the knowledge-base. An innovative system of knowledge capture and maintenance that will indeed trigger developers in using it is missing."*

1.2.1 Information access

The creation of large-scale information repositories, such as the internet or corporate intranets has brought a large amount of information in the reach of users. However, due to constraints in time and mental capacity, it is hard for humans to find information suitable for solving a given task. Thus, recommendation systems, providing an intelligent "information push" functionality, which suggest information for a given task context of users are highly desirable [4].

As a knowledge intensive domain, the idea of supporting Software Engineering tasks with expert and recommendation systems is quite obvious. Given the existence of large amount of reusable source code in both corporate repositories and the internet, there is a large potential to improve the efficiency by assisting developers. Several research efforts presented in deliverable D15 [20] indicate a high interest for recommendation tools. However, our analysis in this deliverable has shown that current systems have a number of limitations such as their architecture, flexibility and the limited usage of implicit context information.

The user context information maintained by the described systems is rather simplistic and recommendations are either triggered automatically in a continuous way, or have to be requested by users. For the TEAM system, we strive to create a richer user context which allows identifying certain problem situations automatically and thus allows for more focused recommendations.

1.2.2 Information provision

Since no single person can be specialist for every field, modern organizations are forced to adopt a division of knowledge working style, where specialized employees are responsible for particular fields of expertise. Accordingly, in complex development projects, many people have to collaborate in ad hoc or distant project teams, introducing the need to exchange information and knowledge [18]. Thus, the effective and efficient sharing of knowledge among collaborating workers is considered a major success factor for modern organizations [19], [5], [7].

Personalization and codification are typically described to be the two core strategies for knowledge sharing [6]. The personalization strategy primarily relies on personal communication to share tacit knowledge, which only exists in the “heads” of individuals (see e.g. [6]). In turn, the codification strategy targets explicit knowledge which is for example captured in documents. Here information technology is an important enabler, especially in cases where regular personal communication is not feasible, such as in large organizations or distributed settings.

Traditional IT-based knowledge management systems (KMS, see e.g. [1], [15]) such as central information repositories or groupware systems with shared folders have limited success since they suffer from a number of motivational, organizational and technical barriers [19][17]. Key issues are the low motivation to contribute knowledge to public repositories [3][23] and the effort for creating and maintaining central knowledge repositories [8][9].

Thus, there is a fragmentation of codified knowledge in the organization. Fragmentation means, that codified knowledge is scattered across private and public information spaces in an organization. By private information spaces, we mean all (electronic) information, which is only accessible by a single person in an organization (e.g. local files or personal E-Mails).

Therefore, distributed teams require tools which help them to provide existing knowledge or to capture existing knowledge, which can contribute to raise the efficiency of the overall organization.

1.2.3 Conclusions and research challenges

In deliverable D15 [20], we analyzed the state-of-the-art regarding semantic recommendation. Basic findings regarding applications in a Software Engineering context are:

- Existing systems are limited to either recommend methods to use next or artefacts which are “related” to the current situation of the user.
- The description of the user’s situation or “context” is limited to single properties such as current class a user is working in.

- There is no pro-active triggering of information push: recommendations are either triggered automatically in a continuous way, or have to be requested by users.
- Architectures of the surveyed systems are inflexible and do not allow for extensions.
- The existing systems are based upon a centralized, static corpus. The aspect of information provision is not addressed at all.

In the remainder of this document, the initial conceptual model and design of the Semantic Recommendation subsystem for the TEAM system are presented. Based on the ontology-based infrastructure and the complementary components of the TEAM, we identify the following contributions of our model:

- Recommendations can be based on deep semantic analysis and understanding of the user's working context.
- Recommendations can be autonomously triggered by the TEAM system (instead of continuous or manual triggering of existing systems).
- Recommendations can be based on changes in the outside information space.
- Recommendations can be both synchronous and asynchronous.
- Recommendations not only comprise information push, but also the recommendation to capture and share information.

2 REQUIREMENTS

In this chapter, the functional and non-functional requirements of the recommendation subsystem are presented. The overall TEAM system requirements are addressed by a separate deliverable (D9 [21] resp. D24 [22]), in which TEAM system use-cases are described in detail.

2.1 Purpose of the recommendation subsystem

The recommendation subsystem is responsible for a) proactively providing users with knowledge, which can be of help in their current situation and b) proactively assisting users in capturing and sharing knowledge which is beneficial for other members of their team.

2.2 Scenarios

In this section, typical scenarios for using the recommendation subsystem are described. They comprise a subset of the scenarios described in deliverable D9 and should serve as a basis for the general understanding of the purpose of recommendation functionality within the TEAM system. System actions are written in italic face.

2.2.1 *Error handling (assistance for knowledge access and provision)*

Actors: Developer Dave and Tester Tim

Flow of Events

- Tim should test the new internal release of an extended OpenXChange groupware for testing. During that, he identifies that the application does not work as expected and reports it using the bug tracking system.
- Dave receives a note from the bug tracking system indicating the new issue with the module he maintains along with the description how to reproduce it.
- Dave reproduces the issue. The log does not suggest the location of the error.
- *During reproduction, the TEAM system recognizes that Dave switched to a maintenance task and provides assistance regarding his new working context, by offering relevant knowledge artefacts.*
- Dave starts the debugger. He sets breakpoints and watches variables to locate the problem in the source code.
- During this step, Dave annotates the breakpoints and watches the reasons of setting them.
- *The TEAM system captures this information and furthermore recommends on where to set breakpoints and which variables to watch based on what previous developers watched while handling similar errors.*
- Dave finds the problem, and fixes it. He annotates the fix with the most useful breakpoints and watches along with the indication of why these were the most useful.
- *The TEAM system stores this information.*

2.2.2 Component reuse (assistance for knowledge provision)

Actors: Developer Dave and Peer-Developer Peter

Flow of Events

- Dave is developing a groupware application. He downloads a recent version of the OpenXChange software and adds it to the Java Build Path setting of the project.
- Then he imports the package in a source file of his groupware application he is currently editing.
- *The TEAM system recognizes that this is code maintained by a third party. To support Dave, the TEAM system tries to provide him with a list of other projects which include OpenXChange. There are none available in the network.*
- *However, the TEAM system recognizes that Peter, a coworker in the same organization, has some code and documents about OpenXChange in his private workspace.*
- *The TEAM system notifies Peter, that colleagues seek information that he owns exclusively and asks if he wants to share it.*
- Peter decides to share part of his information which TEAM then suggests to Dave.
- *The TEAM system displays the new resources to Dave.*
- Dave takes a look at the information provided by Peter and recognizes some interesting documentation on the OpenXChange API and some example code from Peter's project.

Note that these scenarios are just examples to illustrate possible usages of (parts of) the recommendation functionality. The complete functionalities and interfaces of the recommendation subsystem are presented in the following.

2.3 Functional requirements

Functional requirements from the users' perspective have been defined for the whole TEAM system in deliverable D9 as use cases ([21] resp. D24 for an updated version [22]). Six of these use cases are of particular importance for the recommendation subsystem:

- Recommend knowledge
- Request knowledge
- Recommend expert
- Request expert
- Ask developer to capture solution
- Ask developer to share knowledge
- Provide suggestions for knowledge to share
- Subscribe to query

In the past weeks, we have been collecting questionnaires from our industrial partners to refine the TEAM scenarios for reuse and error handling. The answers indicate that the mentioned requirements are highly relevant, with particular emphasis on recommending knowledge and motivating developers to capture and share knowledge (c.f. also section 1.2).

Based on end user demands and UML communication diagrams that describe the realization of the TEAM use cases (c.f. D9 [21] and D24 [22]), the following requirements for the recommendation subsystem have been derived:

R1: Identify relevant knowledge

- The system should be able to identify which new knowledge is relevant to the user.

R2: Identify suitable situation for context-aware recommendation

- The system should be able to identify, if the current situation is suitable to provide the user with knowledge.

R3: Identify standing interest

- The system should be able to automatically identify long-term (“standing”) information needs of the user.

R4: Derive suitable recommendations

- The system should be able to automatically formulate queries and retrieve appropriate recommendations in a given situation.

R5: Determine if user needs assistance

- The system should be able to judge, if the user needs proactive assistance in a given situation.

R6: Determine organizational information need (OIN)

- The system should be able to determine which particular information that is sought in the overall organization that the user could contribute.

R7: Identify value of current activity

- The system should be able to determine, how valuable the current activity of the user is for solving problems other members of the organization had before.

R8: Calculate expertise gap

- The system should be able to identify if the user can provide meaningful expertise for a given organizational information need.

R9: Represent and persist information need

- The system should be able to track and store a user's information needs, since this is helpful to judge if assistance at later time is suitable.

Table 1 describes the connection between the use cases and these requirements.

	R1	R2	R3	R4	R5	R6	R7	R8	R9
Recommend knowledge	x	x	x	x	x				x
Request knowledge				x					
Recommend expert		x	x	x	x			x	x
Request expert				x				x	
Ask developer to capture solution		x				x	x		
Ask developer to share knowledge						x		x	
Provide suggestions for knowledge to share						x			
Subscribe to query									x

Table 1: Dependencies between use cases and requirements for the recommendation subsystem

2.4 Non-functional requirements

An overview of non-functional requirements for the whole TEAM system is given in deliverable D9. Several statements of the industrial partners refer directly to the recommendation subsystem. Three non-functional requirements are of particular importance:

- **Reliability** is rated as very important by the industrial partners in terms of the relevance of result (“4/5 recommendations should return meaningful results”).
- **Performance** is also related to the recommendation subsystem. While it is not rated as the most crucial issue, end users expect the system to provide results in about one to five seconds. For complex queries, utilizing the P2P network, longer response times are also considered to be acceptable. However, this aspect mainly depends on the performance of the metadata store and the P2P component. Also, providing results should not consume considerable processing power of the local machine.

- **Usability** is mentioned to be important – especially in terms of ease of use and time to learn the system. However, this is not directly relevant for the recommendation subsystem, since its user interactions will be carried out by the Knowledge Desktop.

3 CONCEPTUAL MODEL

This section describes the conceptual model of the recommendation functionality in the TEAM system. Therefore, we will first clarify our notion of knowledge access and sharing by defining differences between “search” and “recommendation” functionality. We will then map these different notions to the functional requirements, discussed in the previous section. Afterwards, the conceptual model for realizing these requirements will be described.

3.1 Dimensions of knowledge access and sharing

As described in section 1.2, we have shown that knowledge providers and knowledge consumers are the crucial roles for efficient knowledge sharing. In this section, we will elaborate on the respective activities of knowledge access and sharing. In particular, we derive three characteristics, which help to distinguish certain types of “search” and “recommendation” systems:

- The dimension of **time**, which stands for the dynamics of document collections¹ and influence the direction of interaction – i.e. if the information need is triggered by the user (knowledge pull) or by the system (knowledge push).
- The **description of information needs**, which describes if the user provides an explicit representation of her information need (e.g. a query).
- The dimension of **provision**, which explains the appearance of new information.

We will now describe these characteristics and the resulting knowledge access modes.

3.1.1 *The dimension of time*

Most enterprise and web search systems implement a standard architecture which is based on user queries and documents. Documents are periodically crawled from repositories (such as folders in a file system or the web) and analyzed in order to build an index. Once a user query comes in, this query is matched against the index in order to derive suitable results.

Obviously, this architecture has difficulties to deal with the dynamics of document collections. Since new documents are only added during periodical crawls, users can only retrieve those documents, which were already indexed. Accordingly, this paradigm is also called retroactive search ([12], see the top left side of Figure 3). From the user’s point of view, this is problematic in two cases. If the information need is not satisfied by the existing results, the user must return at later times in order to check if there is something new.

¹ Since we will introduce our notions based on discussing existing (web) search systems, we will partially refer to “documents” in this section, instead of using the more general notion of “knowledge artefacts” in TEAM.

Secondly, if the user has a constant information need, e.g. if she wants to monitor a topic for a certain period, she also needs to do a manual query from time to time. This is especially relevant for Software Engineering ecosystems, where constant changes of related artefacts make it difficult to keep up to date.

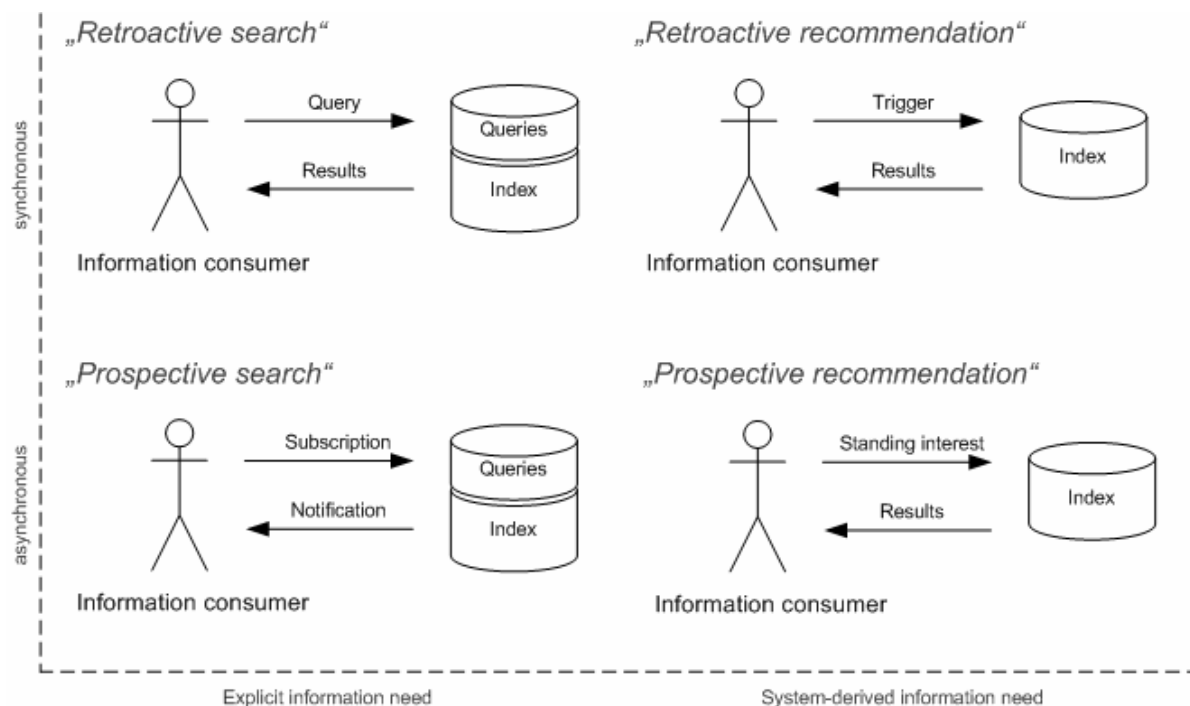


Figure 1: Time and description of information needs

These problems have been addressed by a new paradigm called prospective search ([12] [24], see the bottom left side of Figure 3). Instead of querying the status quo of information, prospective search allows the user to register for a certain topic and to receive notifications if new information appears. Alternate terms for this system model are “publish/subscribe” or “continuous querying”. Popular implementations of such functionality are Google Alerts [11] and Windows Live Alerts [16]. These systems collect the queries from the users and match them periodically against new documents, which have been discovered during crawling. If there is a match, users are notified about these new results, either by E-Mail or by embedding the results in a personalized user interface.

Accordingly, while in retroactive search applications, knowledge access is typically triggered by the user – e.g. by pressing some kind of “search” button, prospective approaches are usually not directly triggered by the users. In turn, they are triggered by the external situation (which we will later call outside information space), in which new information appears.

3.1.2 The description of information needs

While most search and recommendation systems do actually not differ in the direction of interaction (since most recommendations are triggered explicitly), the main difference lies in the description of the information need.

In search systems, the information need is explicitly stated by the user in the form of a query. This can be only optionally complemented by additional, automatically derived metadata. Recommendation approaches on the other hand, do not require users to explicitly state a

query. Instead, they automatically derive the information need, based on contextual information, such as items browsed by the user, resp. a user profile.

As depicted in Figure 3, the distinction between retroactive and prospective approaches can also be made for recommendation. Retroactive recommendations provide a user with immediate results while prospective recommendation systems try to derive a notion of “standing interest” [24] and can provide new information continuously.

Accordingly, the approaches to knowledge access discussed so far can be classified as depicted in Figure 4. It lists the different approaches, and characterizes them by the dimensions of “trigger” and “information need description”.

“Trigger” refers to the entity or event that causes the system to deliver results. Rounded shapes denote events, which are primarily triggered by the system user while rectangles denote system or environmental events.

Knowledge access approaches	Trigger	Information need description
Retroactive Search	User click	Query formulation
Prospective Search	Subscription/ New results	Query formulation (Subscription)
Retroactive Recommendation	User click	Query construction
Prospective Recommendation	New results	Standing interest
Proactive Recommendation	Need identification	Query construction

Figure 2: Different modes of knowledge access

Similarly, the “information need description” can either be specified directly by the user (e.g. in form of a query or subscription) or derived by the system.

Compared to the previous section, Figure 4 contains one more recommendation approach named “proactive recommendation”. As our state-of-the-art analysis revealed (c.f. deliverable D15 [20]), most recommendation approaches differ from pure search applications just in the automatic derivation of the information need and do not take into account the current situation of the user (i.e. if this situation affords a recommendation).

Since the monitoring and derivation of user activities is a core feature of the TEAM context system, and since the notion of working context (what is the current development task, which artefacts are edited and which tools are used) is considered very relevant in software development [13], we will discuss proactive recommendations later in section 3.2.5.

3.1.3 The dimension of provision

While prospective systems acknowledge the dynamics of document collections and support users in satisfying information needs with documents that appear after query time, they do not address *how and why* new documents actually appear.

Prospective as well as retroactive approaches assume that new documents are arbitrarily added to the document collection over time and become part of the index during one of the periodical crawls. However, much information which is contributed to public repositories is initially part of a hidden, private information space. Thus, we think that search and recommendation systems should not just focus on knowledge access, but also be complemented by functionality that fosters knowledge sharing (c.f. our discussion in section 3.2 of deliverable D15 [20]).

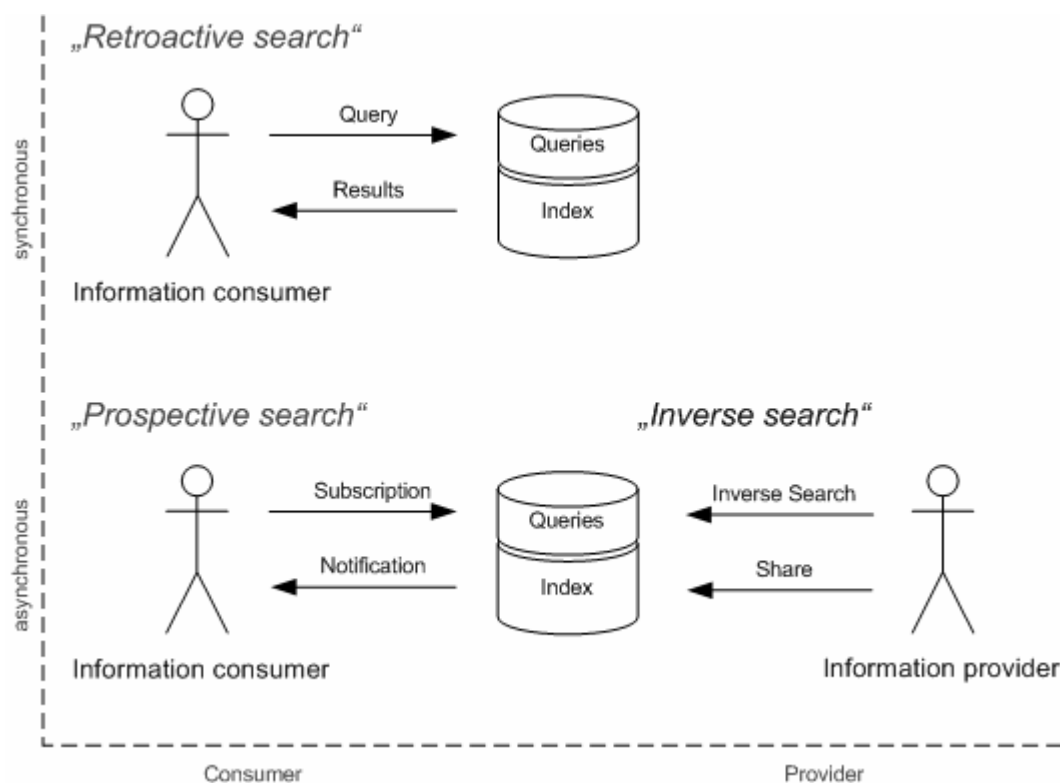


Figure 3: Time and Provision

This step requires the introduction of information providers to the overall knowledge access process. Based on the analysis of already existing query logs, information providers can find out if their documents should be included to the public space. Thereby, the process of information provision to the public collection would no longer have to be black-box, but could be informed and stimulated by existing information needs. Indeed this is in particular important for software development where informal knowledge sharing is crucial for project success. While developers in collocated teams are aware of the problems and needs of their colleagues, distributed teams have fewer clues about the situation of their peers. Thus, tools and approaches that support distributed teams in knowledge sharing can be considered to be beneficial.

As depicted in Figure 5, we call our approach for triggering the sharing of private knowledge artefacts *inverse search*. However, inverse search only helps to include information, which

already exists in explicit form. Given the additional user information within the TEAM system, the information need could also be applied to help capturing implicit information.

Knowledge sharing approaches	Trigger	Provided information
Inverse search	On demand	Existing document/artefact
Passive recommendation to capture	On demand	To be captured by the user
Active recommendation to capture	Situation dependant	To be captured by the user

Figure 4: Different modes of knowledge sharing

We call this “recommendation to capture” (c.f. Figure 6). As opposed to inverse search, it does not help to provide existing explicit information (e.g. document or other artefacts), but implicit information captured by the user upon the systems recommendation.

Two triggers for such recommendations can be envisioned. Passive recommendation to capture is based on the idea, that the user decides when to get a general list of desired information, which she could contribute. Active recommendation in turn, tries to identify, if the current activity of the user could be helpful for other members of the TEAM network. If so, active recommendation asks the user explicitly to capture information. The underlying rationale is, that users might be more willing to capture information, if they can do it in harmony with their current task, without distraction.

As an example, the TEAM system could ask a developer who just resolved a particular error by debugging, to capture her experience, when it can derive, that other developers were dealing with the same problem without success.

3.2 Role of ontologies

So far, the different approaches towards search and recommendation have been primarily compared based on the notions of documents and a (keyword) index. While the “search”-related approaches can *benefit* from considering metadata/triples and semantic queries instead (see e.g. deliverable D15 [20]), the described recommendation scenarios can hardly been realized without semantic representations.

Considering for example the notion of prospective search, a semantically enabled prospective search can leverage background information from the underlying ontologies to either refine (in the case of too many results) or relax (in the case of few results) the returned information.

Furthermore, the approach of proactive recommendation is hardly imaginable without semantic information. A rich and structured representation of the user’s current situation is required to semantically match if a particular piece of new knowledge is relevant or not. A pure syntactical keyword match is insufficient in this case.

Without semantics, the matching will be based on keywords, while in Software Engineering these can be different in different tools, programming languages, problem domains etc. Take

as an example the annotation of knowledge artefacts. A syntactical similarity could match only those annotations, which appear almost identically in the interest profile of a user. A developer interested in “OSGi technologies” would thus not be notified, if a new release of “Spring DM” appears. Given a software engineering ontology, as described in deliverable D9 (section 2.3.3.2 of the revised version) which assigns “Spring DM” as a subconcept of “OSGi technologies”, an ontology-based similarity measure could infer, that the topic might be relevant for the developer.

Additionally, syntactically different expressions of the same information need can not be compared without a semantic abstraction. A particular error message for example, can be expressed based on the runtime environment output, based on the developers own conceptualization, based on data from the build management tool or from the operating system log. Also, without semantics it is not possible to abstract from the problem domain: a developer using OSGi for an embedded system in a medical device and another developer using OSGi for luggage tracking at an airport might have problems to share experience, although their technical problems might be similar, independently of the particular domain vocabulary.

3.3 Challenges for assistance systems

Summarizing section 3.1, in a situation of knowledge consumers, who efficient access to information and knowledge providers, who must capture and share information, assisting tools can play an important role as a mediator between both roles.

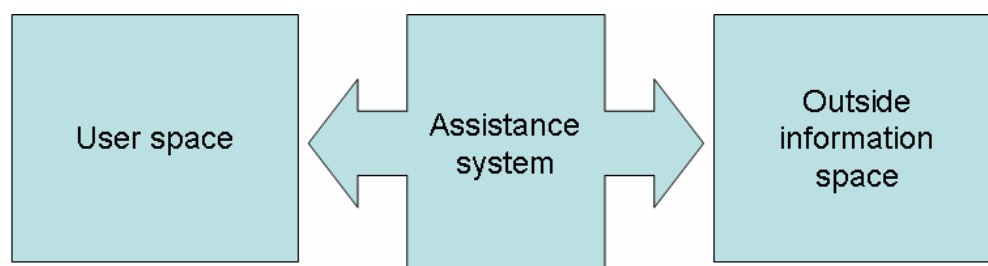


Figure 5: Elements of the abstract recommendation architecture

Translating that to the TEAM environment, the corresponding conceptual architecture involves three fundamental elements (see also Figure 7):

- The user under consideration who can access or share information.
- The outside information space, from which the user consumes information resp. shared her information with.
- The assistance or recommendation system, which mediates between both spaces – i.e. which notifies the user about helpful information or about information she should share.

The mediating role of the assistance system has thus to assure an efficient exchange of knowledge between both spheres. However, a practical and user friendly realization of such a mediation task involves a number of challenges, which we discuss in the following sections.

3.3.1 *Identify interesting new information (R1)*

In an asynchronous recommendation scenario, it is necessary to predict how interesting new information which appears in the outside information space is for the user. While it is not feasible to assess the relevance of all outside information, we focus on the relevance of outside information with respect to concrete information needs of the user under consideration. These information needs are either explicitly stated by the user or inferred by the system (c.f. R3).

Typical measures are (c.f. [24][14]):

- Updates of information the user has already accessed.
- The overall amount of information regarding the information need.
- The difference of the new information from existing information.
- The expertise of the user and annotations of the knowledge artefacts.

Thus, the ontology-based similarity between the user information need and the respective knowledge artefact defines if an artefact should be considered for recommendation.

Compared to purely syntactical measures of similarity, which operate on raw attributes such as strings or real numbers, ontology-based similarity measures take into account the semantic relations between concepts in an ontology (c.f. section 3.2).

3.3.2 *Identify suitable situation for recommendation (R2)*

The identification of a suitable situation to provide proactive assistance to a user is important for asynchronous scenarios. It deals with the best situation to a) inform a user about interesting new information (R1) and b) to ask the user to capture experience (R7).

We assume that a user is a) more receptive for new information and b) more willing to capture knowledge if it fits her current task and she does not have to switch her current task. An example could be to push information about an issue related to a class only, if this class is currently within the working context of the user.

A recommendation should thus be based on the ontology-based similarity between the current working context of the user on the one hand, and the context of the new/required information on the other hand.

3.3.3 *Identify standing interest (R3)*

Standing interest denotes a kind of implicit subscription which is derived from the behaviour of the user. As an example, we can assume that a user has a high standing interest in information w.r.t a certain source code class, if she does continuous modifications on it.

Works in the context of web search engines propose several heuristics for deriving standing interest. Based on these derivations, they abstract a generalized representation of “standing interest”, which is later used for prospective search purposes (c.f. section 3.1). We now define the derivation and generalization process.

In order to derive standing interest, we have to maintain information about the user’s activities and search behaviour. We derive two kinds of standing interest

- Standing interest about knowledge artefacts

- Standing interest about an information need

To derive artefact-based standing interest, we adopt the context-model of the mylyn approach [13]. It consists of degree-of-interest information, which is based on direct interactions with a class, and on a degree of separation information, which is based on the call-distance among classes. However, since we are using the information for a different purpose, we redefine the notion of decay.

A standing interest about an information need can be defined as an information need, for which the user would be interest in results, also if they appear in the future. As an example, a developer with an information need about when a certain library or a tool version is released would probably not be interested in similar information in the future. On the other hand, a developer with an information need about a certain research topic (e.g. distributed applications or ontology-based systems) might be interested in future results also.

In order to derive such kinds of standing interest, existing approaches analyze the query history of users and try to identify recurring patterns. Therefore, we based upon the model described in [24], which identifies the following aspects:

- Prior fulfilment (if the user has already found satisfactory results)
- Query interest level (based on the number of related queries)
- Need/interest duration (based on the time-dependency of the information need)

3.3.4 *Derive suitable recommendations (R4)*

An identification of suitable recommendations is necessary if there is a *signal* of an actual information need – either by derivation (R3) or by explicit demand of the user – but *without a concrete specification* of the information need (e.g. a query).

The challenge is therefore to formulate an information need based on the current situation of the user. Since this formulation will never be precise, the goal of this step is to generate a number of candidate needs, which can be approved by the user.

Therefore, we generate a list of candidate instances of the current user situation and rank them by their probability. Sources for this list are:

- Current or recently addressed problems of the user.
- Elements of the current context (e.g. method, class, file, package, project).
- Problems related to elements of the current context.
- History and experience of the user (e.g. exclusion of past recommendations or elements of the current context, with which the user is well acquainted).
- Recent search activities.

All matching instances will be retrieved and sorted by their mutual distance in the ontology (e.g. problems relating to the current method will rank higher than problems relating to the project). The refinement dialog should be further accompanied with the possibility to create a new problem instance, in case none of the recommendations fit.

3.3.5 *Determine if user needs assistance (R5)*

The determination if a user requires proactive assistance is driven by a) the current situation of the user, b) the outside information space (i.e. information that is shared by others within the TEAM network; see also section **Error! Reference source not found.**) and c) the gap between a) and b).

Among these, the current situation of the user is the most dominant factor. It is largely driven by the context system. Exemplary situations are part of the TEAM Ontologies, as defined in deliverable D9 (see section 2.3.2 of the refined version).

Such situations do not only relate to obvious problems, such as errors or exceptions, but also to typical situations in which software developers require information assistance. Some examples are:

- A developer creates a new class.
- A developer modifies an existing class.
- A developer spends much time without constructive activities.
- A developer encounters an exception while running the program.
- Test cases fail.
- A developer debugs a program.

On the other side, also changes in the outside information space can trigger proactive assistance:

- A new bug is filed w.r.t. to the current working context.
- A new document is created w.r.t. the current working context.
- A document w.r.t the current working context is updated.
- A remote developer is working on the same file.
- A remote developer is working on an adjacent file.
- A new version of a used library appears.

Triggering a recommendation thus depends on matching the user's context with the outside information space.

3.3.6 *Determine organizational information need (OIN) (R6)*

Since the notion of recommending knowledge for sharing is tightly related to the demand of external users (i.e. queries), there is no clear distinction between “search” and “recommendation” aspects. Accordingly, the notion of OIN has already been discussed in deliverable D11 [10].

3.3.7 *Identify value of current activity (R7)*

For deciding, if a user should be asked to capture her experience immediately it is required to a) identify if the user is currently solving a problem or solved it already and b) if this solution is important for the organization.

Therefore, if the context system signals activity that indicates work on a certain issue, the recommendation subsystem finds out if there is a situation of information shortage – i.e. there is neither information in the local metadata store, nor in the TEAM network (R6). If this information shortage is accompanied by organizational information need (c.f. deliverable D11 [10]), the developer is asked to capture information on this topic.

Besides based on the current activity, such a capturing request can similarly be triggered in the search interface of the Knowledge Desktop. The rationale for this is, that people searching for some information do in fact have basic knowledge about it (otherwise they could hardly formalize a query).

3.3.8 *Calculate expertise gap (R8)*

Calculating an “expertise gap” is required for providing asynchronous, passive recommendation to capture knowledge. The idea here is to provide knowledgeable users with the possibility to externalize some knowledge, which is helpful to their colleagues.

Therefore, the recommendation subsystem must be able to calculate a so called “expertise gap” between knowledge required within the organization and the expertise of the user. Note that this is similar to the situation of sharing knowledge (R6), but differs by the absence of existing explicit knowledge that could be shared immediately.

Accordingly, the calculation of the expertise gap is based on a) external demand for information and b) possibility to contribute.

This will be realized in a three step process:

- The regular retrieval of problems without solutions and unsatisfied search queries from the outside information space.
- The selection of a subset of candidate problems based on a) explicitly stated expertise of the user, b) severity of the organizational information need and c) previous capturing recommendations to retrieve additional context information for them.
- The matching of these candidate problems with a) the explicitly stated expertise of the user and b) similarity of the user context with the context of candidate problems.

3.3.9 *Relation to requirements*

In this section, we assign the requirements, as initially stated in section 2.3 and further refined in the previous sections to the different modes of knowledge access and knowledge sharing (section 3.1). As we can see in italic style, there is a large overlap of requirements, resp. each mode defines its unique requirement.

Prospective search

- Which new information is relevant for the user information need (R1)
- What is the best situation to show new information (R2)

Prospective recommendation

- What is standing interest of the user (R3)
- *Which new information is relevant for the user information need (R1)*
- *What is the best situation to show new information (R2)*

Retroactive recommendation

- What is the current interest of the user (R4)

Proactive recommendation

- Does the user need assistance (R5)
- *What is the current interest of the user (R4)*

Recommend to share (“Inverse search”)

- Are there documents with a high organizational information need (R6)

Active recommendation to capture

- Can the current activity be helpful for others (R7)

Passive recommendation to capture

- Is the user expert in a desired topic/for a problem without solution (R8)

4 SYSTEM DESIGN

This section describes the design of the recommendation functionality inside the TEAM system. In the following, we will refer to this as the “Search and Recommendation” module², since recommendation is part of the same work package and shares functionality with the search sub-system. Thus, this chapter is widely complemented by the system design section of the semantic search system (see deliverable D11 [10]). While the “big picture” will be reproduced in the following, the detailed design of those components, which were already introduced in D11 will be omitted.

² In the Description of Work, this is sometimes also called „Knowledge Access“ subsystem

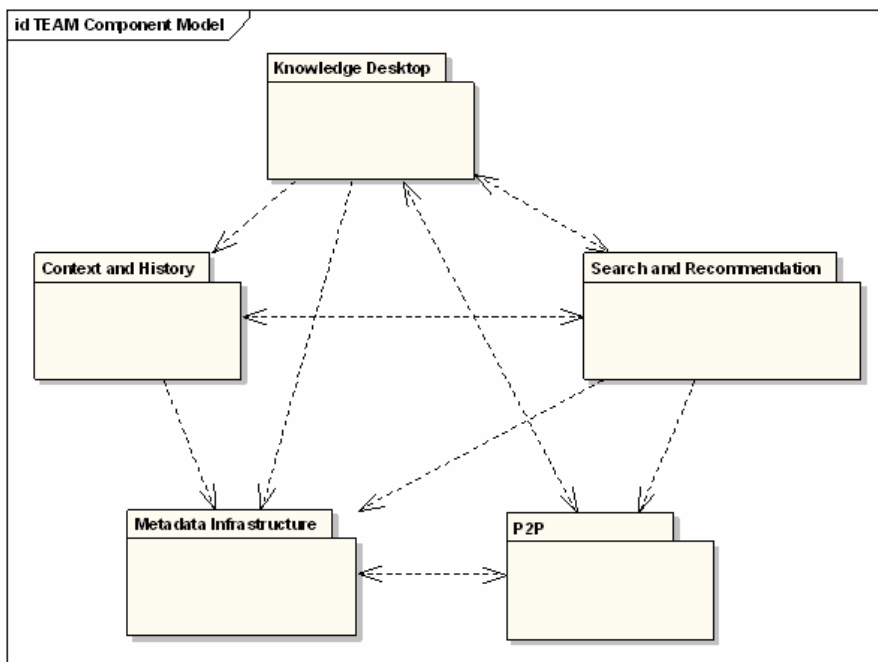


Figure 6: High-level architecture of the TEAM system

The high-level architecture of the whole TEAM system is depicted in Figure 8. Obviously, the Search and Recommendation module has dependencies on all other modules. Based on the communication dependencies described in D9 [21], these dependencies can be qualified as follows:

- The **Knowledge Desktop** is used to trigger search actions and display search results and recommendations.
- Information from the **Context and History** module is used to match suitable search results and recommendations.
- The **P2P** module offers access to the information shared by other developers in the Peer-to-Peer network.
- Finally, the **Metadata Infrastructure** provides access to all structured information stored in and by the TEAM system.

The following section describes the decomposition of the search and recommendation module into several subsystems. Each subsystem is explained in more detail afterwards.

4.1 Subsystem decomposition

Our initial decomposition of the Search and Recommendation module is depicted in Figure 9. The choice is based on grouping related functionality in a modular fashion, such that some subsystems might also be reused in a different context or by other components.

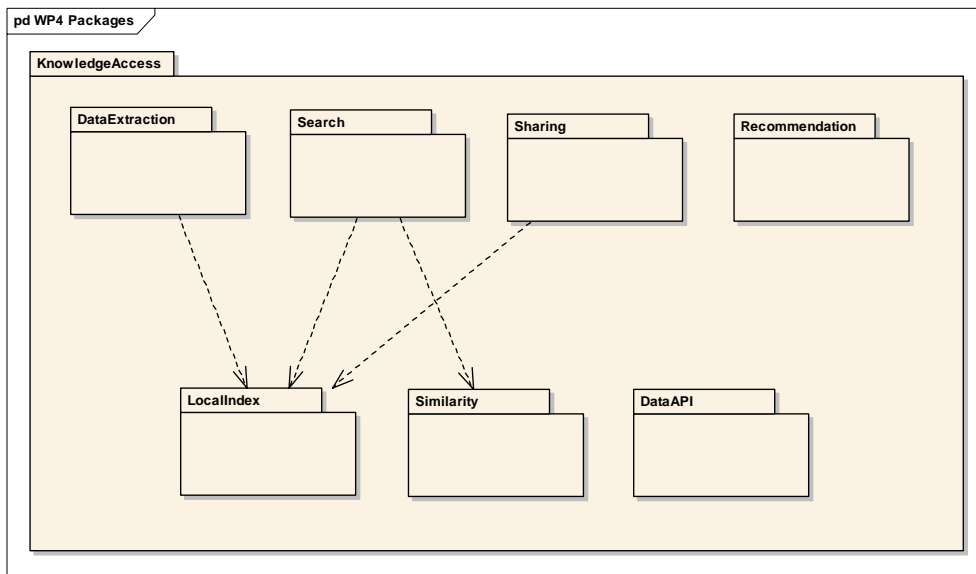


Figure 7: Subsystem decomposition

In total, we identified seven subsystems, as depicted in Figure 9:

- The **DataExtraction** subsystem is responsible for extracting metadata and index terms from resources.
- The **LocalIndex** subsystem wraps a full-text index storing and retrieval interface on the local machine.
- The **Search** subsystem offers the core search functionality and controls the merging and ranking of results from the local metadata store and the Peer-to-Peer network.
- The **Sharing** subsystem contains functionality for information sharing (c.f. section 3.3).
- The **Similarity** subsystem provides similarity algorithms for comparing strings and ontology-based metadata such as user context or knowledge artefacts. It is currently used by the search subsystem and might be extended later for extended usage by the recommendation subsystem.
- The **DataAPI** subsystem contains a object representation API for ontologies and metadata, which is intended to be used in communication with the metadata store.
- The **Recommendation** subsystem serves as placeholder for functionality which is defined and designed later in the project.

Since most of these subsystem have already been described in deliverable D11 [10], we will only describe the recommendation subsystem in more detail.

4.2 Recommendation subsystem

The recommendation subsystem provides functionality to analyze the current situation of the user, to monitor the outside information space, and to derive proactive recommendations based on the comparison of both (c.f. Figure 7). Furthermore, it maintains a history of recommendations and about explicit and implicit long term information needs (subscriptions/standing interest).

Thus, the subsystem provides the following services:

- Request knowledge artefacts
- Request expert
- Store and remove information need subscriptions
- Remove standing interest descriptions

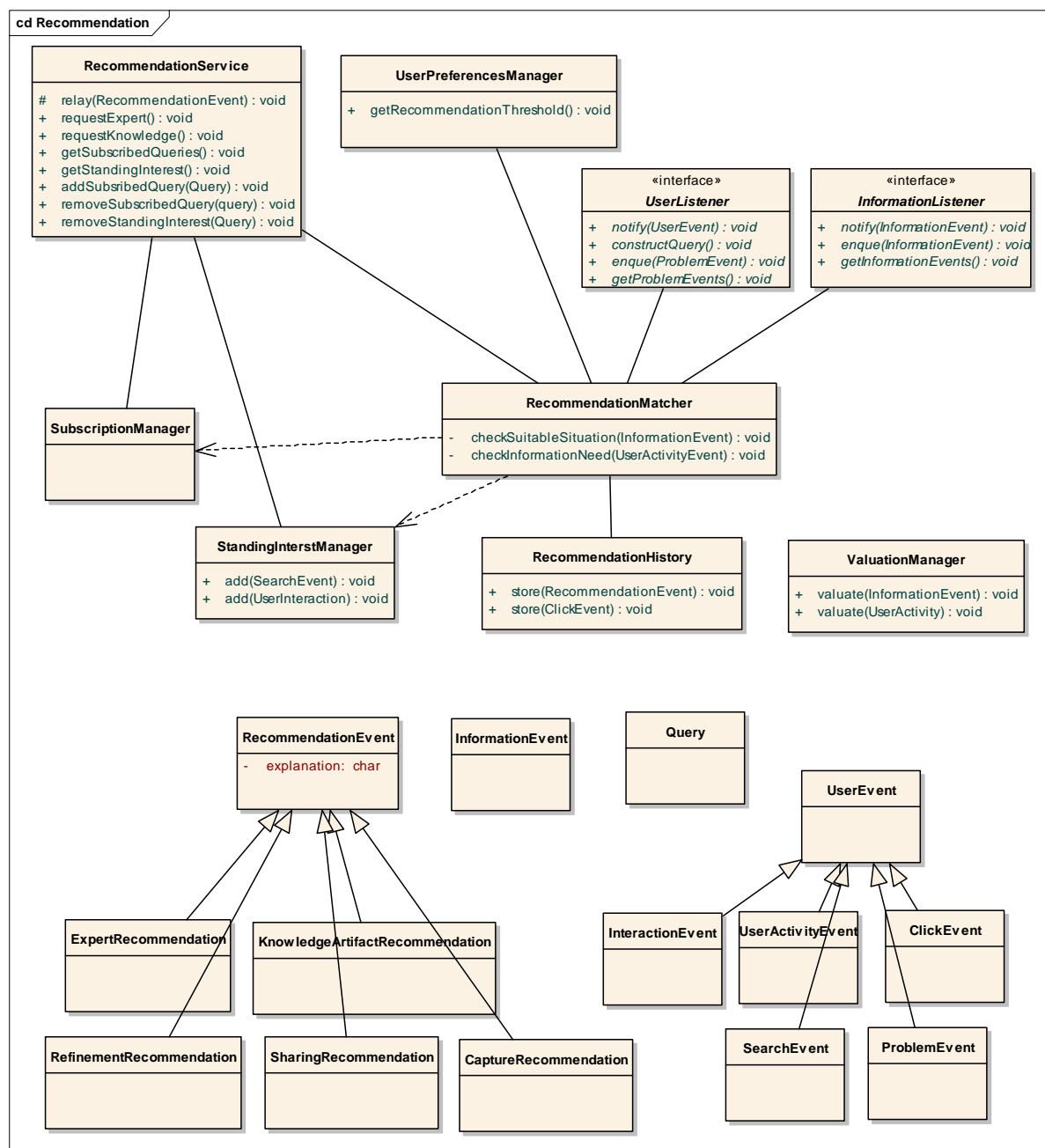


Figure 8: Design of the Recommendation subsystem

The subsystem consists of the following classes (see Figure 10):

- **RecommendationService:** This is the basic service façade, which can be triggered for recommendations and manages subscriptions and standing interest. The actual realization of these services is delegated to other classes.
- **SubscriptionManager:** This class takes care of user information need subscriptions.
- **StandingInterestManager:** This class is responsible for deriving standing interest from user interactions and search events.
- **UserListener:** This interface defines methods for the interaction with the activities of the user.
- **InformationListener:** This interface provides methods to manage data about the outside information space.
- **RecommendationMatcher:** This is the main class of the recommendation subsystem. It creates new recommendations by processing data stemming from the UserListener and InformationListener. It generates RecommendationEvents, which are stored in the RecommendationHistory and sent out to the TEAM system using the relay() method of the RecommendationService.
- **RecommendationHistory:** This class takes care of past recommendations which have been sent to the user.
- **ValuationManager:** This class is responsible for calculating the “value” of a current activity of the user (to decide if the user should capture her experience) and about the value of new information in the outside information space (to help deciding, if the user should be notified).
- **RecommendationEvent:** This class (and its subclasses) represent recommendations which are generated by the RecommendationMatcher.
- **UserEvent:** This class (and its subclasses) represent different activities of the user within the IDE. They are observed and created by the TEAM context system. The recommendation system receives them by the way of the UserListener interface.
- **UserPreferencesManager:** This is a help class to access preferences for the level of recommendations, which the user has been explicitly specified in the TEAM configuration dialogs.

To gather information, the system relies on events generated by the context system. Data is retrieved from the metadata store and the P2P module. The Knowledge Desktop is the primarily responsible listener for RecommendationEvents, which are emitted by the recommendation subsystem.

5 SUMMARY

In this document, we described the conceptual model for the recommendation subsystem within the TEAM system. The described model goes far beyond current state-of-the art systems in several aspects:

- Recommendations are based on deep semantic analysis and understanding of the user's working context.
- Recommendations can be autonomously triggered by the TEAM system (instead of continuous or manual triggering of existing systems).
- Recommendations can be based on changes in the outside information space.
- Recommendations can be both synchronous and asynchronous.
- Recommendations not only comprise information push, but also the recommendation to capture and share information.

As it has been highlighted throughout the deliverable, many of these aspects are dependent on the availability of semantically rich representation of the user context and the knowledge artefacts under consideration. This is particularly important for the case of proactive assistance. Our model is furthermore extensible and not limited to certain types of knowledge artefacts.

Currently, we are working on the implementation of the described subsystems. Through the course of implementing and defining recommendation functionality, the initial design choices in this document will be continuously revisited and evolved. The upcoming prototype iterations of the semantic search and recommendation will reflect such refined version of the concepts presented in this document.

REFERENCES

- [1] Maryam Alavi and Dorothy E. Leidner. Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly*, 25(1):107–136, 2001.
- [2] Brügge, B. & Dutoit, A.H. (2004), *Object-Oriented Software Engineering: Using UML, Patterns and Java*, Prentice Hall International.
- [3] Angel Cabrera and Elizabeth F. Cabrera. Knowledge-sharing dilemmas. *Organization Studies*, 23:687–710, 2002.
- [4] Davor Cubranic, Janice Singer, and Kellogg S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Softw. Eng.*, 31(6):446–465, 2005. Member-Gail C. Murphy.
- [5] Jonathon N. Cummings. Work groups, structural diversity, and knowledge sharing. *Management Science*, 50(3):352–364, 2004.
- [6] Thomas H. Davenport and Laurence Prusak. *Working Knowledge*. Harvard Business School Press, 1998.
- [7] Morten T. Hansen. The search-transfer problem: The role of weak ties in sharing knowledge across organization subunits. *Administrative Science Quarterly*, 44:82–111, 1999.
- [8] Kevin C. Desouza. Barriers to effective use of knowledge management systems in software engineering. *Commun. ACM*, 46(1):99–101, 2003.
- [9] Kevin C. Desouza and J. Roberto Evaristo. Managing knowledge in distributed projects. *Commun. ACM*, 47(4):87–91, 2004.
- [10] Hans-Jörg Happel, Ljiljana Stojanovic. D11: Conceptual model and specification for semantic search. Project Deliverable. IST-Project TEAM 35111. July 2007. <http://www.team-project.eu/documents/>
- [11] Google Inc. Google alerts, 10 2007.
- [12] Utku Irmak, Svilen Mihaylov, Torsten Suel, Samrat Ganguly, and Rauf Izmailov. Efficient query subscription processing for prospective search engines. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 1037–1038, New York, NY, USA, 2006. ACM Press.
- [13] Mik Kersten and Gail C. Murphy. Using task context to improve programmer productivity. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 1–11, New York, NY, USA, 2006. ACM.
- [14] Dirk Kukulenz and Alexandros Ntoulas. Answering bounded continuous search queries in the world wide web. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 551–560, New York, NY, USA, 2007. ACM Press.
- [15] Ronald Maier. *Knowledge Management Systems*. Springer, 2003.
- [16] Microsoft Inc. Windows live alerts, 10 2007.

- [17] Todd Mooradian, Birgit Renzl, and Kurt Matzler. Who trusts? personality, trust and knowledge sharing. *Management Learning*, 37(4):523–540, 2006.
- [18] Gary M. Olson and Judith S. Olson. Distance matters. *Human-Computer Interaction*, 15(2/3):139–178, 2000.
- [19] Cynthia T. Small and Andrew P. Sage. Knowledge management and knowledge sharing: A review. *Information, Knowledge, Systems Management*, 5(3):153–169, 2006.
- [20] Ljiljana Stojanovic, Hans-Jörg Happel: D15: Report describing state-of-the art in semantic recommendation. Project Deliverable. IST-Project TEAM 35111. October 2007. <http://www.team-project.eu/documents/>
- [21] The TEAM Consortium: D9: User requirements and conceptual architecture. IST-Project TEAM 35111. July 2007.
- [22] The TEAM Consortium: D24: Conceptual architecture of the TEAM integrated system. IST-Project TEAM 35111. February 2008.
- [23] Molly McLure Wasko and Samer Faraj. Why should i share? examining social capital and knowledge contribution in electronic networks of practice. *MIS Quarterly*, 29(1):35–57, 2005.
- [24] Beverly Yang and Glen Jeh. Retroactive answering of search queries. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 457–466, New York, NY, USA, 2006. ACM Press.