



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΔΙΑΧΕΙΡΙΣΗ ΓΝΩΣΗΣ ΣΤΗΝ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ

**(SEMANTIC KNOWLEDGE MANAGEMENT
IN SOFTWARE DEVELOPMENT)**

ΔΗΜΗΤΡΙΟΣ Π. ΠΑΝΑΓΙΩΤΟΥ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

στο πλαίσιο του Προγράμματος Μεταπτυχιακών Σπουδών
του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

ΑΘΗΝΑ, Απρίλιος 2011



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

**ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΔΙΑΧΕΙΡΙΣΗ ΓΝΩΣΗΣ
ΣΤΗΝ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ**

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του Δημήτριου Π. Παναγιώτου

Συμβουλευτική Επιτροπή:

Μέντζας Γρηγόριος, Καθηγητής Ε.Μ.Π.

(επιβλέπων)

Σαμουηλίδης Ι.Ε., Επίτιμος Καθηγητής Ε.Μ.Π.

Ψαρράς Ιωάννης, Καθηγητής Ε.Μ.Π.

Επταμελής Εξεταστική Επιτροπή:

Γρηγόριος Μέντζας

Καθηγητής Ε.Μ.Π.

Ιωάννης Ψαρράς

Καθηγητής Ε.Μ.Π.

Ι.Ε. Σαμουηλίδης

Ομότιμος Καθηγητής Ε.Μ.Π.

Δημήτριος Ασκούνης

Αν. Καθηγητής Ε.Μ.Π.

Δημήτριος Αποστόλου

Επικουρος Πανεπιστημίου Πειραιώς

Βασίλειος Ασημακόπουλος

Καθηγητής Ε.Μ.Π.

Γεώργιος Σαμάρας

Καθηγητής Πανεπιστημίου Κύπρου

Αθήνα, Απρίλιος 2011

Copyright © Δημήτριος Π. Παναγιώτου, 2011

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ' ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

*Αφιερώνεται στην οικογένεια
μου και στη σύζυγο μου Μαρία.*

Table of Contents

1	INTRODUCTION	27
1.1	PROBLEMS WHILE MANAGING KNOWLEDGE IN SOFTWARE DEVELOPMENT	27
1.2	MOTIVATION.....	29
1.3	MAIN OBJECTIVES	30
1.4	CONTRIBUTION OF THIS THESIS	32
1.5	STRUCTURE OF THE THESIS	33
1.6	RELATION TO RESEARCH PROJECTS	34
1.7	RELATION TO PUBLICATIONS.....	35
2	KNOWLEDGE MANAGEMENT IN SOFTWARE ENGINEERING	37
2.1	INTRODUCTION	37
2.2	MOTIVATION FOR KM IN SE	39
2.3	KM APPROACHES & TOOLS IN SE	42
2.3.1	<i>KM Approaches in SE</i>	<i>42</i>
2.3.2	<i>KM Tools in SE.....</i>	<i>50</i>
2.4	KNOWLEDGE MANAGEMENT SCHOOLS IN SE.....	59
2.4.1	<i>Technocratic schools.....</i>	<i>60</i>
2.4.2	<i>Behavioural schools</i>	<i>63</i>
2.4.3	<i>Implications for research and practice</i>	<i>65</i>
3	SOCIAL SEMANTIC DESKTOPS	67
3.1	SEMANTIC DESKTOPS	69
3.1.1	<i>Introduction</i>	<i>69</i>
3.1.2	<i>Chandler.....</i>	<i>71</i>
3.1.3	<i>DeepaMehta</i>	<i>73</i>
3.1.4	<i>Fenfire</i>	<i>74</i>
3.1.5	<i>Gnowsis.....</i>	<i>76</i>
3.1.6	<i>Haystack</i>	<i>78</i>
3.1.7	<i>IRIS semantic desktop</i>	<i>81</i>
3.1.8	<i>NEPOMUK.....</i>	<i>84</i>
3.1.9	<i>Comparison of Surveyed Implementations.....</i>	<i>86</i>
3.2	SEMANTIC DESKTOPS IN SOFTWARE ENGINEERING.....	88
3.3	SEMANTIC WIKIS.....	90
3.3.1	<i>Wikis</i>	<i>90</i>
3.3.2	<i>Semantic Wikis.....</i>	<i>94</i>
3.3.3	<i>COW.....</i>	<i>95</i>
3.3.4	<i>IkeWiki</i>	<i>98</i>
3.3.5	<i>Kaukolu</i>	<i>100</i>
3.3.6	<i>Makna.....</i>	<i>101</i>
3.3.7	<i>OntoWiki.....</i>	<i>104</i>
3.3.8	<i>OpenRecord</i>	<i>107</i>
3.3.9	<i>Platypus Wiki</i>	<i>108</i>
3.3.10	<i>Rhizome</i>	<i>110</i>
3.3.11	<i>Semantic MediaWiki</i>	<i>112</i>
3.3.12	<i>SemperWiki.....</i>	<i>114</i>
3.3.13	<i>SweetWiki</i>	<i>116</i>
3.3.14	<i>WiksAR.....</i>	<i>118</i>
3.3.15	<i>Comparison of surveyed implementations.....</i>	<i>121</i>
3.4	SEMANTIC WIKIS IN SOFTWARE ENGINEERING	124
3.4.1	<i>Wikis in Software Engineering</i>	<i>124</i>

3.4.2	<i>Semantic wikis in Software Engineering</i>	130
4	APPROACH	135
4.1	INTRODUCTION	135
4.2	KNOWLEDGE MANAGEMENT LIFECYCLE AND THE THESIS' APPROACH	137
4.2.1	<i>Knowledge Goals</i>	139
4.2.2	<i>Knowledge Identification</i>	139
4.2.3	<i>Knowledge Acquisition</i>	141
4.2.4	<i>Knowledge Development</i>	142
4.2.5	<i>Knowledge Distribution</i>	143
4.2.6	<i>Knowledge Use</i>	144
4.2.7	<i>Knowledge Preservation</i>	146
4.2.8	<i>Knowledge Measurement</i>	148
4.3	RESEARCH CHALLENGES OVERVIEW	148
4.4	CONCLUSIONS	152
4.4.1	<i>Summarization of Thesis' Contributions</i>	153
5	KNOWBENCH FUNCTIONALITY	155
5.1	KNOWBENCH AT A GLANCE	155
5.2	KNOWBENCH FUNCTIONALITY EXTRACTION FROM REQUIREMENTS	157
5.3	KNOWBENCH FUNCTIONALITY	159
5.3.1	<i>Manual Semantic Annotation</i>	159
5.3.2	<i>Knowledge Base Editing</i>	162
5.3.3	<i>P2P Services</i>	163
5.3.4	<i>Semantic Search</i>	164
5.3.5	<i>Software Development semantic wiki</i>	166
5.3.6	<i>Preferences and Configuration of Subsystems</i>	167
5.3.7	<i>Semi-automatic Semantic Annotation</i>	167
5.3.8	<i>Knowledge Base Graph-based Browsing</i>	172
6	TECHNICAL IMPLEMENTATION OF KNOWBENCH	175
6.1	HIGH-LEVEL KNOWBENCH TECHNICAL ARCHITECTURE	176
6.2	CORE COMPONENT	179
6.3	MODEL COMPONENT	180
6.4	COMMONS COMPONENT	180
6.5	ANNOTATION/ANNOTATION.SEMIAUTO COMPONENTS	180
6.6	KBEDITOR COMPONENT	181
6.7	KBGRAPH COMPONENT	182
6.8	KBGRAPH.PLUGIN COMPONENT	182
6.9	GLOBAL METADATA STORE COMPONENT	183
6.9.1	<i>LocalMDS</i>	184
6.9.2	<i>P2PMDS</i>	184
6.10	SEARCH COMPONENT	185
6.11	WIDGETS COMPONENT	186
6.12	WIKI COMPONENT	187
7	KNOWBENCH WALKTHROUGH	189
7.1	OVERVIEW	189
7.2	CONFIGURATION	193
7.2.1	<i>Peer-to-Peer Configuration</i>	194
7.2.2	<i>Search Configuration</i>	200
7.3	KNOWLEDGE ACQUISITION	204
7.4	KNOWLEDGE DEVELOPMENT	206
7.4.1	<i>Manual Knowledge Development</i>	206
7.4.2	<i>Semi-automatic Knowledge Development</i>	215
7.4.3	<i>Wiki-based Knowledge Development</i>	218

7.5	KNOWLEDGE DISTRIBUTION	223
7.5.1	<i>Joining the P2P Network</i>	223
7.5.2	<i>Sharing Knowledge</i>	226
7.6	KNOWLEDGE USAGE	227
7.6.1	<i>Knowledge Base Visualization</i>	227
7.6.2	<i>Search</i>	232
7.6.3	<i>Views</i>	237
7.7	KNOWLEDGE PRESERVATION	242
8	EVALUATION OF KNOWBENCH	247
8.1	THE GQM METHOD.....	247
8.2	AREAS OF EVALUATION (GOALS)	249
8.2.1	<i>Basic Perceptions of Knowledge Management</i>	250
8.2.2	<i>General Perceptions of KnowBench's Efficiency and Effectiveness</i>	250
8.2.3	<i>The System</i>	251
8.3	METHOD OF ANALYSIS	251
8.4	BASIC PERCEPTIONS OF KNOWLEDGE MANAGEMENT	253
8.5	KNOWBENCH SUPPORT FOR KNOWLEDGE MANAGEMENT (LIFECYCLE) FUNCTIONS	254
8.5.1	<i>Knowledge Identification in KnowBench</i>	254
8.5.2	<i>Knowledge Acquisition in KnowBench</i>	256
8.5.3	<i>Knowledge Development in KnowBench</i>	258
8.5.4	<i>Knowledge Sharing in KnowBench</i>	260
8.5.5	<i>Knowledge Usage in KnowBench</i>	262
8.5.6	<i>Knowledge Preservation in KnowBench</i>	264
8.6	SYSTEM PERFORMANCE.....	265
8.6.1	<i>Usability</i>	265
8.6.2	<i>Reliability</i>	267
8.6.3	<i>Scalability & Availability</i>	267
8.6.4	<i>Security</i>	268
8.6.5	<i>Deployment</i>	269
8.7	EVALUATION RESULTS SYNOPSIS	270
9	CONCLUSIONS AND FUTURE WORK	273
9.1	LIMITATIONS AND POSSIBLE IMPROVEMENTS	275
9.2	FURTHER RESEARCH	276
	REFERENCES.....	277
	PUBLICATIONS RELATED TO THE THESIS	299
APPENDIX A.	ONTOLOGIES	301
SECTION A.1	WHAT HAS TO BE MODELLED?	301
SECTION A.2	METHODOLOGY.....	301
SECTION A.3	CONTENT LAYER	302
A.3.1	<i>Knowledge artefact ontology</i>	303
A.3.2	<i>Problem/solution ontology</i>	314
A.3.3	<i>Annotation ontology</i>	318
SECTION A.4	ORGANIZATIONAL LAYER	320
SECTION A.5	KNOWBENCH ONTOLOGY	323
APPENDIX B.	EVALUATION QUESTIONNAIRES	325
SECTION B.1	GENERAL KNOWLEDGE MANAGEMENT QUESTIONS	325
SECTION B.2	KNOWBENCH PERFORMANCE DURING KNOWLEDGE IDENTIFICATION.....	326
SECTION B.3	KNOWBENCH PERFORMANCE DURING KNOWLEDGE ACQUISITION	328
SECTION B.4	KNOWBENCH PERFORMANCE DURING KNOWLEDGE DEVELOPMENT.....	330
SECTION B.5	KNOWBENCH PERFORMANCE DURING KNOWLEDGE SHARING.....	334

SECTION B.6	KNOWBENCH PERFORMANCE DURING KNOWLEDGE USAGE	337
SECTION B.7	KNOWBENCH PERFORMANCE DURING KNOWLEDGE PRESERVATION.....	338
SECTION B.8	GENERAL PERCEPTIONS OF KNOWBENCH PERFORMANCE	339
SECTION B.9	SYSTEM-SPECIFIC EVALUATION	341

Figures Index

Figure 1.1: The Quality Improvement Paradigm (from (McGarry et al., 1994))	43
Figure 1.2: The Experience Factory Model	44
Figure 1.3: The Experience Factory Model with the new feedback loop of the Knowledge Dust to Pearls approach.	45
Figure 1.4: The TSP process	48
Figure 3.1: Realising the Social Semantic Desktop	68
Figure 3.2: The Chandler user interface	72
Figure 3.3: DeepaMehta's user interface showing a topic map	73
Figure 3.4: The RDF browser of Fenfire	74
Figure 3.5: FenPDF, using Fenfire's user interface technologies	76
Figure 3.6: The Gnowsis user interface (Enquire2)	77
Figure 3.7: The Gnowsis architecture	78
Figure 3.8: Haystack viewing a user's inbox collection	80
Figure 3.9: The IRIS user interface	83
Figure 3.10: High-level architecture of the Social semantic desktop	85
Figure 2.11: COW ontology editor	97
Figure 2.12: COW query template	98
Figure 2.13: IkeWiki user interface	99
Figure 2.14: Kaukolu user interface	101
Figure 2.15: Makna syntax	103
Figure 2.16: Makna predicate assistant	103
Figure 2.17: Makna user interface	104
Figure 2.18: OntoWiki user interface	105
Figure 2.19: OpenRecord user interface	108
Figure 2.20: Platypus Wiki user interface	109
Figure 2.21: Edit mode and metadata view of the Rhizome Wiki user interface	111
Figure 2.22: Semantic MediaWiki user interface	114
Figure 2.23: SemperWiki user interface	115
Figure 2.24: Tags suggested in SweetWiki as the user enters keywords	117
Figure 2.25: Faceted navigation in SweetWiki	118
Figure 2.26: WikSAR user interface	119
Figure 2.27: WikSAR with interactive graph switched on	120

Figure 3.1: The building blocks of knowledge management systems (Probst, 1998)	139
Figure 4.2: Research challenges of the thesis	149
Figure 4.1: Semantic Annotation resulting metadata in RDF/XML notation.....	162
Figure 4.2: Configuration of the root concepts and/or annotation root concepts presented to the KnowBench UI	163
Figure 4.3: The ontology learning layer cake	169
Figure 5.1: KnowBench high-level Architectural Diagram.....	178
Figure 6.1: KnowBench Menu and Toolbar Eclipse Integration.....	190
Figure 6.2: KnowBench Eclipse Preferences Integration	190
Figure 6.3: KnowBench Eclipse Search Integration.....	191
Figure 6.4: KnowBench Eclipse Perspective Integration	191
Figure 6.5: KnowBench Eclipse Status Integration.....	192
Figure 6.6: KnowBench Toolbar	192
Figure 6.7: KnowBench Menu.....	193
Figure 6.8: Eclipse Preferences.....	193
Figure 6.9: KnowBench P2P Preferences	195
Figure 6.10: Adding a Rule to P2P Policy File.....	196
Figure 6.11: Adding Properties to P2P Rule.....	197
Figure 6.12: Adding Constraints for Object Properties of P2P Rule.....	198
Figure 6.13: Adding Groups to P2P Rule	200
Figure 6.14: Service Crawlers Preferences	201
Figure 6.15: Error Message Dialog for consistency checking (Jira)	203
Figure 6.16: Adding a new Crawl Repository	204
Figure 6.17: Starting and Stopping Crawler	205
Figure 6.18: Progress during crawling.....	205
Figure 6.19: Crawling complete	205
Figure 6.20: Opening knowledge editor	206
Figure 6.21: Knowledge editor	207
Figure 6.22: Searching in the ontology tree.....	208
Figure 6.23: Instance part of the knowledge editor	208
Figure 6.24: Manual creation of knowledge items	209
Figure 6.25: Defining values for object properties	210
Figure 6.26: Saving a new individual	210
Figure 6.27: Annotation Toolbar & Menu.....	211
Figure 6.28: Context menu for creating annotation.....	211

Figure 6.29: Annotation pop-up.....	212
Figure 6.30: Searching an annotation tag/class.....	213
Figure 6.31: Creation of a new tag.....	214
Figure 6.32: Result of tag creation.....	215
Figure 6.33: Different sources for semi-automatic annotation	215
Figure 6.34: Choosing Source Code Corpus.....	216
Figure 6.35: Processing Source Code Corpus.....	216
Figure 6.36: Semi-Automatic Annotation Editor.....	217
Figure 6.37: Adding Annotations to Source Code Corpus	218
Figure 6.38: Memory requirement for semi-automatic annotation.....	218
Figure 6.39: Wiki Syntax Explanation.....	219
Figure 6.40: Creating a new Wiki Instance	219
Figure 6.41: Initial Wiki Instance	220
Figure 6.42: Wiki Auto-complete	220
Figure 6.43: Wiki Auto-complete sub-concept.....	221
Figure 6.44: Wiki Property Auto-complete	221
Figure 6.45: Wiki Object Property Values Auto-complete	222
Figure 6.46: Wiki Instance Name Dialog	222
Figure 6.47: Wiki Editor Title Refreshed	223
Figure 6.48: P2P Join Network Toolbar	223
Figure 6.49: Introduction about connecting to the P2P network	224
Figure 6.50: P2P Connection Dialog	225
Figure 6.51: P2P Connected Dialog.....	226
Figure 6.52: P2P Join Network Toolbar when connection is established.....	226
Figure 6.53: P2P Join Network Toolbar when knowledge is shared.....	226
Figure 6.54: Tree-based representation of knowledge base.....	228
Figure 6.55: KB Graph Toolbar.....	228
Figure 6.56: Graph-based representation of knowledge base.....	229
Figure 6.57: Graph Nodes.....	231
Figure 6.58: Expanded Concept Nodes	231
Figure 6.59: KnowBench Eclipse Search Integration.....	232
Figure 6.60: Keyword Search Dialog	233
Figure 6.61: Keyword Search Problem.....	233
Figure 6.62: Structured Query Search Dialog.....	234

Figure 6.63: Structured Query Search Dialog – Adding Properties to the Query	235
Figure 6.64: Structured Query Search Dialog – Property Values.....	236
Figure 6.65: KnowBench Keyword Search Results View.....	237
Figure 6.66: Relationships between Search Results view and other KnowBench options...	237
Figure 6.67: Keyword Search Results Refinement.....	238
Figure 6.68: Keyword Refined Search Results.....	238
Figure 6.69: Structured Query Search Results Dialog – P2P Search Results depicting the solution found for the query shown in Figure 7.64.....	238
Figure 6.70: Retrieving attributes for the P2P search results.....	239
Figure 6.71: Semantic Query Search Results Dialog.....	240
Figure 6.72: View Annotation Toolbar Button.....	240
Figure 6.73: Annotation Views.....	241
Figure 6.74: Wiki Browser	242
Figure 6.75: Modification of Individuals.....	243
Figure 6.76: Modified Individual.....	243
Figure 6.77: Confirmation of Modification	244
Figure 6.78: Removal of Individuals	244
Figure 6.79: Editing a Wiki Page.....	245
Figure 7.1: Schematic overview of the GQM approach (Basili et al., 1994)	248
Figure 7.2: Phases of the GQM process (Basili et al., 1994).....	249
Figure 7.3: Example of Data Analysis.....	253
Figure 7.4: Goal measurement for Knowledge identification	255
Figure 7.5: Goal measurement for Knowledge Acquisition of existing knowledge.....	257
Figure 7.6: Goal measurement for Knowledge Development	258
Figure 7.7: Goal measurement for Knowledge Sharing	260
Figure 7.8: Goal measurement for Knowledge Usage - Search.....	262
Figure 7.9: Goal measurement for Knowledge Preservation.....	265
Figure 7.10: Goal measurement for Usability.....	266
Figure 7.11: Goal measurement for Reliability	267
Figure 7.12: Goal measurement for Scalability and Availability	268
Figure 7.13: Goal measurement for Security.....	269
Figure 7.14: Goal measurement for Deployment	270
Figure 7.15: Evaluation results synopsis – knowledge management lifecycle support.....	270
Figure 7.16: Evaluation results synopsis – FURPS	271
Figure A.1: The ka:KnowledgeArtefact class and its specialisations.....	305

Figure A.2: A part of the source code model.....	307
Figure A.3: Modelling of expertise and experience for a software engineering artefact. Grey colour means that the entities are not defined in the Organisational ontology.	322
Figure A.4: Relations between Content and Organizational ontologies.....	324

Tables Index

Table 1.1: Summary of typical competence management tools' features	52
Table 1.2: Earl's schools of knowledge management – adopted from (Bjornson & Dingsoyr, 2008)	59
Table 3.1: Overview of surveyed semantic desktops' features.....	88
Table 2.2: Overview of surveyed semantic wikis' features.....	123
Table 2.3: Applications of Wikis in software engineering	130
Table 3.1: Research challenges and positioning of the thesis' approach in relation to the state-of-the-art.....	152
Table 4.1: Mapping of requirements derived from chapter 4 to desired KnowBench functionality	159
Table 6.1: Description of KnowBench toolbar	193
Table 6.2: Generating a group-subject and its credentials.....	199
Table 6.3: Overview of the source types	202
Table 6.4: Icons in the knowledge tree	207
Table 7.1: Percentage of answers above threshold	254
Table 7.2: Percentage of answers above threshold	256
Table 7.3: Percentage of answers above threshold	258
Table 7.4: Percentage of answers above threshold	260
Table 7.5: Percentage of answers above threshold	262
Table 7.6: Percentage of answers above threshold	264
Table 7.7: Percentage of answers above threshold	266
Table 7.8: Percentage of answers above threshold	267
Table 7.9: Percentage of answers above threshold	267
Table 7.10: Percentage of answers above threshold	268
Table 7.11: Percentage of answers above threshold	269

ΠΕΡΙΛΗΨΗ

Η παρούσα διδακτορική διατριβή εστιάζει στον τομέα της σημασιολογικής διαχείρισης γνώσης στην ανάπτυξη λογισμικού και προτείνει μια προσέγγιση που βασίζεται στον κύκλο ζωής της διαχείρισης γνώσης ο οποίος αποτελείται από τα ακόλουθα δομικά συστατικά: αναγνώριση, ανάκτηση, ανάπτυξη, διανομή, διατήρηση και χρήση της γνώσης. Ο κύριος ερευνητικός στόχος της διατριβής είναι η ανάπτυξη και εφαρμογή μιας **καινοτόμου προσέγγισης για τη διαχείριση γνώσης στην ανάπτυξη λογισμικού**, καθώς και η σχεδίαση, ανάπτυξη και αξιολόγηση ενός **συστήματος διαχείρισης γνώσης που βοηθά τους προγραμματιστές λογισμικού και στηρίζεται σε τεχνολογίες κοινωνικών σημασιολογικών επιφανειών εργασίας**. Λαμβάνοντας υπόψη τα προβλήματα των υπάρχοντων μεθόδων για διαχείριση γνώσης στον τομέα της ανάπτυξης λογισμικού, είναι προφανές ότι χρειάζονται καινούριες πρακτικές για να καταστεί πιο εύκολη η εκμετάλλευση της γνώσης σε εταιρείες ανάπτυξης λογισμικού. Το κύριο πρόβλημα είναι ότι παρόλο που η ρητή γνώση αποθηκεύεται και επαναχρησιμοποιείται, η άρρητη γνώση παραμένει στους προγραμματιστές και σπάνια επαναχρησιμοποιείται από άλλους υπαλλήλους μέσα στην εταιρεία ανάπτυξης λογισμικού.

Η συμβολή της διατριβής εντοπίζεται σε τρία βασικά θέματα:

- Μια προσέγγιση για τη **διαχείριση γνώσης στην ανάπτυξη λογισμικού** (υποστηρίζοντας τους προγραμματιστές σε **ολόκληρο τον κύκλο ζωής διαχείρισης γνώσης**) εκμεταλλευόμενη τεχνολογίες **κοινωνικών σημασιολογικών επιφανειών εργασίας (social semantic desktops)**
- Ένα σημασιολογικό σύστημα διαχείρισης γνώσης (KnowBench) που υποστηρίζει τους προγραμματιστές στην καθημερινή τους εργασία. Το KnowBench απαρτίζεται από τα ακόλουθα μέρη:
 - ο Χειροκίνητη και ημιαυτόματη σημασιολογική επισημείωση
 - ο Επεξεργασία γνωσιακής βάσης
 - ο Διαμοιρασμός μετα-δεδομένων με χρήση ομότιμων δικτύων (P2P)
 - ο Σημασιολογική αναζήτηση
 - ο Σημασιολογικό Wiki ανάπτυξης λογισμικού (DevWiki)

- ο Πλοήγηση στη γνωσιακή βάση χρησιμοποιώντας γραφική αναπαράσταση
- Αθροιστική (summative) αξιολόγηση του KnowBench εφαρμόζοντας τη μέθοδο Goal-Question-Metric (GQM) στο feedback για τον υπολογισμό των στόχων της αξιολόγησης σε τέσσερεις οργανισμούς (INTRASOFT, LIPSZ, THALES, TXT)

Λέξεις κλειδιά

Ανάπτυξη Λογισμικού, Knowledge Workbench, Διαχείριση Γνώσης στην Ανάπτυξη Λογισμικού, KnowBench, Κοινωνική Σημασιολογική Επιφάνεια Εργασίας.

ABSTRACT

This doctoral thesis focuses on the domain of semantic-based knowledge management in software development and proposes an approach that is based on the knowledge management lifecycle which is synthesized by the following building blocks: identification, acquisition, development, distribution, preservation, and use of knowledge. The main research goal of the thesis is the development and application of an ***innovative approach for managing knowledge in software development***, as well as the design, development and evaluation of ***a knowledge management system that aids software developers and is powered by social semantic desktop technologies***. Considering the problems of the existing methods for managing knowledge in the software development domain, it is prominent that new practices are needed in order to better exploit knowledge in software development houses in an easier and more flexible manner. The main problem is that even though explicit knowledge is captured and reused, implicit knowledge remains in the developers' heads and is seldom reused by other employees inside the software development company.

The contribution of this thesis is constituted by three main topics:

- An approach towards ***managing knowledge in software development*** (supporting developers in the ***whole knowledge management lifecycle***) by exploiting ***social semantic desktop*** technologies
- A semantic-based knowledge management system (KnowBench) that assists software developers in their daily work. KnowBench is constituted by the following parts:
 - Manual and semi-automatic semantic annotation
 - Knowledge base editing
 - Meta-data sharing using P2P services
 - Semantic search
 - Software Development semantic Wiki (DevWiki)
 - Knowledge base graph-based browsing

- KnowBench's summative evaluation by applying the Goal-Question-Metric (GQM) method on the collected feedback for the calculation of the evaluation goals in four organizations (INTRASOFT, LIPSZ, THALES, TXT)

Keywords

Software Development, Knowledge Workbench, Knowledge Management in Software Development, KnowBench, Social Semantic Desktop.

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα διδακτορική διατριβή αποτελεί το επιστέγασμα μιας προσπάθειας έξι ετών, στα πλαίσια του προγράμματος μεταπτυχιακών σπουδών του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Η συναναστροφή με συναδέλφους, αλλά και το κλίμα δημιουργικότητας αποτέλεσαν βασικές πηγές έμπνευσης και συνέβαλλαν σημαντικά στη βελτίωση της προσωπικής αντιμετώπισης και επίλυσης ερευνητικών προκλήσεων.

Το αποτέλεσμα που παρουσιάζεται στις σελίδες αυτές οφείλεται στο μέγιστο βαθμό στη βοήθεια και στην καθοδήγηση που είχα από τον επιβλέποντα Καθηγητή κ. Γρ. Ν. Μέντζα. Του οφείλω ιδιαίτερες ευχαριστίες για τις ευκαιρίες που μου προσέφερε και την πίστη του σε μένα. Τα μαθήματα επιστημονικής κατάρτισης, ερευνητικού ζήλου, αλλά και ηθικής ακεραιότητας που πήρα από αυτόν αποτελούν τα σημαντικότερα εφόδια για τη μελλοντική μου πορεία.

Θα ήθελα να ευχαριστήσω τα άλλα δύο μέλη της τριμελούς εισηγητικής επιτροπής, τον Επίτιμο Καθηγητή Ι.Ε. Σαμουηλίδη και τον Καθηγητή κ. Ι. Ψαρρά, καθώς και τους Καθηγητές κ. Β. Ασημακόπουλο, κ. Γ. Σαμάρα, τον Αναπληρωτή Καθηγητή κ. Δ. Ασκούνη και τον Λέκτορα κ. Δ. Αποστόλου για την τιμή που μου έκαναν να συμμετάσχουν στην επιτροπή εξέτασης της διατριβής.

Θέλω επίσης να ευχαριστήσω τους συναδέλφους μου Χαράλαμπο Μαγγούτα, Κώστα Χρηστίδη, Ευθύμιο Μπόθο και Φώτη Παρασκευόπουλο που υπήρξαν αρωγοί και συμπαραστάτες σε όλη αυτή την πορεία.

Ολοκληρώνοντας θα ήθελα να απευθύνω ένα μεγάλο ευχαριστώ στους γονείς μου, στον αδερφό μου και στη σύζυγο μου, για την αμέριστη αγάπη και ηθική υποστήριξη που μου προσέφεραν όλα αυτά τα χρόνια.

Δημήτριος Π. Παναγιώτου

Απρίλιος 2011

1 INTRODUCTION

This doctoral thesis focuses on the domain of semantic-based knowledge management in software development and proposes an approach that is based on the knowledge management lifecycle which is synthesized by the following building blocks: identification, acquisition, development, distribution, preservation, and use of knowledge. The main research goal of the thesis is the development and application of an ***approach for managing knowledge in software development***, as well as the design, development and evaluation of ***a knowledge management system that aids software developers and is powered by social semantic desktop technologies***. Considering the problems of the existing methods for managing knowledge in the software development domain, it is prominent that new practices are needed in order to better exploit knowledge in software development houses in an easier and more flexible manner. The main problem is that even though explicit knowledge is captured and reused, implicit knowledge remains in the developers' heads and is seldom reused by other employees inside the software development company.

This chapter is structured as follows. In section 1.1, the problems occurring while managing knowledge when developers write software are highlighted. Section 1.2 describes the challenges that motivated the development of the proposed approach and the respective system. In section 1.3 the main objectives of the proposed approach, as originate from the research challenges, are presented. Section 1.4 provides an overview of the main contributions of the present doctoral thesis. Section 1.5 describes how the thesis is structured and in section 1.6 the research projects, which supported partially the present thesis, as well as their relation to the thesis, are discussed. Finally, a discussion, about how the structure of the thesis is related to the papers published, is given in section 1.7.

1.1 Problems while Managing Knowledge in Software Development

Managing knowledge in software development is very important, since software development is a human and knowledge intensive activity. ***The main asset of a software organization consists of its intellectual capital***. Software development is a “design type process” where every person involved has to make a large number of decisions, each of them with several possible choices (Rus, Lindvall, & S. S. Sinha, 2001). Additionally, software development requires that software developers create and share new knowledge during their

daily work. *Since software development is a highly collaborative task, developers are in need of simple and easy-to-use tools that also enable collaborative work.*

Software industry initiatives like the Capability Maturity Model¹ try to establish stable software processes that are independent of individual developers. Thus, *collaboration and flexible ways of solving problems are necessary in order to exploit the synergy between developers and past experience. This experience should be captured inside a software development house in order to be available to all employees instead of being just in the head of one employee.*

When a developer is frustrated investigating source code that he has never seen before (i.e. when extending a third party's software system) and is not capable of understanding its rationale the above mentioned experience could be useful. On the other hand, there are many situations that find a developer seeking source code that is already developed by others. He might not be aware of its existence, or even if he is, he is not able to find it effectively (D. Panagiotou & G. Mentzas, 2009a), (Dimitris Panagiotou & Gregoris Mentzas, 2010).

Usually, developers rely on personal knowledge and experience, but as software development projects grow larger it becomes a group activity where individuals need to communicate and coordinate. Individual knowledge has to be shared and leveraged at a project and organization level, and this is exactly what knowledge management proposes.

Companies developing information systems have failed to learn effective means for problem solving to such an extent that they have learned to fail, according to an article by Lyytinen and Robey (Lyytinen & Robey, 1999). *One suggested mean to overcome this problem is an increased focus on knowledge management.*

Knowledge has to be collected, organized, stored, and easily retrieved when it needs to be applied. Probst (Probst, 1998) has well described the building blocks of KM systems: identification, acquisition, development, distribution, preservation, and use of knowledge. Many knowledge problems occur because organizations neglect one or more of these building blocks.

¹ http://en.wikipedia.org/wiki/Capability_Maturity_Model

1.2 Motivation

Authors in (Natali & Falbo, 2002) note that knowledge management in software engineering might be exploited to capture software engineering knowledge and experience produced during the execution of projects. Even if software differs from one project to another there is still similar knowledge that can be reused in order to fasten the development process. This can assist in the avoidance of problems based on the experience gathered inside an organization.

Exactly this need of managing knowledge in software development is the motive of this thesis. The thesis approaches managing knowledge in software development based on the entire knowledge management lifecycle.

Although benefits can be derived from individual tools addressing separate software development activities, there is a lack of tools supporting the whole knowledge management process. Indeed, an envisioned tool is needed, that could integrate a collection of services that facilitate software development activities by applying knowledge management paradigm in capturing, storing, disseminating, and reusing knowledge created during the software development process as well as in integrating existing sources.

Furthermore, recent research has shown that semantic web technologies can provide the driving force to better manage knowledge in software development activities – ontology-based knowledge management in software development. Since 2005 the Semantic Web Enabled Software Engineering (SWESE) conference takes place every year with promising results. The Semantic Web Best Practice and Deployment Working Group (SWBPD) in W3C included a Software Engineering Task Force (SETF) to investigate potential benefits of applying semantics in software engineering processes. As noted by SETF (Knublauch, Oberle, Tetlow, & Wallace, 1999), advantages of applying semantic web technologies to software engineering include reusability and extensibility of data models, improvements in data quality, enhanced discovery, and automated execution of workflows. Thus, ***another motive for the thesis is the enhancement of knowledge management in software development by exploiting powerful semantic web technologies.***

Nowadays there is a lack of knowledge management systems that assist in software development embedded in the developer's existing Integrated Development Environment - IDE. This fact was a motive for developing such a knowledge management system inside the

Eclipse IDE² as a proof of concept (namely KnowBench - (Knowledge workBench)), supporting in that way advanced capabilities in the distributed engineering and management of software systems. In particular, a decentralized and semantic-based framework is proposed for sharing knowledge about software implementation that is seamlessly integrated into Eclipse. This framework is general enough to be principally applied on every phase of the software development process and in every environment.

1.3 Main Objectives

In an attempt to overcome the aforementioned challenges, this doctoral thesis proposes an approach and a method to apply semantic-based knowledge management in software development as well as a respective system which realizes a framework and its evaluation in real case scenarios during software development in software companies. The evaluation is subjective, in the sense that it is performed by software developers (end-users). Furthermore it is conducted in a summative manner (see chapter 8). In other words it represents users' perceptions about the efficiency and quality of the proposed system.

At the heart of the proposed framework lies a holistic knowledge management approach for software development based on all the building blocks of the knowledge management lifecycle. More specifically the approach suggests that the basic building blocks of knowledge management systems are used: identification, acquisition, development, distribution, preservation, and use of knowledge.

The main research objectives of the proposed approach and system are directly connected to the challenges and limitations of the current state of practice while developing software and the way knowledge is treated and managed, which were discussed in section 1.1.

This thesis addresses the need for knowledge management in software development in distributed environments in order to foster knowledge sharing, supporting in this way advanced capabilities in the distributed engineering and management of software systems.

² <http://www.eclipse.org/>

Another objective of the thesis is to develop an open-source software system, seamlessly integrated in a software development environment (Eclipse IDE) for enabling decentralised and knowledge sharing for software development through:

- Semantic desktop, a component embedded in the software development environment that provides a graphical user interface (GUI) for knowledge manipulation, as well as an integration platform for other knowledge sharing components,
- Semantic search, a component that supports ontology-based proximity search for relevant knowledge items,
- Metadata repository that enables efficient structuring and persistent storage of the acquired knowledge,
- Metadata-based P2P infrastructure for decentralised communication between several local semantic desktops which leads to a social semantic desktop environment and
- Set of ontologies that are used by the proposed system.

As the integration platform the Eclipse IDE is used, although the framework is realized generic enough in order to be easily applied in other IDEs.

In order to achieve this aim two supporting objectives were defined:

1. To approach in an efficient manner sharing knowledge in distributed software communities, that can be seamlessly integrated in a software development environment, through:
 - semantic-based modelling of a local knowledge sharing environment of a software developer (e.g. ontology-based models of a knowledge artefact, available knowledge sources, etc.)
 - semantic-based modelling of a user's preferences regarding knowledge support
 - semantic-based knowledge manipulation mechanism that enables undisruptive knowledge acquisition, knowledge access based on semantic-driven similarity
 - ontology-based knowledge repository that enables representation of informal knowledge by defining semantic dependencies between knowledge artefacts and

- ontology/metadata-based peer-to-peer communication model that supports semantically-routed knowledge flow between local knowledge sharing environments, in order to ensure decentralised but synchronised decision making.
2. To investigate the usefulness of this approach and the resulting knowledge sharing software system in practical scenarios. The proposed platform was piloted and evaluated in four different scenarios: (i) a distributed software development case that includes a large company with its offshore partners, (ii) a distributed software development in nearshoring scenarios, (iii) an Open Source Community, and (iv) a middle-sized software company, in order to illustrate the generality of the proposed approach.

1.4 Contribution of this Thesis

The contribution of this thesis is constituted by three main topics:

- An approach towards *managing knowledge in software development* (supporting developers in the *whole knowledge management lifecycle*) by exploiting *social semantic desktop* technologies
- A semantic-based knowledge management system (KnowBench) that assists software developers in their daily work. KnowBench is constituted by the following parts:
 - Manual semantic annotation
 - Semi-automatic semantic annotation
 - Knowledge base editing
 - P2P services
 - Semantic search
 - Software Development semantic Wiki (DevWiki)
 - Knowledge base graph-based browsing
- KnowBench's summative evaluation by applying the GQM method on the collected feedback for the calculation of the evaluation goals in four pilot organizations (INTRASOFT, LIPSZ, THALES, TXT)

1.5 Structure of the Thesis

The thesis consists of nine (9) chapters and is structured as follows:

After this introductory chapter, chapter 2 presents current knowledge management practices in a broader domain than software development namely software engineering.

Social semantic desktops are reviewed in chapter 3. The ingredients of social semantic desktops are semantic desktops and semantic wikis. Thus, generic semantic desktops are presented first, as well as a comparison of these systems. Then semantic desktop in software engineering are presented. The same structure is followed for semantic wikis as well, meaning that generic semantic wikis are reviewed (and compared), followed by a presentation of implementations in the software engineering domain.

Chapter 4 is the heart of the thesis. It describes the approach towards managing knowledge in software development in an efficient manner. The main research goal is analysed which leads to the development of this approach. The main research challenges are presented, which lead to requirements for the proposed system of this thesis.

Chapter 5 analyses the functionality of the proposed system. The way that the requirements extracted in chapter 4 are met by the functionality of the system is also shown in this chapter. Then, chapter 6 presents the technical implementation details of the system.

In chapter 7, the system walkthrough is shown, highlighting all aspects of the system usage, ranging from the graphical user interface manipulation to the configuration and the usage all components that constitute the integrated system in the Eclipse IDE.

Chapter 8 reports on all pertinent aspects of the system summative evaluation. It starts with the details of the evaluation framework. The GQM method is shortly described and also the main areas of evaluation i.e. GQM Goals. Then, the method of analysis applied on the collected feedback for the calculation of the evaluation goals is described. Finally, the collected feedback from four pilot organizations (INTRASOFT, LIPSZ, THALES, TXT) and the interpretation of the observed trends in users' responses are presented.

Finally, in Chapter 9, the conclusion of the thesis is presented, including its contributions in contemporary research as well as limitations of the thesis' approach and the proposed system and possible implications for further work.

1.6 Relation to Research Projects

It should be noted that the present doctoral thesis was partially supported by the European Commission through the Information Society Technologies (IST) project TEAM³ (Tightening knowledge sharing in distributed software communities by applying semantic technologies, FP6-35111), and NEPOMUK⁴ (Networked Environment for Personalized, Ontology-based Management of Unified Knowledge - The Social Semantic Desktop, FP6-27705).

TEAM

The objective of TEAM project was to address the need for a knowledge sharing environment, supporting in that way advanced capabilities in the distributed engineering and management of software systems. In particular, TEAM proposed a decentralized, personalized, context-sensitive and semantic-based framework for sharing knowledge about software implementation that was seamlessly integrated into a software development environment (Eclipse IDE).

The research reported in this thesis is related mainly to the work that was done in the context of the TEAM project regarding knowledge management in software development as well as in the semantic enhancements of the envisioned system. Some of the work in TEAM is out of the context of the present doctoral thesis, while the latter uses some system components which were developed in TEAM, either by other research partners (metadata-based P2P (P2PMDS) which is based on GridVine, semantic search engine) or by joint efforts of other partners with us (metadata component - LocalMDS).

NEPOMUK

NEPOMUK intended to realize and deploy a comprehensive solution – methods, data structures, and a set of tools – for extending the personal computer into a collaborative environment, which improved the state of art in online collaboration and personal data

³ <http://www.team-project.eu/>

⁴ <http://nepomuk.semanticdesktop.org/>

management and augmented the intellect of people by providing and organizing information created by single or group efforts.

The *social semantic desktop*. This enhanced personal workspace (the Desktop) is Semantic since it gives information a well-defined meaning, making it processable by the computer. It is Social since it supports the interconnection and exchange with other desktops and their users.

The research reported in this thesis was affected by the NEPOMUK project in the sense that more insights were gained about social semantic desktop technologies (semantic wikis and desktops) as well as aspects of social collaboration (which were exploited in order to interconnect Eclipse IDE desktops in the proposed system and assist in collaborative knowledge authoring and navigation supported by lightweight semantic wiki technologies – *social semantic desktop environment*) and P2P systems in general.

1.7 Relation to Publications

This thesis resulted in two (2) journal publications and eight (8) conference presentations. This section describes how the structure of the thesis is related to these publications. The list of publications can be found at the “Publications related to the thesis” section of this thesis. Although the research contributions of a single publication may concern more than one chapters, in the following the relation of each chapter to the most relevant publication(s) is presented.

- The state-of-the-art in Social Semantic Desktops which is described in Chapter 3 has been published in (D. Panagiotou & G. Mentzas, 2007) and (Maalej, D. Panagiotou, & Happel, 2008).
- Part of the approach of the thesis which is described in Chapter 4 has been published in (Dimitris Panagiotou, Paraskevopoulos, & Gregoris Mentzas, 2011)
- The KnowBench system which is described in Chapters 5 and 6 has been published in (D. Panagiotou & G. Mentzas, 2009a), (D. Panagiotou & G. Mentzas, 2009b), (D. Panagiotou & G. Mentzas, 2008), (Dimitris Panagiotou, Paraskevopoulos, & Gregoris Mentzas, 2011) and (Dimitris Panagiotou et al., 2011).

- The evaluation of the KnowBench system which is described in Chapter 8 has been published in (Dimitris Panagiotou, Paraskevopoulos, & Gregoris Mentzas, 2011).

2 KNOWLEDGE MANAGEMENT IN SOFTWARE ENGINEERING

This chapter describes the state of the research and practice in the area of Knowledge Management (KM) in Software Engineering (SE).

2.1 Introduction

Demarest (Demarest, 1997) defines knowledge management as the systematic: (a) underpinning, (b) observation, (c) instrumentation and (d) optimization of the firm's knowledge economies.

“These activities – or perhaps process areas – are sequential. That is to say, it is not possible to optimize a firm's knowledge economies until they are instrumented; instrumentation is selective measurement based on empirical observation, and observation of the for-the-most-part underground knowledge economies within the firm requires that these economies be brought out into the light of day, made official, and formally underpinned with technology, process and policy that makes construction, embodiment, dissemination and use of knowledge officially – sanctioned and formally – valued activities within the firm”.

“Software engineering is (1) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software, and (2) the study of approaches as in (1)”⁵.

Knowledge management is described by Davenport and Prusak (Davenport & Prusak, 2000) as “a fluid mix of framed experience, values, contextual information, and expert insights and grounded intuitions that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the minds of the knower. In software organisations, knowledge often becomes embedded not only in documents or repositories, but also in organisational routines, processes, practices, and norms”.

The first argument in favour of managing knowledge in Software Engineering is that Software Engineering is a human and knowledge intensive activity (Birk, Surmann, & K. D

⁵ Definition by IEEE (<http://www.ieee.org/>)

Althoff, 1999). Software development is also a process that involves a great deal of decision making with several possible alternatives to follow. Thus, it is simply insufficient to rely solely on personal knowledge and experience to tackle decision making in Software Engineering. It is considered more a group activity where individuals need to communicate and coordinate, sharing individual knowledge and leveraging it at a project and organisation level. This is exactly what Knowledge Management proposes shifting the focus to collective creativity, exploiting the emerging behavioural idea – “none of us is as smart as all of us” (Bennis & Ward Biederman, 1998).

In software engineering, there has been much discussion about how to manage knowledge, or foster “learning software organizations”. In this context, Feldmann and Althoff have defined a “learning software organization” as an organization that has to “create a culture that promotes continuous learning and fosters the exchange of experience” (Feldmann & K.-D. Althoff, 2001). Dybå places more emphasis on action in his definition: “A software organization that promotes improved actions through better knowledge and understanding” (Dybå, 2001).

Furthermore, reusing life cycle experience, processes and products for software development is often referred to as having an “Experience Factory” (Basili, Caldiera, & D. H. Rombach, 1994). In this framework, experience is collected from software development projects, and are packaged and stored in an experience base. By packing, it is meant generalising, tailoring, and formalising experience so that it is easy to reuse.

In 1999, the first workshop on “learning software organizations” was organized in conjunction with the SEKE conference⁶. This workshop has been one of the main arenas for empirical studies as well as technological development related to knowledge management in software engineering.

The May 2002 issue of IEEE Software (Lindvall & Rus, 2002) was devoted to knowledge management in software engineering, giving several examples of knowledge management applications in software companies. In 2003, the book “Managing Software Engineering Knowledge” (Doran, 2004) was published, focusing on a range of topics, from

⁶ <http://www.ksi.edu/seke/seke10.html>

identifying why knowledge management is important in software engineering (Lindvall & Rus, 2003), to supporting structures for knowledge management applications in software engineering, to offering practical guidelines for managing knowledge.

However, Edwards notes in an overview chapter in the book on Managing Software Engineering Knowledge (Edwards, 2003) that knowledge management in software engineering is somewhat distanced from mainstream knowledge management.

In addition, a number of overviews of work on knowledge management in software engineering have previously been published. Rus et al. (Rus et al., 2001) present an overview of knowledge management in software engineering. The review focuses on motivations for knowledge management, approaches to knowledge management, and factors that are important when implementing knowledge management strategies in software companies. Lindvall et al. (Lindvall, Rus, Jammalamadaka, & Thakker, 2001) describe types of software tools that are relevant for knowledge management, including tools for managing documents and content, tools for managing competence, and tools for collaboration. Dingsøyr and Conradi (Dingsøyr & Conradi, 2002) surveyed the literature for studies of knowledge management initiatives in software engineering. They found eight reports on lessons learned, which are formulated with respect to what actions companies took, what the effects of the actions were, what benefits are reported, and what kinds of strategy for managing knowledge were used.

The following section presents the needs related to knowledge management in the software engineering domain, namely the motivation for KM in SE.

2.2 Motivation for KM in SE

The nature of Software Engineering is rather complex since it involves many people working in different phases and activities. It is a field where constant technology changes take place rendering the work of the involved people extremely dynamic, in the sense of discovery and solution of new problems on a daily basis. The knowledge in Software Engineering is diverse and its proportions immense and continuously expanding. Organisations face the problems of keeping track of what this knowledge is, where it is, and who has it. Thus, a structured way of managing the knowledge and treating the knowledge and its owners as valuable assets could help organisations leverage the knowledge they possess. The most important needs that drive the use of KM in SE are the following:

Capturing and sharing process and product knowledge

Software product and process differ from each other in terms of goals and contexts. It is simply unrealistic to assume that the same software development approach is applicable for all projects or products (Lindvall & Rus, 2000). According to (Basili & H. D. Rombach, 2002) this is one of the reasons that make the software discipline inherently experimental and, thus, experience is constantly enhanced with each development project. *“Knowledge emerges in work practices, often being defined by the first project to address the issues involved”* (Henninger, 1997). The ideal way to go forward would be to apply past, accumulated experience to future projects in order to avoid mistakes and leverage successes and that software development teams benefit from existing experience. Unfortunately, this is not always the case since, often, past work practices are not captured (Henninger, 1997) resulting in development teams reinventing the wheel and repeating the same mistakes that have been resolved in the past (Basili, Lindvall, & Costa, 2001), (Brössler, 1999). The aforementioned problems are also intertwined with the problem of transferring knowledge to new members in the organisation. Knowledge Management is ideal in comprising the means to address the issues of capturing and sharing knowledge in the Software Engineering domain.

Acquiring knowledge about the application domain

It is indispensable that software development teams possess adequate knowledge with respect to the domain for which software is being developed. It is rather frequent that a new domain requires learning a specific technique or a new programming language or application of a new kind of project management technique, resulting in longer times with respect to acquisition of the required experience and skills (Brössler, 1999). The ways in which domain knowledge, that no one in the organisation possesses, can be acquired is either by training or by hiring knowledgeable employees. Knowledge Management can, however, play an important role in assisting the organisation of knowledge acquisition and the identification of expertise as well as capturing, packaging and sharing knowledge that already exists in the organisation.

Acquiring knowledge about new technologies

The pace of change in Software Engineering technologies is fast resulting in difficulties in keeping up with the latest changes. On one hand, the emergence of new

technologies makes software more powerful, but on the other every emerging technology cannot be mastered overnight. It is evident that accurately estimating the cost of a project is a very difficult or even, sometimes, not feasible task when the technologies to be employed are new, not adequately tested and may even change during the project's lifecycle. The "learning by doing" approach is where software engineers resort to when dealing with new technologies for which little or no knowledge exists in the heads of the development team(s) (Brössler, 1999). This often results in serious delays of projects. Knowledge Management's role in this specific case is to foster a knowledge sharing culture within the organisation in order to help facilitate sharing of knowledge related to new technologies. Knowledge sharing can be promoted via communities of practice and interests, thus speeding up the learning curve.

Knowing who knows what

"It is more important for Software Engineering organisations to exploit and manage their intangible assets in contrast to their physical assets" (Tiwana, 2000). The intangible assets comprise in this case the tacit knowledge that resides in the heads of the members of a Software Engineering organisation. Therefore, it is imperative that the organisation invests in management of those intangible assets, i.e. possessing knowledge about who knows what. This is part of competence management and is the right direction towards reducing the time required to seek and retrieve experts in a specific field. Experts can help not only in resolving a potential problem but also (or instead) in pointing to the right piece of information that is pertinent to the respective problem. The issue of time invested in seeking the right information is of exceptional importance for the unhindered execution of a project. This is corroborated by the results of a study reporting that people in software organisations spent 40% of their time in searching for and accessing different types of information related to their projects (Henninger, 1997). *"Software organisations are heavily dependent on tacit knowledge, which is very mobile"* (Tiwana, 2000). In order to avoid or, at least, mitigate the risk of creation of severe knowledge gaps in cases of people suddenly leaving the organisation, it is imperative to know what knowledge they possess (Basili et al., 2001). Thus, retaining, if possible, the people who possess critical knowledge about processes and practices, can be of outmost importance, even for the viability of

the organisation. Knowledge Management comes into play by assisting in building structures and frameworks for capturing key information that can help retain some knowledge when employees leave.

Distance collaboration

As far as large scale software development is concerned, this is inherently a group activity. The division of work into phases, almost always, presupposes the assignment of different phases to different groups and their involvement either at the same point in time or at a later one. These groups can also be based in different geographic locations and it is common that group members live and work in different time zones. Outsourcing of subsystems to subcontractors also results in geographically co-located teams that need to communicate, collaborate, and coordinate independently of time and place. Knowledge Management acknowledges the need to capture, organise and store knowledge, as well as the necessity of knowledge transfer. Therefore:

- Communication in Software Engineering is often related to the transfer of knowledge.
- Collaboration is related to mutual sharing of knowledge.
- Coordination that is independent of time and space is facilitated if the work artefacts and their status are stored and made part of a knowledge repository.

2.3 KM Approaches & Tools in SE

2.3.1 *KM Approaches in SE*

2.3.1.1 Experience Factory

As far as software development is concerned, the most common way of learning is during projects. Consequently, the need for capturing that kind of knowledge arises in order to assist organisational learning. The knowledge from all projects ought to be documented, collected and organised into a structured repository aiming at supporting posterior decision making with respect to future projects.

A concept that supports this idea is the Quality Improvement Paradigm (QIP) (McGarry, Pajerski, Page, Waligora, & Basili, 1994). Figure 2.1 illustrates the way learning occurs at each project level by analyzing and drawing conclusions about the project's results,

both during execution and post mortem. These results are then analysed for a second time by means of abstracting from the specific problem domain, subsequently packaged in an organisational context and stored in an experience database. The resulting experience repository will support the planning process regarding future projects (i.e. selecting the processes, methods, techniques and tools that have been proved to be useful to the organisation in past projects).

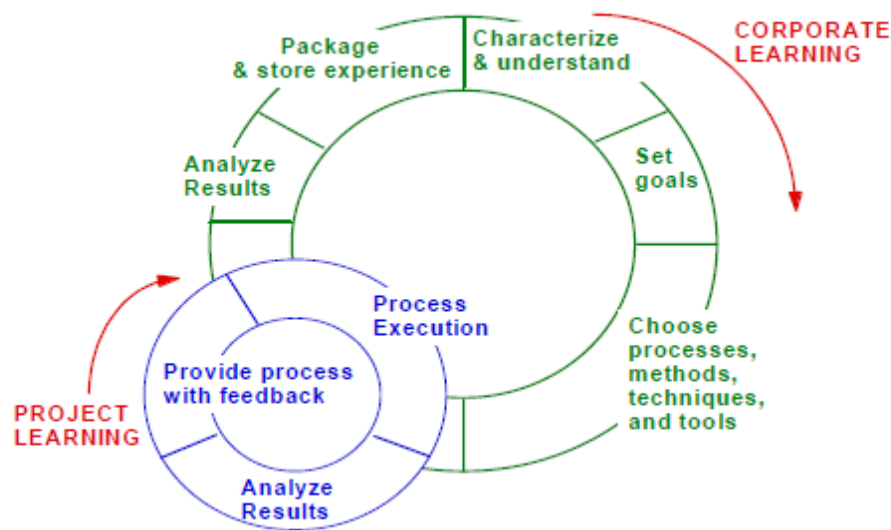


Figure 1.1: The Quality Improvement Paradigm (from (McGarry et al., 1994))

The Experience Factory (Basili et al., 1994) is a framework for the implementation of the Quality Improvement Paradigm. The approach has been successfully applied to software development at NASA for more than 25 years and recently at other organisations. The Experience Factory enables organisational learning and professes the need of existence of a separate support organisation. The role of the support organisation is to support the project organisation in order to manage and learn from its own experience. This goal is achieved by assisting the project organisation in observing and collecting data about itself, building models and drawing conclusions based on that data. The collected experience is subsequently stored in packages for further reuse, and most importantly, it is fed back to the project organisation (see Figure 2.2).

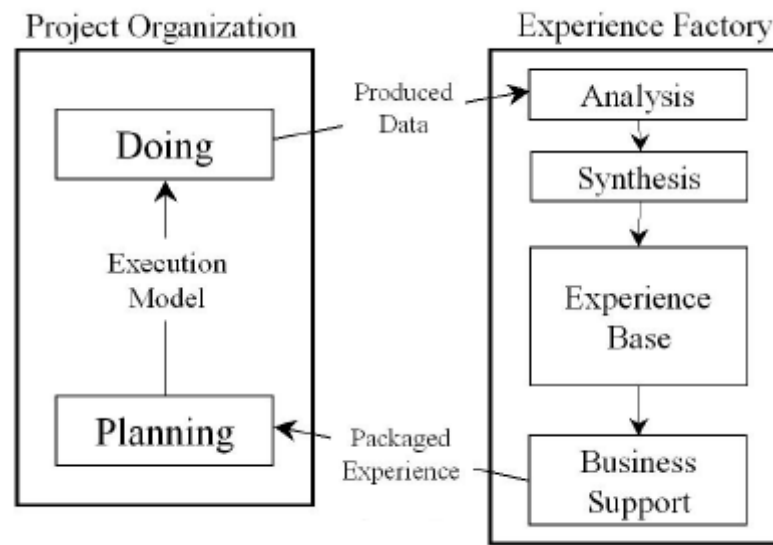


Figure 1.2: The Experience Factory Model

The Experience Factory approach takes into account the software discipline's experimental, evolutionary, and non repetitive characteristics. It has components that address capturing, storing, distributing, applying, and creating new experience. It also has components that address analysis and synthesis of knowledge (Basili et al., 2001). Organisations for which software development is not their core business process but whose processes are design-oriented rather than production-oriented (e.g. manufacturing), have recently been found capable of implementing successfully custom versions of the Experience Factory approach, thus having as a primary benefit their transformation into learning organisations. In summary, the Experience Factory is an example of an approach that is based upon the assumption that knowledge and experience can be made explicit so that they can be stored in knowledge and experience bases.

2.3.1.2 Knowledge Dust to Pearls approach

The Knowledge Dust to Pearls approach is influenced by the ideas of the Quality Improvement Paradigm (QIP) (see Section 2.3.1.1) which provides a model for process improvement in software organisations. QIP uses the notions of continuous improvement and iterations as the main vehicle for planning, executing, evaluating, and improving processes. The backbone of the Knowledge Dust to Pearls approach is the Experience Factory (see Section 2.3.1.1), which establishes a learning organisation. The Experience Factory is, however, a sophisticated approach that satisfies an organisation's long-term needs of sharing

experience. A complementary approach that satisfies the short-term needs of an organisation is the AnswerGarden approach (Ackerman & Malone, 1990).

The AnswerGarden approach lets employees store and organise questions and answers as they are received and answered by the organisation. By storing questions and corresponding answers in a common repository, the knowledge can easily be spread throughout the organisation.

The Knowledge Dust to Pearls approach (Basili et al., 2002) combines and makes use of benefits both from the AnswerGarden (which represents Knowledge Dust) and the Experience Factory (which represents Knowledge Pearls). It captures the knowledge dust that employees use and exchange on a daily basis and immediately, with minimal modifications, makes it available throughout the organisation. This process is accomplished by creating a system that supports peer-to-peer activities; i.e. the employees of the organisation help each other and fulfil the short-term return goals of a knowledge capturing and sharing approach.

The Knowledge Dust to Pearls approach adds a new and shorter feedback loop to the Experience Factory, which can be seen in Figure 2.3.

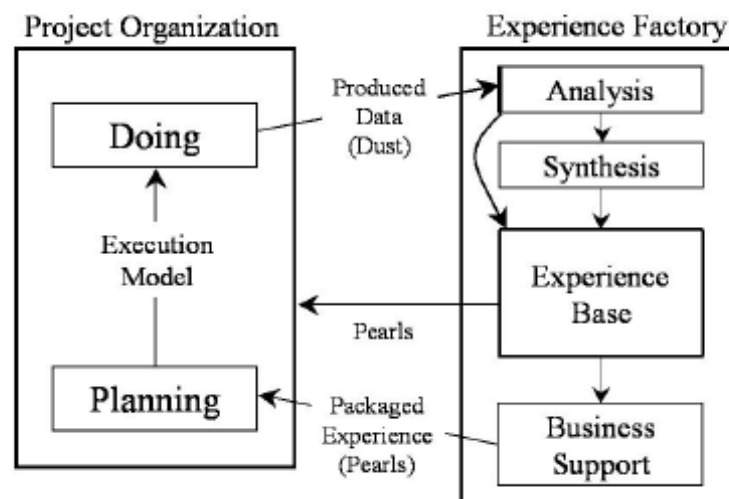


Figure 1.3: The Experience Factory Model with the new feedback loop of the Knowledge Dust to Pearls approach.

What the above figure illustrates is that in the Knowledge Dust to Pearls approach the data about an organisation, i.e. the dust, goes through a minimal analysis phase turning it into a mini-pearl. The mini-pearl entirely bypasses the synthesis phase and is stored in the experience base, thus it is made available to the project organisation almost immediately

after collection. In this way, the organisation receives benefits from the very beginning, as soon as the dust collection process is established.

2.3.1.3 Personal Software Process⁷

It is a reality that most software-development groups are significantly late and over budget regarding more than half of all software projects they undertake, and nearly a quarter of those projects are cancelled without ever being completed. Although developers recognise that unrealistic schedules, inadequate resources, and unstable requirements constitute often the main causes for such failures, only a minority of them knows how to solve these types of problems. Watts S. Humphrey and the Software Engineering Institute (SEI) profess that the Personal Software Process (PSP) is a clear and proven solution to the aforementioned problems. PSP (Humphrey, 2005) is a self-improvement process for software engineers comprising precise methods developed over many years by SEI. The PSP has successfully transformed work practices in a wide range of organisations and has already produced some promising results⁸.

PSP training focuses on the skills required by individual software engineers to improve their personal performance. Once learned and effectively applied, PSP-trained engineers are qualified to participate on a team using the Team Software Process (TSP) (see section 2.3.1.4).

PSP presents a disciplined process for software engineers and anyone else involved in software development. This process includes defect management, comprehensive planning, and precise project tracking and reporting. The goal for PSP is to provide to developers exactly what they need in order to deliver quality products on predictable schedules.

The Personal Software Process can be applied to many parts of the software development process, including:

- small-program development
- requirement definition

⁷ <http://www.sei.cmu.edu/tsp/psp.html>

⁸ For more information: <http://www.sei.cmu.edu/tsp/results/results.html>

- document writing
- systems tests
- systems maintenance
- enhancement of large software systems

2.3.1.4 Team Software Process⁹

- The Team Software Process (TSP) (Humphrey, 2000), along with the Personal Software Process, helps the high-performance engineer to
- ensure quality software products,
- create secure software products,
- improve process management in an organisation.

Engineering groups use the TSP to apply integrated team concepts to the development of software-intensive systems. The TSP provides team projects with explicit guidance on how to accomplish their objectives. As shown in Figure 2.4, the TSP guides teams through the four typical phases of a project. These projects may start or end on any phase, or they can run from beginning to end. Before each phase, the team goes through a complete launch or relaunch, where they plan and organise their work. Generally, once team members are PSP trained, a four-day launch workshop provides enough guidance for the team to complete a full project phase. Teams then need a two-day relaunch workshop to kick off the second and each subsequent phase. These launches are not training; they are part of the project.

To start a TSP project, the launch process script leads teams through the following steps (Humphrey, 1999):

- Review project objectives with management.
- Establish team roles.
- Agree on and document the team's goals.
- Produce an overall development strategy.

- Define the team's development process.
- Plan for the needed support facilities.
- Make a development plan for the entire project.
- Make a quality plan and set quality targets.
- Make detailed plans for each engineer for the next phase.
- Merge the individual plans into a team plan.
- Rebalance team workload to achieve a minimum overall schedule.
- Assess project risks and assign tracking responsibility for each key risk.
- Hold a launch post mortem.

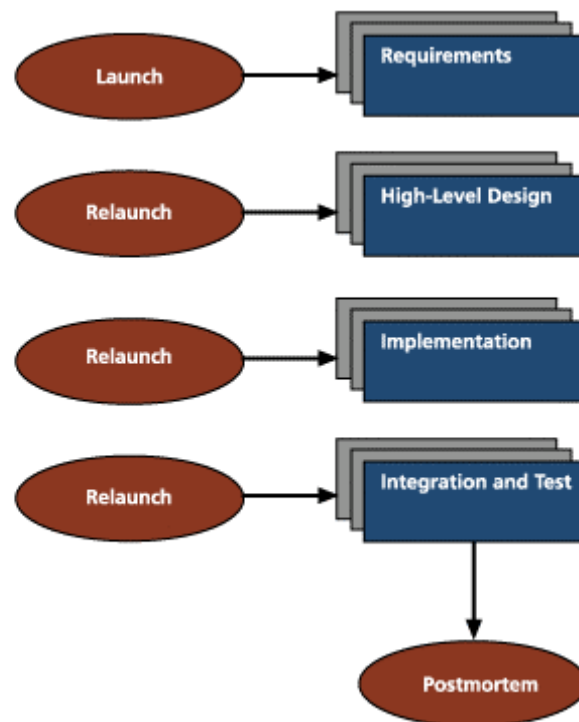


Figure 1.4: The TSP process

⁹ <http://www.sei.cmu.edu/tsp/tsp.html>

In the final launch step, the team reviews its plans and the project's key risks with management. Once the project starts, the team conducts weekly team meetings and periodically reports its status to management and to the customer.

After the launch, the TSP provides a defined process framework for managing, tracking and reporting the team's progress. Using TSP, an organisation can build self-directed teams that plan and track their work, establish goals, and own their processes and plans. TSP can provide assistance to organisations towards establishing a mature and disciplined engineering practice that produces secure, reliable software.

2.3.1.5 Process-based Knowledge Management support for Software Engineering

The primary goal of Process-Oriented Knowledge Management (POKM) is to establish, run and maintain an organisational environment that provides process participants with the information needed to successfully perform their activities as defined in the process model.

Holz presents in (Holz, 2003a) a detailed life-cycle model for POKM that is specific to software development processes. This life-cycle model for Software Engineering Process-Oriented KM (SE-POKM) is integrated into the life-cycle model performed by the organisation's Process Group and becomes an essential part of a continuous organisational learning process.

The SE-POKM model encompasses the following:

An explicit representation of dynamic (i.e. situation-specific) information needs that typically arise for process participants during software development activities; this representation also covers potential ways to satisfy those information needs.

A specification of the information need retrieval during process enactment. Depending on a characterization of their current situation (i.e. current activities, individual preferences and skills etc.), process participants are provided with modelled information needs that are expected to arise for them during their activities; in particular, corresponding selected information items are retrieved for each of these information needs, which are assumed to satisfy the information needs in required detail.

A guideline for the experience packaging phase based on feedback from process participants; during this phase, the initial model of relevant and useful information needs is updated to better reflect the participants' actual information needs.

In order to automate the retrieval of information needs and corresponding information items, Holz also presents the Process-oriented Information resource Management Environment (PRIME). PRIME provides a technical infrastructure for knowledge distribution and feedback communication and is designed to be coupled with the organisation's Process-Centred Software Engineering Environment.

2.3.2 *KM Tools in SE*

Software engineering involves a great variety of knowledge-intensive tasks: e.g. analysing user requirements for new software systems, identifying and applying best software development practices, collecting experience about project planning and risk management, etc. (Birk et al., 1999). In this section, an overview of representative KM tools for assisting Software Engineering tasks is given categorised by a classification adapted by Rus et al. (Rus et al., 2001).

2.3.2.1 Document Management Tools

The results of all Software Engineering tasks are documents (even source code and executable programs can be regarded as documents). Because Software Engineering is so dominated by the documents, Rus et al. in (Rus et al., 2001) argue that the foundation for a Knowledge Management system in Software Engineering is a document management system (DMS). A DMS tracks and stores electronic documents and/or images of paper documents.

A very popular and powerful DMS is "Hyperwave IS/6". According to NewHyperG AG (Ag, 2008) it provides, among others, document management with version control, support for multiple file formats, archiving with configuration management, search in arbitrary indexed meta data, user and group management with LDAP and Active Directory, finding similar documents and search in external sources (e.g.: of Web spiders, databases, file server).

Although Document Management systems support Software Engineering tasks, it is interesting to investigate how these technologies apply to Software Engineering documents. The Software Concordance for example, is a prototype integrated development environment (IDE) that uses a tree-based document representation for software documents, an integration between hypermedia and program analysis services, and inline multimedia documentation in program source code to improve software document management (Nguyen & Munson, 2003). Another example is srcML (SouRCe Markup Language). srcML presumes a

document view of source code where information about the syntactic structure is layered over the original source code document. The resultant multi-layered document has a base layer of all the original text (and formatting). The second layer is the syntactic information, derived from the grammar of the programming language, and is encoded in XML, thus enabling tasks such as analysis and transformation of the source code (Collard, Maletic, & Marcus, 2002).

2.3.2.2 Competence Management Tools

A definition for competence management, resultant from a recent extensive review of the state-of-the-art (Harzallah, Berio, & Vernadat, 2005) is the following: competence management concerns the way in which competencies in a corporation (of a group or individuals of the corporation) are organised and controlled. Competency (also called skill whenever related to humans) is a way to put in practice some knowledge, know-how, and also attitudes within a specific context. Competence management therefore has the prime objective to well define and continuously maintain the set of competencies according to the objectives of the corporation. A quite extensive survey of competence management systems and approaches has been conducted by Draganidis and Mentzas in (Draganidis & G. Mentzas, 2006).

Typical examples of competence management tools are Skillscape Competence Manager (*Skillscape Competence Manager*, n.d.) and SkillSoft Skillview (*SkillSoft Skillview*, n.d.). Table 2.1 summarises their most important features.

Skillscape Competence Manager	SkillSoft Skillview
Personal Skill Gap Analysis	Single assessments
Personal Development Planning	Gap analysis
Performance Planning	Individual and group reporting capabilities
Career Management	Skill-based searches
Organisational Skill Health Analysis	Multi-rater (360 degree) employee assessment
Predict Training Demand	Recruiting elements
Team Building and Internal Resourcing	

Skillscape Competence Manager	SkillSoft Skillview
Succession Planning	Total control over skill library, jobs and job profiles Skill weighting capabilities Employee resume posting and searching Employee free form posting and searching Employee scheduling

Table 1.1: Summary of typical competence management tools' features

Most of the aforementioned features are self explanatory. For detailed descriptions the reader is referred to (*Skillscape Competence Manager*, n.d.) and (*SkillSoft Skillview*, n.d.).

2.3.2.3 Intelligent Requirements Assistants

Developers of complex software systems are challenged to demonstrate how a software system satisfies a set of customer requirements expressed as operational scenarios (Sutcliffe, Chang, & Neville, 2003). Scenarios are a key approach to eliciting and validating requirements. They provide concrete examples of system usage that can help in stimulating critical inquiry into systems requirements. However requirements that were elicited based on scenarios are validated with much effort and are often error prone.

Authors in (Sutcliffe et al., 2003) propose an “Evolutionary Requirements Analysis (ERA)” tool which applies evolutionary computing techniques to automatically select optimal combinations of human and machine agents in a system model to match non-functional requirements. The tool assesses the reliability, performance times and cost of different system models by executing many model variants, as evolving forms, with scenarios and different combinations of environmental variables. Better performing models are selected, to converge on an optimal solution.

Authors in (Czuchry Jr & Harris, 2002) present a knowledge-based requirements assistant (KBRA) that is a component of the knowledge-based software assistant (KBSA). The idea behind KBSA is to create a knowledge-based life-cycle paradigm spanning software development from requirements to code and to formalise software practice so that computers can be used as active reasoning agents in developing software. They identify

knowledge-representation issues associated with requirements acquisition and analysis, and note the three realms in which mechanisms operate to resolve knowledge issues: presentations, structured text, and evolving system description. They describe artificial intelligence techniques used to provide consistent reasoning processes for the intelligent assistant: inheritance of properties from generic object types, automatic classification based on discriminators indicating how to specialise instances, and constraint propagation for processing ramifications of requirements decisions and for supporting retraction when people change their minds.

2.3.2.4 Knowledge Based Program Designers

Program designers are tools that present their user with algorithms that match current requirements and specifications. These tools help transfer knowledge from the requirements phase to the design phase. Algorithms are either generated with a machine learning approach or retrieved from the knowledge base by case-based reasoning (Rus et al., 2001).

Tools like CAESAR (Fouque & Matwin, 1992) use case-based reasoning for retrieving algorithms from the knowledge base by matching current requirements and specifications. This great help in code design comes with a shortcoming. CAESAR tries to match every possible case with the current requirements. Although the result might be close matches, it becomes cumbersome for the user to deal with so many examples. While CAESAR presents the user with all possible cases that can be reused, RT-Syn (T. E. Smith & Setliff, 1992) tries to select one single possible algorithm. It looks into the algorithm database, choosing a likely candidate algorithm, and then makes design decisions based on the given constraints in requirements and specifications (Vrain, 1992). Designer Assistant (Terveen, Selfridge, & Long, 1995) addresses three kinds of knowledge in its knowledge base:

- Expert knowledge of design with which most designers are not familiar.
- Impact knowledge (i.e. how characteristics of a design affect another area of software).
- Fault prevention knowledge (i.e. how characteristics of design could lead to a fault).

Tools or design environments like the Designer Assistant not only help transfer design knowledge which exists as folklore in organisations, but they also help organisations build their knowledge bases and increase the expertise level of designers and developers.

2.3.2.5 Knowledge Based Code Generators and Recommendation Systems

When coding to a framework, developers often become stuck, unsure of which class to subclass, which objects to instantiate and which methods to call. Example code can help developers make progress on their task (Holmes & Murphy, 2005). Code generators rely on a previously acquired knowledge base (built from examples, or existing applications) that may or may not evolve, and strive to generate executable code. There are two kinds of code generation tools, using push and pull technology respectively (Rus et al., 2001). Tools using push technology will automatically let the developer know about every possible opportunity of reusing previous acquired knowledge and components. On the other hand, in tools using pull technology, the user has to actually search for knowledge and reusable components.

An exemplary tool using push technology is CodeBroker (Ye & G. Fischer, 2002). CodeBroker queries a repository automatically after each comment or method signature written by a developer. The queries made to the repository are based on these comments and method signatures. To retrieve matches, a developer must write comments that explain the functionality of the software in terms similar to that of the repository code.

In CodeFinder system (Henninger, 1991), the developer formulates a simple text query (using pull technology), executes the query and is then presented with a list of terms in the repository that are similar to those in the query. Depending on the terms and options selected by the developer, a different set of restrictions is presented to help narrow the search space to a specific class of examples of interest.

The Strathcona tool (Holmes & Murphy, 2005) is a plug-in for the Eclipse IDE which extracts the structural context of the code on which a developer is working when the developer requests examples. The server portion of the tool houses the example repository and selects examples to be returned using a set of structural matching heuristics. Authors in (Holmes & Murphy, 2005) consider that an appropriate example is a subset of one of the applications stored in the repository, consisting of a set of relevant classes and relationships. The developer is presented with a structural overview of each example using a compact visual representation. The developer can access a rationale for why the example has been returned, as well as the source for the example.

There is a great variety of tools with similar functionalities such as KIDS (R. D. Smith, 1990), SINAPSE (Kant, 1992), CodeWeb (Amir, 2001), Component Rank (Inoue et al.,

2003), Reuse View Matcher (Rosson & Carroll, 1996), Automatic Method Completion (Hill & Rideout, 2004), Hipicat (Čubranić & Murphy, 2003) and RASCAL (Mccarey, Cinnéide, & Kushmerick, 2005) to name just a few.

2.3.2.6 Smart Code Analysis Tools

Code analysis tools can offer substantial help during software testing and quality assurance activities. Analyzing code requires expert knowledge about the quality of the written code and good programming style. This knowledge is captured in a knowledge base and integrated with tools like OGUST (Vrain, 1992) that analyze the code for quality and good programming style.

Style checkers such as PMD¹⁰ turn up poorly named variables, duplicated code, and many other deviations from coding conventions. Bug detectors like FindBugs¹¹ spot common reliability problems such as dereferencing null pointers, infinite recursive loops, and broken idioms that do not achieve what the author intended.

A system for detecting redundancies in source code is R²D² (Leitao, 2004). R²D² identifies redundant code fragments on large software systems. For each pair of code fragments, R²D² uses a combination of techniques ranging from syntax-based analysis to semantics-based analysis, that detect positive and negative evidences regarding the redundancy of the analyzed code fragments. These evidences are combined according to a well-defined model and fragments sufficiently redundant are reported to the user. R²D² explores several techniques and heuristics to operate within reasonable time and space boundaries and is designed to be extensible.

Another approach is program slicing, which was first introduced by Weiser in 1979 (Weiser, 1979). A definition in (Xu, Qian, X. Zhang, Wu, & Chen, 2005) for program slicing is the following: “*a decomposition technique that extracts from program statements relevant to a particular computation*”. A program slice consists of the parts of a program that potentially affect the values computed at some point of interest referred to as a slicing criterion. Typically, a slicing criterion consists of a pair $\langle p, V \rangle$, where p is a program point

¹⁰ <http://pmd.sourceforge.net>

¹¹ <http://findbugs.sourceforge.net>

and V is a subset of program variables. The parts of a program that have a direct or indirect effect on the values computed at a slicing criterion C are called the program slice with respect to criterion C . The task of computing program slices is called program slicing.

Slicing was first developed to facilitate debugging, but it is then found helpful in many aspects of the software development life cycle, including program debugging, software testing, software measurement, program comprehension, software maintenance, program parallelisation and so on.

2.3.2.7 Software Maintenance Tools

Oman in (Hanebutte & Oman, 2005) defines software maintenance as “the Software Engineering activities pertaining to non-essential enhancements in software, adapting the software due to changing environmental requirements, and the removal or mitigation of faults”. Software maintenance efforts and research focus on prevention and removal of software faults throughout the Software Engineering life cycle, including issues that arise from modification of software after delivery.

Authors in (O’Keeffe & Cinnéide, 2006) describe a novel approach in providing automated re-factoring support for software maintenance: the formulation of the task as a search problem in the space of alternative designs. Such a search is guided by a quality evaluation function that must accurately reflect re-factoring goals. Based on this approach they have built a search-based software maintenance tool. CODE-Imp applies automated re-factorings to a program in order to move through the space of alternative designs and search for those of highest quality, based on a given design quality function. The effectiveness of the search can be measured in terms of the change in quality function, but the effectiveness of the approach itself can only be judged in terms of the actual changes made to the program and to what extent it is more maintainable than the original. For this reason, choice of design quality function is a key facet of this work.

Another approach proposed by Mens et al. in (Mens, Poll, & González, 2003) is a lightweight abstraction of intentional source-code views as a means of making the conceptual structure of existing software systems (which is often implicit or non-existing in the source code) more explicit. An intentional source-code view (as defined in (Mens et al., 2003)) is a set of related program entities (such as classes, instance variables, methods, method statements) that is specified by one or more alternative descriptions (one of which is

the ‘default’ description). Each alternative description is an executable specification of the contained elements in the view. Such a description reflects the commonalities of the contained elements in the view, and as such, codifies a certain intention that is common to all these elements. In addition, all alternative descriptions of a given view are ‘extensionally consistent’, in other words, after computation they should yield the same set of elements. The computational medium in which the intentional views are described is a declarative metaprogramming language. The practical usefulness of intentional views to aid software maintenance is described in (Mens et al., 2003) in the context of two case studies that Mens et al. conducted.

2.3.2.8 Wikis

According to (Désilets, Paquet, & Vinson, 2005), Wikis are simple to use, asynchronous, web-based collaborative hypertext authoring systems. The general consensus is that a Wiki is a collective website where a large number of participants are allowed to modify any page or create a new page using their web browser.

Wikis are also able to provide assistance in the Software Engineering process. Chau and Maurer in (T. Chau & F. Maurer, 2006) propose a wiki-based experience repository (MASE) that utilises both informal and formal knowledge representations. The need of knowledge sharing tools to incorporate not only codification-oriented repository technologies but also those that facilitate communication and collaboration among people, and to support not only structured but also unstructured knowledge representation is obvious. An informal knowledge authoring tool such as MASE is used for sharing content for problem understanding, instrumental, projective, social, expertise location, and content navigation purposes. Semantic media Wiki¹² is using semantic technologies to represent knowledge and can therefore be used to assist MediaWiki (non-semantic)¹³.

A more detailed review of wikis and especially semantic wikis is presented in Section 3.3.

2.3.2.9 Collaborative Tools

¹² http://ontoworld.org/wiki/Semantic_MediaWiki

¹³ <http://www.mediawiki.org/wiki/MediaWiki>

Software engineering is a predominantly collaborative activity. Typically multiple teams of people develop and maintain successive versions of a range of products in parallel. Surprisingly, tools to support synchronous or real-time Collaborative Software Engineering (CSE) are still restricted to minor tasks for specific Software Engineering purposes (Cook & Churcher, 2006).

Cook and Churcher (Cook & Churcher, 2006) propose the CAISE architecture to assist in the construction of new tools to support the real-time development of a collaborative software project. The CAISE architecture, allows isolated programmers to work collaboratively without sacrificing communication. CAISE-based tools achieve this by keeping all programmers within a group synchronised in real-time, at the same time providing customisable user awareness and project state information to the individual tools. The CAISE architecture provides an infrastructure with the potential to support the entire Software Engineering process. CAISE tools can be constructed that provide more than just the shared editing of basic software artefacts. Collaborative compilation, testing and debugging of software projects are also possible to implement using the services of CAISE. Comprehensive inter-developer communication facilities can also be constructed.

The Augur system (Froehlich & Dourish, 2004) can be viewed as an example technique to look at software development as a system of evolution. The Augur system simultaneously visualises the structure of a software system (i.e. artefacts) and the structure of the development process carried out by developers (i.e. developers and the community). Augur visualises the result of call graph analysis and networks of contributors to a project, relating those who worked together on a single module. By looking at how developers worked together on what parts of a software system, a user of Augur could tell how relationships between artefacts (software system module structures) and developers have changed over time, including phenomena such as types of projects, the different roles undertaken by different developers, how such roles shift between core and periphery, how authorship changes, and what patterns of stability and changes are observable. Augur currently supports ways to view the structural changes from an objective standpoint, providing ego-centric individual viewpoints, for instance, from a particular developer's point of view.

Another approach to CSE is using hybrid representations of online activities. Authors in (Medynskiy, Ducheneaut, & Farahat, 2006) have developed a system to observe and

analyze collaborative activities in online groups merging three data sources: not only communication patterns (i.e. email traffic) but also topical and material relationships. Each component in this hybrid network allows easy access to the raw data, so that analysts can examine the qualitative information behind the structure of the network. They show how the simultaneous visualisation of heterogeneous data reveals collaboration patterns that would not have been visible using social networks exclusively.

2.4 Knowledge Management Schools in SE

It should be noted that this section is adopted from a paper (Bjornson & Dingsoyr, 2008) written by Bjørnson and Dingsøyr.

Earl (Earl, 2001) has classified work in knowledge management into schools (see table 2.2). The schools are broadly categorized as “technocratic”, “economic” and “behavioural”. The technocratic schools are: (1) the systems school, which focuses on technology for knowledge sharing, using knowledge repositories; (2) the cartographic school, which focuses on knowledge maps and creating knowledge directories; and (3) the engineering school, which focuses on processes and knowledge flows in organizations.

The economic school focuses on how knowledge assets relates to income in organizations.

The behavioural school consists of three sub-schools: (1) the organizational school, which focuses on networks for sharing knowledge; (2) the spatial school, which focuses on how office space can be designed to promote knowledge sharing; and (3) the strategic school, which focuses on how knowledge can be seen as the essence of a company’s strategy.

	Technocratic			Economic commercial	Behavioural		
	Systems	Cartographic	Engineering		Organizational	Spatial	Strategic
Focus	Technology	Maps	Processes	Income	Networks	Space	Mindset
Aim	Knowledge bases	Knowledge directories	Knowledge flows	Knowledge assets	Knowledge pooling	Knowledge exchange	Knowledge capabilities
Unit	Domain	Enterprise	Activity	Know-how	Communities	Place	Business

Table 1.2: Earl’s schools of knowledge management – adopted from (Bjornson & Dingsoyr, 2008)

Bjørnson and Dingsøyr (Bjornson & Dingsoyr, 2008) discovered 29 empirical studies and 39 reports of lessons learned. Within Earl's framework, they found a heavy concentration on the technocratic schools and a fair mention of the behavioural school. They did not find any papers relating to the economic school. Within the technocratic schools, systems and engineering stand out as areas that have received much attention. Within the behavioural schools, organizational and strategic have received the most attention.

Looking at the papers by year of publication, they noticed an increasing interest in the area from 1999 onwards. They also noticed a shift from more papers on lessons learned to empirical papers from 2003 onwards.

2.4.1 *Technocratic schools*

The technocratic schools are based on information or management technologies, which largely support and, to different degrees, condition employees in their everyday tasks. They identified a total of 19 empirical studies and 29 papers on lessons learned in this category. The main focus is on the engineering and systems schools.

In software engineering processes, the systems and engineering schools support sharing of explicit knowledge, which is important in traditional software development. Both of these schools require a technical infrastructure in order to facilitate knowledge sharing. However, a finding both from studies in other fields of the systems school (Kankanhalli, Tan, & Wei, 2005) and studies of specific engineering activities, electronic process guides, is that it is difficult to get such technology in actual use (Dingsøyr & Moe, 2008). However, many companies have invested in such infrastructure, and this indicates that a better understanding of the factors that lead to effective knowledge sharing within these two schools is needed.

2.4.1.1 Systems

As defined by Earl, the systems school is built on the underlying principle that knowledge should be codified in knowledge bases. This is what Hansen et al. refer to as the "codification strategy", and what Nonaka and Takeuchi refer to as externalisation.

This school is the longest established school of knowledge management. In total, Bjørnson and Dingsøyr classified six papers as empirical in this school, and 20 as lessons

learned. The empirical papers in this category can broadly be defined as either dealing with the development or use of knowledge repositories.

In (Chewar & McCrickard, 2005), Chewar and McCrickard present their conclusions from three case studies investigating the use of their knowledge repository. On the basis of their case studies, they present general guidelines and tradeoffs for developing a knowledge repository. In (Bjornson & Staalhane, 2005), Bjørnson and Stålhane follow a small consulting company that wanted to introduce an experience repository. On the basis of interviews with the employees, they draw conclusions about attitudes towards the new experience repository, and the content and functionality preferred by the employees. Barros et al. (Barros, Werner, & Travassos, 2004) investigate how risk archetypes and scenario models can be used to codify reusable knowledge about project management.

Concerning the actual usage of experience repositories or knowledge bases, Dingsøyr and Røyrvik (Dingsøyr & Røyrvik, 2003) investigate the practices in a medium-sized software consulting company where knowledge repositories are used in concrete work situations. They found several distinct ways of using the tool and highlight the importance of informal organization and the social integration of the tool in daily work practices. A more formal approach to knowledge management tools is found in (Skuce, 1995), where Skuce describes experiences from applying a knowledge management tool in the design of a large commercial software system. Concerning longterm effects of experience repositories, Kurniawati and Jeffery (Kurniawati & Jeffery, 2004) followed the usage of a combined electronic process guide and experience repository in a small-to-medium-sized software development company for 21 weeks, starting a year after the tool was introduced. They conclude that tangible benefits can be realized quickly and that the tool remains useful with more benefits accruing over time.

2.4.1.2 Cartographic

The principal idea of the cartographic school is to make sure that knowledgeable people in an organization are accessible to each other for advice, consultation, or knowledge exchange. This is often achieved through knowledge directories, or so-called “yellow pages”, that can be searched for information as required.

Bjørnson and Dingsøyr (Bjornson & Dingsoyr, 2008) found only one empirical paper within this school and no papers on lessons learned. In (Dingsøyr, Djarra, & Royrvik, 2005), Dingsøyr et al. examine a skills management tool at a medium-sized consulting company. They identify four major usages of the tool and point out implications of their findings for future or other existing tools in this category

2.4.1.3 Engineering

The engineering school of knowledge management is a derivative or outgrowth of business process reengineering. Consequently it focuses on processes. According to the classification of Bjørnson and Dingsøyr, the largest amount of empirical papers came from this school. Two major categories can be identified. The first contains work done by researchers who investigate the entire software process with respect to knowledge management. The second contains work done by researchers who focus more on specific activities and how the process can be improved within this activity.

Baskerville and Pries-Heje (Baskerville & Pries-Heje, 1999) used knowledge management as the underlying theory to develop a set of key process areas to supplement the Capability Maturity Model (CMM) (Paulk, 1995) in a Small- and Medium-sized Enterprise (SME) software development company. Realizing that the CMM did not fit well with an SME company, they helped their case companies to develop new key process areas that focused on managing their knowledge capability. Arent et al. (Arent, Norbjerg, & Pedersen, 2000) address the challenge of creating organizational knowledge during software process improvement. They argue for the importance of creating organizational knowledge in Software Process Improvement (SPI) efforts and claim that its creation is a major factor for success. On the basis of an examination of several cases, they claim that both explicit and tacit knowledge are required, no matter what approach is pursued. Segal (Segal, 2001) investigates organizational learning in software process improvement. Using a case to initiate and implement a manual of best practice as a basis, she observed that the ideal and actual scenarios of use differed and identified possible reasons for the difference. In (Folkestad, Pilskog, & Tessem, 2004), Folkestad et al. studied the effect of using the rational unified process as a tool for organizational change. In this case, it was used to introduce development staff to a new technology and methodology. Folkestad et al. concluded that the iterative approach of the unified process had obvious effects on organizational and individual learning. The unified process also resulted in new patterns of communication and a new

division of labour being instituted, which had a significant effect on the company. Wangenheim et al. (Wangenheim, Weber, Hauck, & Trentin, 2006) report on their experiences of defining and implementing software processes. They confirm what others have experienced, that it is possible to define and implement software processes in the context of small companies in a beneficial and cost-effective way.

2.4.2 Behavioural schools

The behavioural aspects of knowledge management are covered in three schools in Earl's framework: the organizational, spatial, and strategic schools. Bjørnson and Dingsøyr found three empirical studies and two reports of lessons learned in the organizational school, no empirical study and one report of lessons learned in the spatial school, and three empirical studies and nine reports of lessons learned in the strategic school. They present the main concepts and findings from the organizational and strategic schools.

In software engineering, they believe that this school has the potential to deliver inexpensive solutions for companies, although as the studies in software engineering indicate, there is a debate on whether such initiatives are best left to grow by themselves or if the management should have an active involvement. For software engineering, it could be useful with studies that address this strategy in relation to specific challenges for software development, like challenges with new technology, process improvement or understanding customer needs. This school is relevant for organizations that run multidisciplinary projects, which they believe is the case for most software companies, whether they do agile or traditional development.

2.4.2.1 Organizational

The organizational school focuses on describing the use of organizational structures (networks) to share or pool knowledge. These structures are often referred to as "knowledge communities". Work on knowledge communities is related to work on communities of practice.

The role of networking as an approach to knowledge management has been investigated in three settings where software is developed. Grabher and Ibert (Grabher & Ibert, 2006) discuss what types of network exist in companies, where one case is a software

company based in Germany. Mathiassen and Vogelsang (Mathiassen & Vogelsang, 2005) discuss how to implement software methods in practice and use two concepts from knowledge management: networks and networking. The network perspective emphasizes the use of technology for sharing knowledge, while networking focuses on trust and collaboration among practitioners involved in software development. The authors stress that knowledge management is highly relevant to understand challenges when introducing new methods for software engineering, and that every company has to find a suitable balance between strategies. In the case company, the emphasis on networks and networking changed considerably during the project. Nørbjerg et al. (Nørbjerg, Elisberg, & Pries-Heje, 2006) discuss the advantages and limitations of knowledge networks. They base their discussion on an analysis of two networks related to software process improvement in a medium-sized software company in Europe. Their main finding is that building a network on existing informal networks gave the highest value to the organization.

2.4.2.2 Strategic

In the strategic school, knowledge management is seen as a dimension of competitive strategy. Skandia's views are a prime example (Sveiby, 1997). Developing conceptual models of the purpose and nature of intellectual capital has been a central issue.

One important issue in the literature on knowledge management has been to identify the factors that lead to the successful management of knowledge. Feher and Gabor (Fehér & Gábor, 2006) developed a model of the factors that support knowledge management. The model includes technological, organizational and human resource factors, and was developed on the basis of data on 72 software development organizations that are contained in the European database for the improvement of software processes.

Another issue of strategic importance is the processes that are in place to facilitate learning. Arent and Nørbjerg (Arent & Nørbjerg, 2000) analysed three industrial projects for the improvement of software processes, in order to identify the learning processes used. They found that both tacit and explicit knowledge were important for improving practice, and that improvement requires ongoing interaction between different learning processes.

Trittmann (Trittmann, 2001) distinguish between two types of strategy for managing knowledge: "mechanistic" and "organic". Organic knowledge management pertains to

activities that seek to foster innovation, while mechanistic knowledge management aims at using existing knowledge. A survey of 28 software companies in Germany supported the existence of two such strategies. This work parallels the works of Hansen et al. on codification and personalization as important strategies for managing knowledge in the field of management science.

2.4.3 Implications for research and practice

The systematic review of Bjørnson and Dingsøyr has implications both for researchers planning new studies of knowledge management initiatives in software companies, and for practitioners working in software companies who would like to design knowledge management initiatives to meet local needs.

2.4.3.1 Implications for research

Bjørnson and Dingsøyr have distinguished between two types of development which has implications for strategy for knowledge management, namely traditional and agile development. In their systematic review, they have seen that the knowledge management schools associated with traditional software development so far has received the most attention, namely the systems and engineering schools. They believe the schools that are relevant to agile software development should be given further attention in the future, as this trend seems to have much influence on industry practice today.

Another issue in deciding on priorities for research is the cost of implementing activities in the schools. In general, the schools which do not require codification and a technical infrastructure will be less expensive than the others. Therefore, they argue that in particular the organizational school should be further researched as this school is both relevant for agile and traditional software development, and is inexpensive. Also, the cartographic and spatial schools are good candidates for further research.

2.4.3.2 Implications for practice

The technocratic schools are closely related to traditional software development while the behavioural schools are more related to the agile approach to development.

Practitioners following a traditional approach can find some empirical papers and several lessons learned reports on how to build a knowledge repository. Even though all papers identified within the systems school are positive it is important to remember the

objections to following a pure codification strategy. There is potential bias in the number of positive reports from this school versus those who report negative results. Findings from the engineering school also support this view, where several papers underline the importance of not focusing exclusively on codification. An advantage of following the technocratic approach to knowledge management is that there is more material available within this “classical” school. A disadvantage is the cost of implementing strategies relying heavily on codification.

The most important finding from the behavioural schools with implications for practitioners developing in an agile environment would be that network building is more likely to be successful if they are built on already existing networks. Also, the need for diversity in both learning processes and strategies are stressed as important in order to improve practice. An advantage of the behavioural approach to knowledge management is the reduced cost compared to implementing the more application heavy solutions in the technocratic school. However, it has its disadvantage in the relatively few publications on this theme to learn from.

3 SOCIAL SEMANTIC DESKTOPS

The social semantic desktop paradigm (Groza et al., 2007) adopts the ideas of the Semantic Web paradigm, which offers a solution for the web. Formal ontologies capture both a shared conceptualization of desktop data and personal mental models. RDF (Resource Description Format)¹⁴ serves as a common data representation format. Web Services - applications on the web - can describe their capabilities and interfaces in a standardized way and thus become Semantic Web Services. On the desktop, applications (or rather: their interfaces) will therefore be modelled in a similar fashion. Together, these technologies provide a means to build the semantic bridges necessary for data exchange and application integration. The Social Semantic Desktop will transform the conventional desktop into a seamless, networked working environment, by loosening the borders between individual applications and the physical workspace of different users.

Decker (Stefan Decker, 2006) explains how the “social semantic desktop” is realized (Figure 3.1). Furthermore, he explains that semantics are used in order to make metadata and metadata items, ideas, entities, and topics all interconnected in a person’s space. Then, it is possible to share this information in social protocols; social in the sense of individual to individual; or organisation to organisation; communication exchange.

¹⁴ <http://www.w3.org/RDF/>

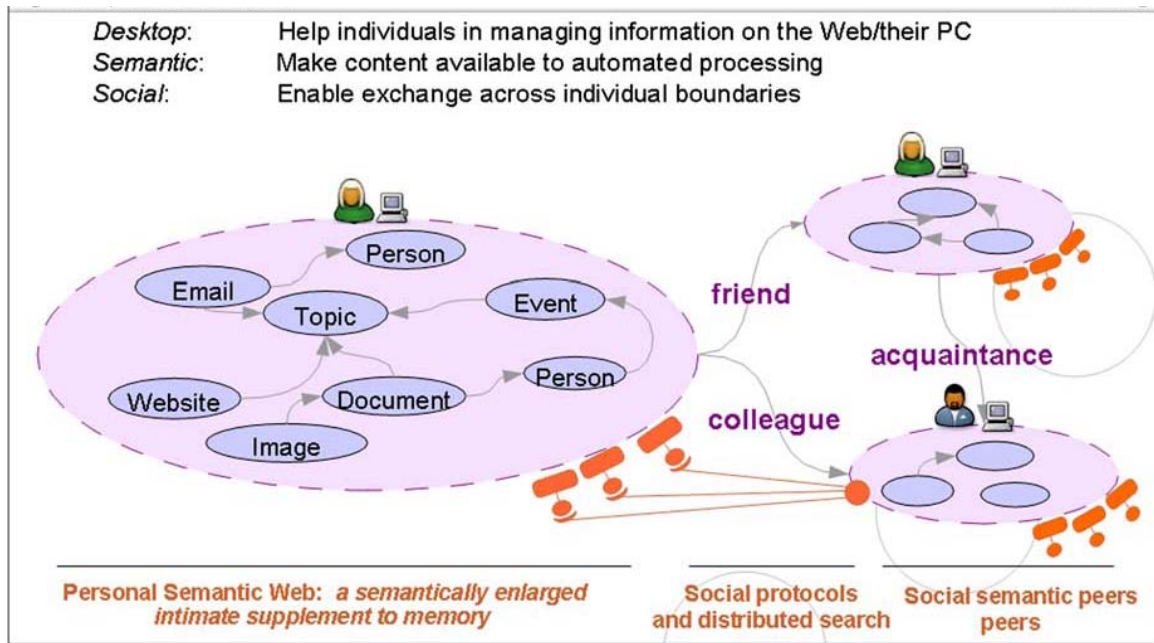


Figure 3.1: Realising the Social Semantic Desktop

Metadata are standardized in the past few years, because these standards assist in formulating and to exchanging data. The effort now is to develop a scalable distributed infrastructure. P2P-computing enables to interconnect without civilised infrastructures and to in-an-ad-hoc-way, communicate and exchange information (Stefan Decker, 2006).

Knowledge articulation can be assisted by desktop technology, but also from the web itself, for example from wikis. Also, Natural Language Processing (NLP) is assisting in dealing with unstructured sources to a certain extent, so with decreased data. Last, but not least, there is a shift from pure technical matter and how to deal with sending information for NLP, to sending information about people to a community, or to an individual (Stefan Decker, 2006).

Current desktop technology needs help in terms of metadata processing. Also from the web there are new entities dealing with metadata. Among these are folksonomies, wikis and blogs, which are already well known.

There is also a level of research and development integrating the web and the desktop technologies, which is called by Decker (2006) “social semantic information space”.

Specific examples that development is taking place would be semantic blogging, which is being explored; integrating desktop data into a blog. Also, wikis for personal information can be shared; which is a step further than what wikis are able to do now. These semantic

personal wikis and the integration of folksonomies and desktop data are a very active research area. Folksonomies are being used to tag items from the web. Once this is done, it enables reusing tagged information.

As already mentioned in chapter 1, the thesis approaches knowledge management in software development by using **social semantic desktop** technologies. In this chapter the state-of-the-art in semantic desktops and semantic wikis (which constitute together the social semantic desktop) is described. Furthermore, semantic wikis and desktops in software engineering are described.

3.1 Semantic Desktops

In this section a definition of semantic desktop is given, highlighting the growing need of knowledge workers for such systems. Then, a survey of the following semantic desktop solutions is presented, focusing on the functionality and architecture of each system: Chandler, DeepaMehta, Fenfire, Gnowsis, Haystack, Iris and Nepomuk. These systems are selected for the survey, since they are the most popular and provide a rich set of functionalities. In the last section the comparison based on the characteristics of these systems is outlined.

3.1.1 Introduction

Authors in (Sauermann, Bernardi, & Dengel, 2005) define a semantic desktop in the following way:

“A semantic desktop is a device in which an individual stores all her digital information like documents, multimedia and messages. These are interpreted as Semantic Web resources, each is identified by a Uniform Resource Identifier (URI) and all data is accessible and queryable as RDF graph. Resources from the web can be stored and authored content can be shared with others. Ontologies allow the user to express personal mental models and form the semantic glue interconnecting information and systems. Applications respect this and store, read and communicate via ontologies and Semantic Web protocols. The semantic desktop is an enlarged supplement to the user’s memory.”

Knowledge workers need to perform their daily job anytime and in anyplace. This originates from the fact that the Internet is always available and has a wide variety of available services. Thus, the trend is to shift knowledge work from desktop-based systems to

web-based systems. A problem here can be the overlap between web and desktop applications. Editing and creating information is usually done in desktop applications. Documents are downloaded or received by email, edited and then sent on to others or posted on the web again. This overlap is more and more replaced with a coherence (Sauermann, 2005). Office applications like Microsoft Office and Open Office can export their data in html. Web applications are used in office scenarios to realize organizational memories, search functions, collaboration environments. Software is either built to run on the web or it is web-enabled.

Sauermann in (Sauermann, 2005) argues that building information management systems would be much simpler if data on desktop computers could be treated like web resources. The state of the art in web architecture is the Semantic Web, *“an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation”* (Berners-Lee, Hendler, & Lassila, 2001). A translation of the Semantic Web to the topic of desktop information systems is the next step.

Data processed by knowledge workers come from an extensive number of existing applications. Emails, documents, contact and calendar information are such obvious data sources. These are the main resources that any personal information management system supports. The semantic desktop is targeted on integrating the various information systems. The challenge is to integrate data from SQL databases, Office Applications and other common office appliances - based on the Semantic Web standards.

The data entities (emails, documents, photos, videos, etc.) in a semantic desktop are all seen as Semantic Web resources. In this way, much less effort is required in order to integrate all this information. For this integration, there are three requirements:

All resources are identified with a URI - Uniform Resource Identifier.

All structured data is accessible through the RDF, which provides a common framework for expressing semantic information so it can be exchanged between applications without loss of meaning.

The represented RDF information has to comply with ontologies. The semantic meaning of the data is described in ontologies using RDF Schema or OWL, which allows application integration in a more effective way (Fensel, 2001).

3.1.2 Chandler

Chandler (Kapor, 2005) is a personal information manager aiming to assist in everyday information and communication tasks, such as managing email messages, appointments and contacts. Chandler users can share information with others and thus Chandler can be characterized as an Interpersonal Information Manager.

Chandler is intended to be an open source personal information manager for email, calendars, contacts, tasks, and general information management, as well as a platform for developing information management applications.

General Information Management

Chandler associates and interconnects information resources and, in the same time, collects related resources in a single space creating a context sensitive "view" of many types of data. Data in Chandler can be stored in repositories on the user's local machine, on others' machines and on shared resources such as servers.

Chandler lets the user keep track of several concurrent, ongoing activities. A primary goal of Chandler is the ability to gather relevant information from disparate sources. For example, an individual using Chandler can combine a variety of emails, contacts, documents and calendar events into an ad-hoc collection related to a specific project.

Another main goal of Chandler is to provide unified search over all of the user's information, both on her own PC and in other Chandler repositories across the network.

Power Email

Chandler gives the ability to manage large volumes of email efficiently and effectively. Many users use email as a form of very lightweight "to-do" list, leaving messages in a very visible place until they have been read, acted on, and/or replied to. Chandler facilitates the "task management" aspects of email by assisting users in organizing and prioritizing their messages.

Calendar

Chandler's peer-to-peer calendaring system enables any subgroup of Chandler users (e.g., a small business, a study group, etc.) to efficiently schedule meetings, browse the calendars of others, and see overlays of multiple calendars simultaneously.

Sharing and Collaboration

Chandler facilitates information synchronization and updating between repositories. Its collaboration environment can facilitate discussions, organization and coordination of projects, creation and reviewing of documents and management of information flow and tasks.

Chandler as a Platform

Chandler is extensible and can be both employed and further extended by programmers and end-users in many different ways:

End-users can create customized views of their data, mixing local and remote data with sophisticated sharing policies. They can customize the look and feel by changing colors, images, etc., as well as add new buttons and menu items and other widgets to perform custom tasks. Finally, they can instruct agents to perform complex actions, which take place over a span of time automatically.

Programmers can create new parcels (the basic unit of organization of Chandler code) to manage any type of information. Parcels have access to all of Chandler's subsystems. They can contain agents that can automatically respond to events.

Chandler's user interface is shown in Figure 3.2, depicting a week in a calendar view.

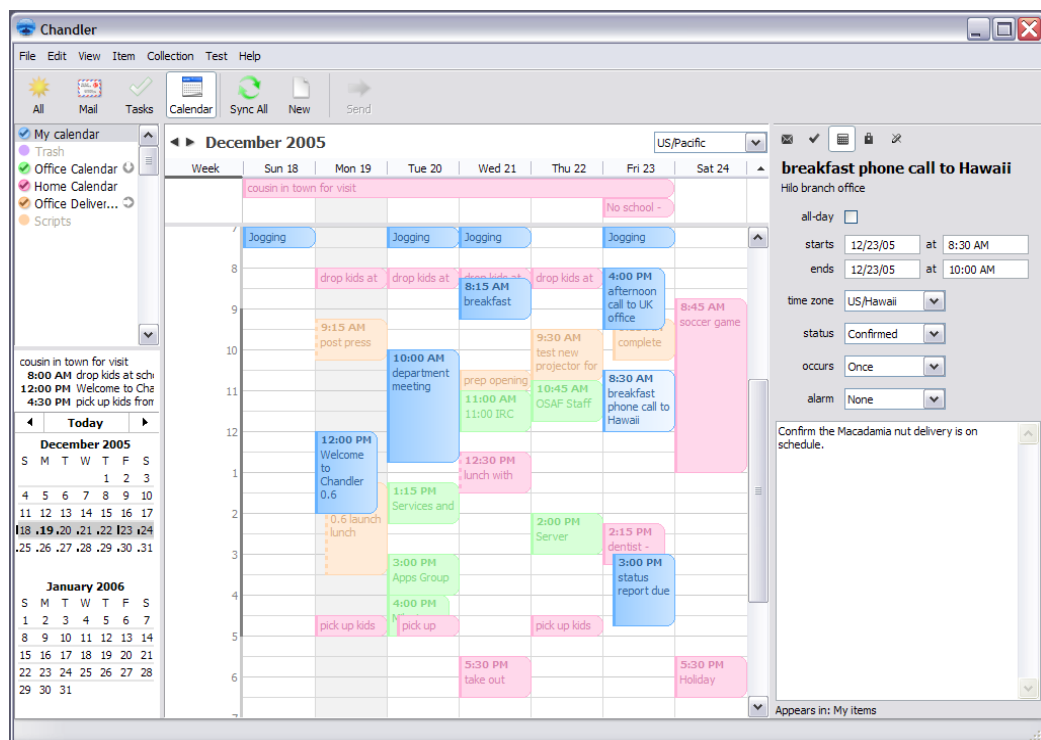


Figure 3.2: The Chandler user interface

3.1.3 DeepaMehta

DeepaMehta (Richter, Völkel, & Haller, 2005) is a personal knowledge management (PKM) tool that integrates information objects of all kinds into a coherent and intuitive user environment. It is a service oriented application framework with a data model based on topic maps (Biezunski & Bryan, 1999) and a user interface that renders them as a graph (see Figure 3.3). Any kind of information as well as the relations between information items can be displayed and edited in the same space. DeepaMehta is designed in such a way that there is no need for overlapping windows, menu bars or dialog boxes.

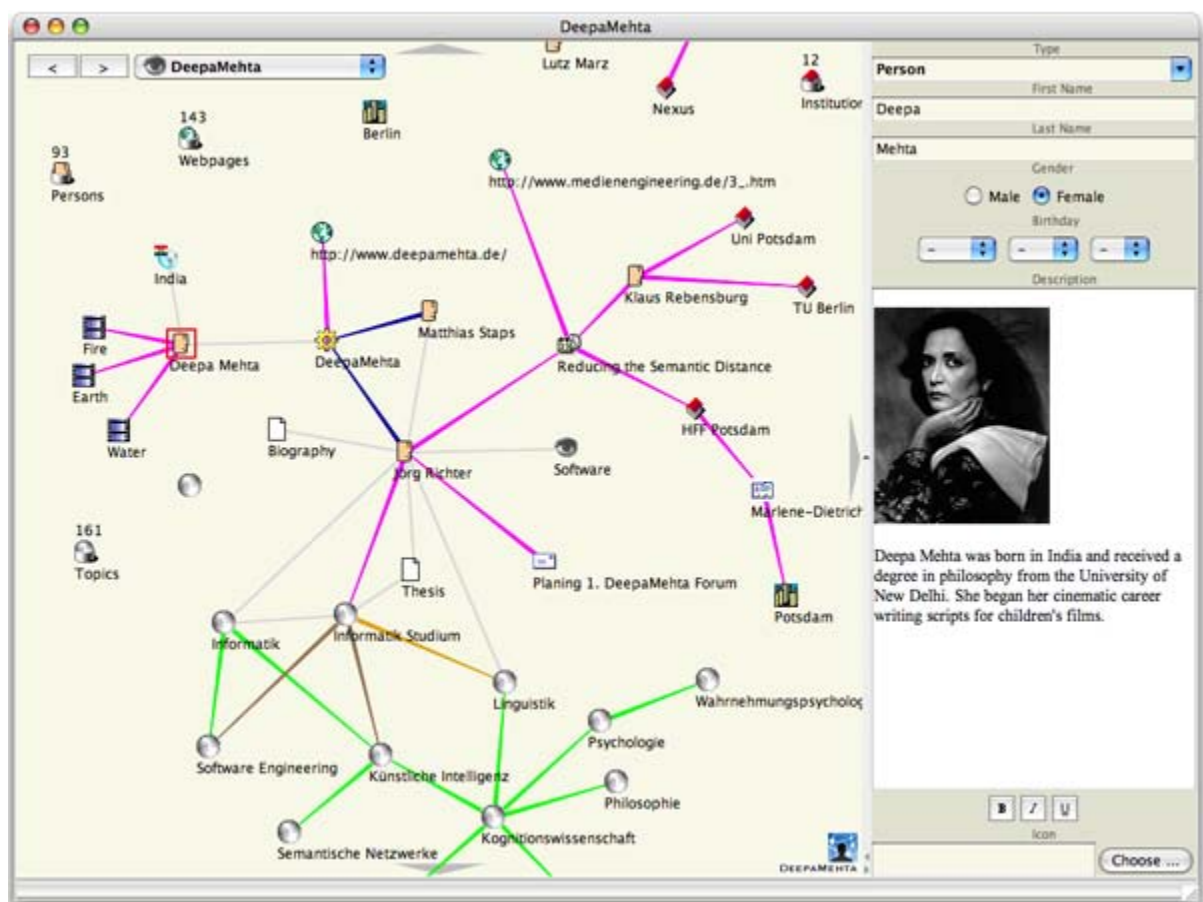


Figure 3.3: DeepaMehta's user interface showing a topic map

User Interface

Stable Views in DeepaMehta let the user focus on the task itself, without leaving the work-context. The user can read an e-mail, link it to an existing topic, attach a note to it, search for related media, save the search results, make semantic statements and spatially arrange all these items on the screen in one and only view.

Constructive Browsing: each resource visited in DeepaMehta is automatically represented as a topic in the current workspace. When surfing the web or accessing other resources a map of viewed objects is automatically created. Searches and search results are also represented in the same way. This spatially arranged map visualises the work process and it is persistent, automatically saved and fully navigable. Furthermore a snapshot of every page viewed is taken and automatically stored in the repository, so it can be referred to and annotated, even when it is offline or no longer on the web.

3.1.4 Fenfire

Fenfire¹⁵ is structured around the user's tasks. Nelson (Nelson, 1999) uses the term *applitude* for such functionality. An *applitude* is not separated from the rest of the system. When information from a different *applitude* is needed it is always available. Fenfire is a free software project aiming at implementing the *applitude*-oriented user interface concepts on top of an RDF graph.

Views

RDF Focus & Context views: The basic visualization of RDF in Fenfire is a browsable hyperstructure. The focus at any time is an RDF node shown in the middle of the screen. The context is also shown as nodes connected to the main one as in Figure 3.4.

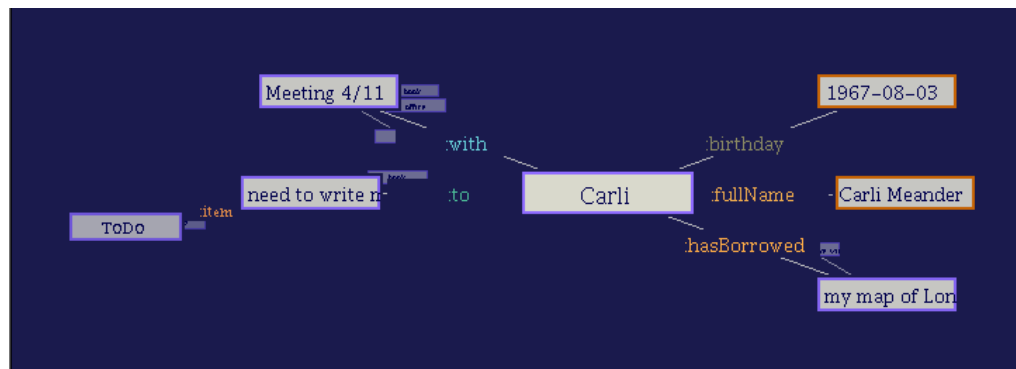


Figure 3.4: The RDF browser of Fenfire

The RDF browser usually shows only a subset of a node's relationships. The user, however, can "switch" individual relationship types on and off. If there are too many

¹⁵ <http://fenfire.org/manuscripts/2004/hyperstructure/>

connections along the active relationship types, Fenfire shows only as many as possible that fit on the screen and allows the user to scroll through the list of connections.

Buoys: Buoys are an important part of Fenfire. The idea of buoys is that some extra information (e.g., a comment, a connection to another document) is anchored to the focused document and it floats around the view relatively freely. Buoys move around smoothly when the view is panned or zoomed.

The intent of buoys is to reduce the cognitive overhead of browsing and editing the structure by:

- providing context to the current node;
- having non-disruptive motion between nodes; and
- showing the target document directly.

Libvob

In Fenfire users can add connections to views. For example, when showing meetings on a timeline, users can add buoys showing the participants of each meeting. To allow such connections to be added without changing the view's code, Libvob has been developed - a flexible user-interface toolkit.

Libvob provides functionality to not only construct a scene graph (i.e., specify what to draw on the screen) but to also specify which parts of the scene graph correspond to which objects in the application model (the RDF graph).

FenPDF

FenPDF is the first concrete prototype of Fenfire's architecture that makes use of buoys, Libvob and RDF. FenPDF (see Figure 3.5) is used to structure a set of articles in PostScript or PDF format. Users can transclude pieces of articles onto spatial canvases: infinite, scrollable papers. Transclusions are automatically bi-directionally connected to the article they are from. A buoy shows a shrunken version of the article, and clicking on the buoy brings the article to the centre for the user to read. Additionally, the user can type text onto the canvases, and link two pieces of text on different canvases (linked canvases are also shown as buoys).

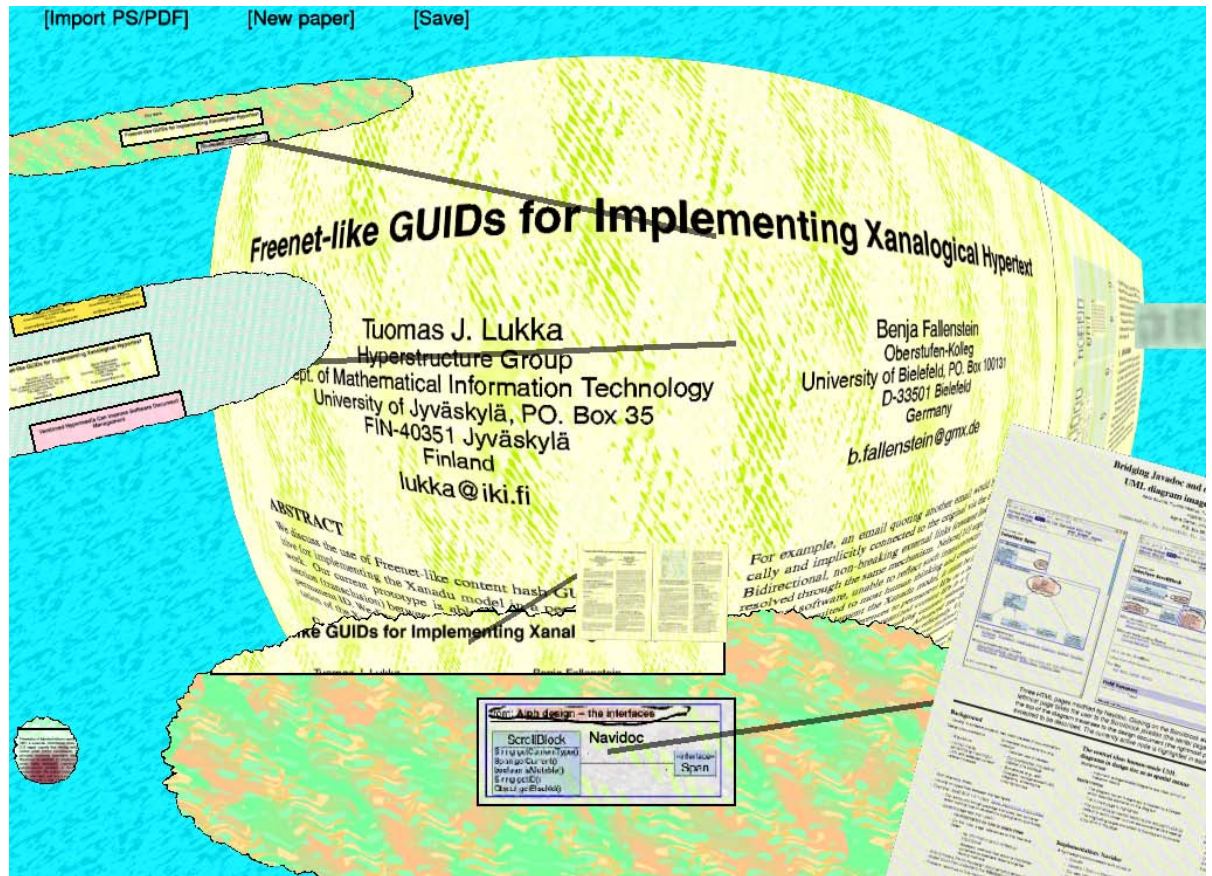


Figure 3.5: FenPDF, using Fenfire's user interface technologies

3.1.5 Gnowsis

Gnowsis (Sauermann et al., 2006) is a semantic desktop with a strong focus on extensibility and integration. The goal of Gnowsis is to enhance existing desktop applications and the desktop operating system with Semantic Web features. Its primary use is Personal Information Management (PIM), technically realized by representing the user's data in RDF. Although the technology used is the same, communication, collaboration, and the integration with the global Semantic Web is not addressed by the Gnowsis system.

Gnowsis consists of two parts:

the Gnowsis Server which does all the data processing, storage and interaction with native applications; and

the graphical user interface (GUI) part (see Figure 3.6), currently implemented as Swing GUI and some web-interfaces.

The interface between the server and GUI is clearly specified, making it easy to develop alternative GUIs. It is also possible to run Gnowsis Server standalone, without a GUI. Gnowsis uses a Service Oriented Architecture (SOA), where each component defines its interface. After the server starts the component, the interface is available as XML/RPC service¹⁶, to be used by other applications.

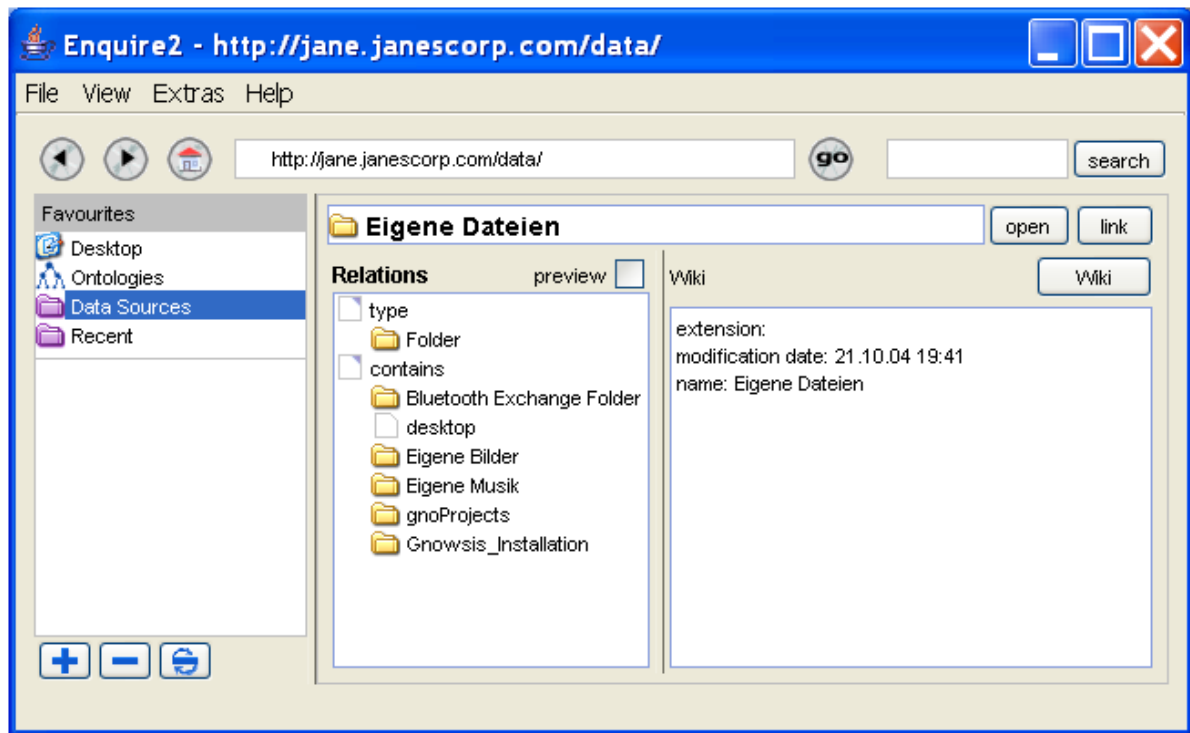


Figure 3.6: The Gnowsis user interface (Enquire2)

External applications like Microsoft Outlook or Mozilla Thunderbird are integrated via Aperture data-source¹⁷; their data is imported and mirrored in Gnowsis. Some new features were also added to these applications using plugins, e.g., in Thunderbird users can relate e-mails to concepts within their personal ontology.

The Gnowsis Server

¹⁶ <http://www.xmlrpc.com/>

¹⁷ <http://aperture.sourceforge.net/>

The architecture of the Gnowsis Server is shown in Figure 3.7. The main component is the RDF storage repository which is based on Sesame2¹⁸. Gnowsis has four different stores:

1. The PIMO store handles the information in the user's Personal Information Model.
2. The resource store handles the data crawled from Aperture data-sources.
3. The configuration store handles data about available data-sources, log-levels, crawl-intervals, etc.
4. The service store handles data created by various Gnowsis modules, such as user profiling data or metadata for the crawling of data-sources.

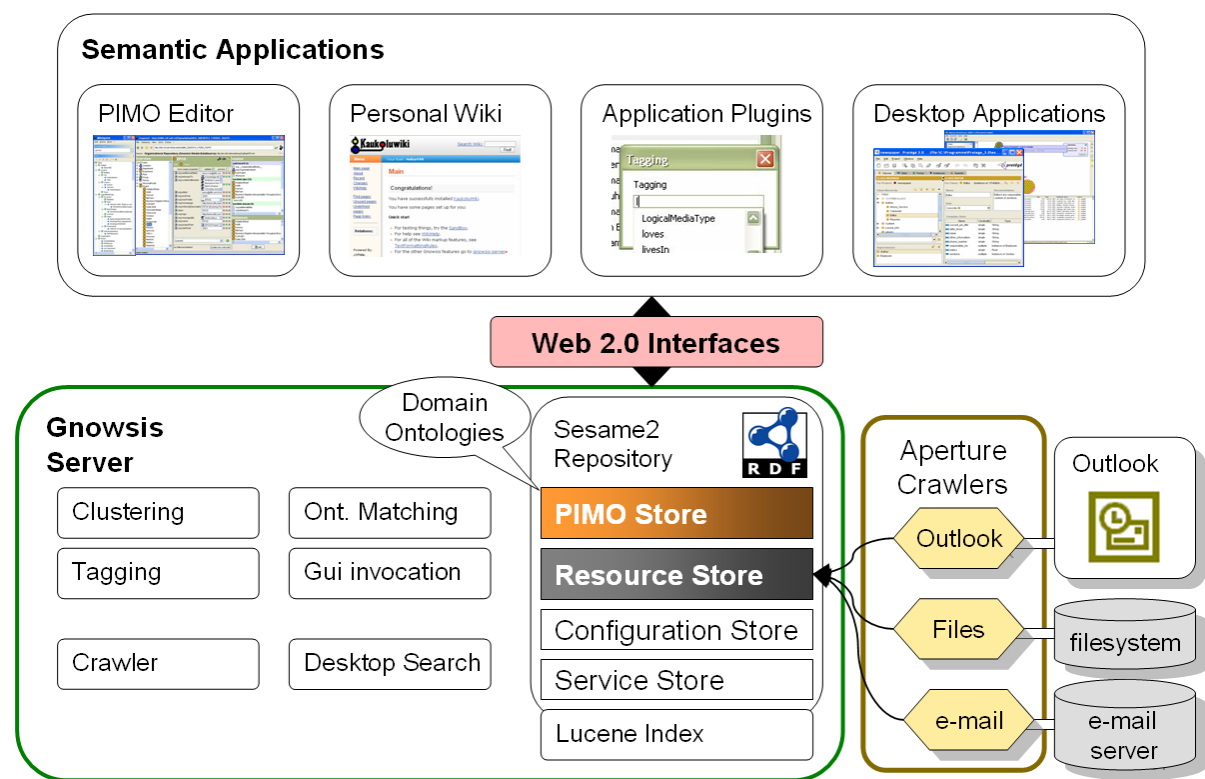


Figure 3.7: The Gnowsis architecture

3.1.6 Haystack

The Haystack project (Karger, Bakshi, Huynh, Quan, & V. Sinha, 2005) is developed by the MIT Computer Science and Artificial Intelligence Laboratory. It is an open source

¹⁸ <http://www.openrdf.org/>

research prototype that is driven by the idea that every individual deals with information in a different way. All have different needs and preferences regarding:

which information objects need to be stored, viewed and retrieved;

what relationships or attributes should be stored to help find information later;

how those relationships or attributes should be presented and manipulated when inspecting objects and navigating the information space; and

how information should be gathered into coherent workspaces in order to complete a given task.

Haystack aims to give end users significant control over all of the facets mentioned above. Haystack stores (by reference) arbitrary objects of interest to the user. It records arbitrary properties of relationships between the stored information. The user interface is flexible and supports manipulation of objects and properties in a meaningful fashion.

Haystack uses URIs to name everything – a digital document, a physical document, a person, a task, a command, a menu operation, etc. Once named, the object can be annotated, related to other objects, viewed, and retrieved.

Retrieval is supported by recording arbitrary (predefined or user-defined) properties to capture any attributes of or relationships between information that the user considers important. The properties serve as useful query arguments, as facets for metadata-based browsing, or as relational links to support associative browsing.

The user interface is designed to be flexible within the information space: Haystack supports views that describe how different types of information should be presented. The views are customizable data in the system, so they can be imported or modified by a user to handle new types of information, new properties of that information, or new ways of looking at old information. A similar approach is used to describe workspaces for a particular user task. In a related vein, operations that manipulate the data are reified into data that can similarly be modified by the user to customize the way they manipulate information.

A convenient way to describe Haystack's user interface is to describe an example view. In Figure 3.8, a screenshot of Haystack shows the management of an individual's email inbox. Haystack shows the user's inbox in the primary browsing pane. The layout is tabular, with columns listing the sender, subject, and body among other things. The

collection includes a “preview” pane for viewing selected items. On the right hand side of the screen is a “holding area” for arbitrary items. Various aspects of the message are shown, including the body, attachments and recommended categories. Attributes displayed about the person include messages to and from them, address and phone number.

The bottom of the left panel shows that the “Email” task is currently active, and lists various relevant activities and items that the user might wish to invoke or visit while performing this task, as well as a history of items that the user previously accessed while performing this task.

The user can click on any item on the screen in order to browse to a view of that item. Similarly, the user can right click on any visible item in order to invoke a context menu of operations that can be applied to that object. In Figure 3.8, the user has right-clicked on a message sender; a context menu has opened up listing operations that might be invoked on that person, such as sending him an email message, initiating a chat, or entering him in the address book.

Finally, the user can drag one item onto another in order to “bind” those two items in an item-specific way – for example, dragging an item onto a collection places the item into the collection, while dragging an item into a dialog box argument field binds that argument to the dragged item.

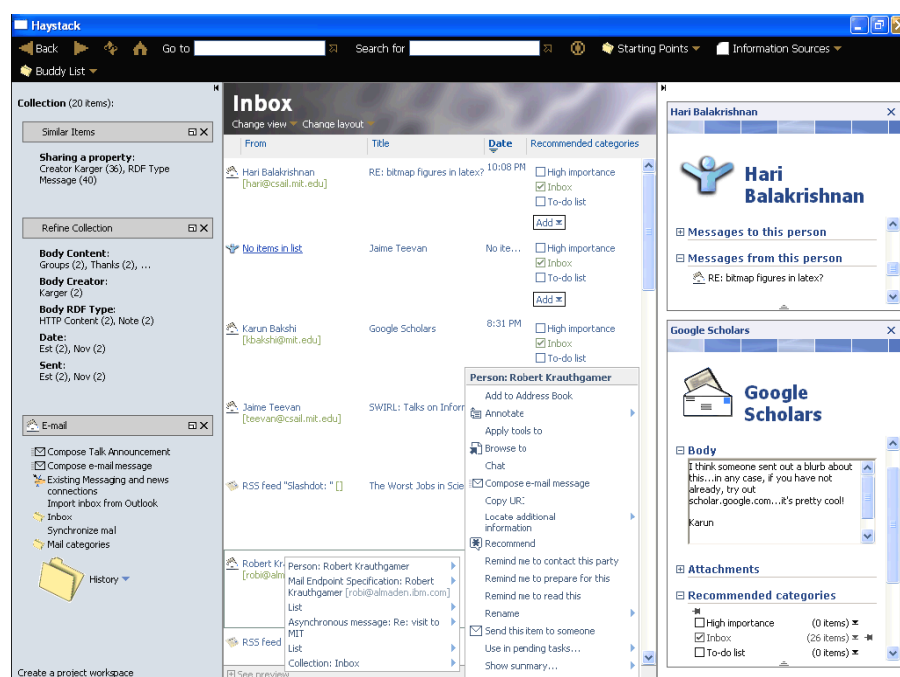


Figure 3.8: Haystack viewing a user’s inbox collection

Some of the items in Haystack's inbox are not email messages. In the screenshot some RSS feeds and a person are shown. The RSS message has a sender and a date, but the person does not. This is characteristic of Haystack: the inbox is a collection like all other Haystack collections, distinguished only as the collection into which the user has specified incoming email (and news) to be placed. It is displayed using the same collection view as all other collections. Any items can be placed in the collection.

A "browsing advisor", also showed in the left pane, suggests various "similar items" to those in the collection and ways to "refine the collection" being viewed, e.g., limiting to emails whose body contains certain words, or that were sent at a certain time.

3.1.7 IRIS semantic desktop

IRIS (Cheyer, Park, & Giuli, 2005) is an open source application framework that has been developed at SRI International¹⁹ as part of the CALO²⁰ research project, one of the two projects funded under DARPA's "Perceptive Assistant that Learns" (PAL) program²¹. The goal of the PAL program is to develop an enduring personal assistant that "learns in the wild," evolving its abilities more and more through automated machine learning techniques rather than through code changes. IRIS allows users to create a "personal map" across their office-related information objects. IRIS is an acronym for "Integrate – Relate – Infer – Share". IRIS can be described according to these terms as follows:

Integrate

One of the main goals of IRIS is to integrate data from several applications using reified semantic classes and typed relations. For instance, it should be possible to express that "File F was presented at Meeting M by Tom Jones, who is the Project Manager of Project X," even if the file manager, calendar program, contact database, and project management software are separate applications.

IRIS provides integration at three levels:

¹⁹ <http://www.sri.com/>

²⁰ <http://www.ai.sri.com/software/CALO>, CALO is an acronym for "Cognitive Assistant that Learns and Organizes"

²¹ DARPA's PAL program: <http://www.darpa.mil/ipto/programs/pal/>

1. Information resources and their corresponding applications can be made accessible to IRIS for instrumentation, automation, and querying.
2. A knowledge base provides the unified data model, persistence store, and query mechanisms across the information resources and semantic relations among them.
3. The IRIS user interface framework allows plug-in applications to embed their own interfaces within IRIS, and to interoperate with global UI services of IRIS.

IRIS provides several integrated office applications: email, web browser, calendar, chat, file explorer and data editor/viewer for browsing and editing the knowledge base.

The IRIS user interface (Figure 3.9) has two collapsible side panels that frame the main application window: the left one serves as an application selector and the right one for managing semantic links of the selected application object and presenting contextual suggestions.

Relate

IRIS supports storing and relating information objects (e.g., email messages, calendar events, files, etc.) to each other across applications and across users. It, thus, allows reasoning and inference upon these information resources. IRIS provides a framework for harvesting application data and instrumenting user actions in IRIS applications. The harvesting of data refers to importing external data into semantic (ontology-based) structures that can be later exposed to other IRIS plug-ins or external applications for consumption.

Infer

A very interesting - and uncommon to many other semantic desktops - feature of IRIS, is machine learning and the implementation of a plug-and-play learning framework. Authors in (Cheyer et al., 2005) consider machine learning as a possible solution around a main problem of the Semantic Web affecting its mass adoption: Who is going to enter all of the required links and knowledge?

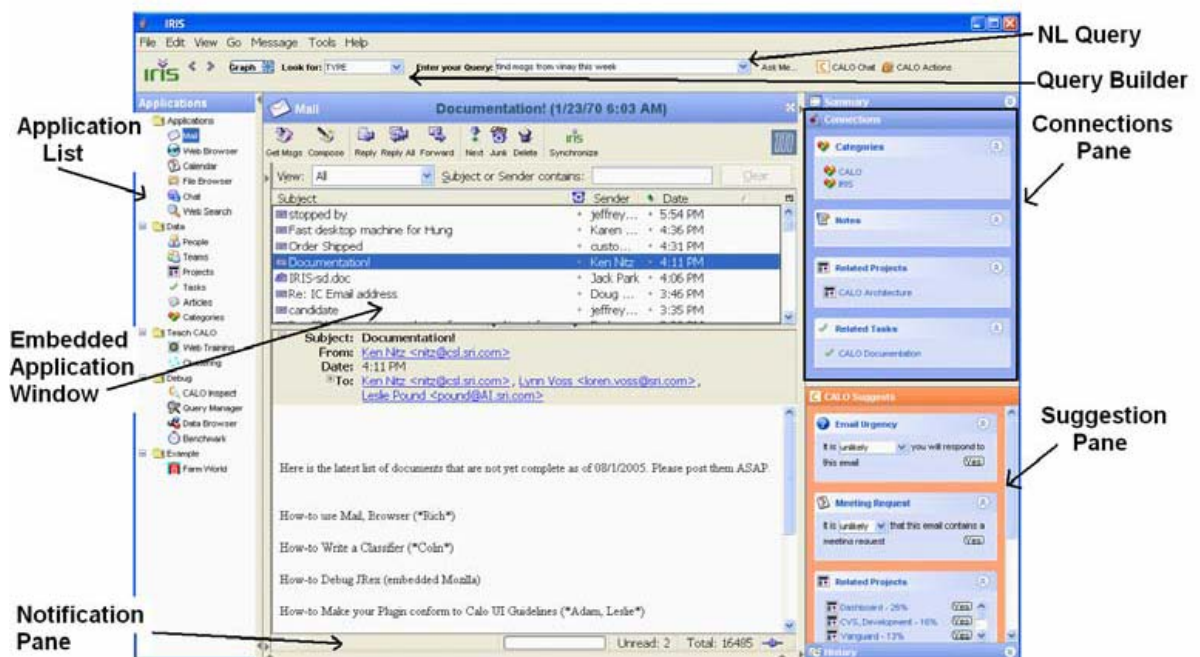


Figure 3.9: The IRIS user interface

Currently, IRIS has integrated the following learning components:

Email harvesting: automatic harvesting of messages and addition as semantic instances in the knowledge base, linking email elements (e.g., contacts) with records in the knowledge base.

Contact/expertise discovery: when a new person is added to the knowledge base, IRIS tries to discover additional contact information about this person and his expertise.

Learn from files: similarly to email messages, IRIS can harvest files - currently LaTeX, BiB and Microsoft Office files are supported.

Project creation: clustering algorithms are applied to the user's email to propose new projects to be added in the knowledge base.

Classification according to project: leveraging the textual content and relations extracted for projects, people, and files, a Bayesian classifier is applied to hypothesize relationships between projects and objects such as emails, files, web pages, and calendar appointments.

Higher-level reasoning: specialized reasoners examine events in the user's activities and attempt to make useful inferences. A typical example is when the user opens up an email and IRIS predicts whether he should reply. If the user gives negative feedback to IRIS about its suggestion, then the reasoners will try to adapt themselves accordingly.

Share

In the first version of IRIS, a simple collaborative functionality using a Jabber²²-based transport mechanism was used. However, it had no locking mechanisms and is temporarily removed from IRIS. Still, the authors of IRIS consider using functionality and applications that will require the development of an infrastructure to support collaborative team decision making, as well as reasoning over a shared document space.

3.1.8 NEPOMUK

One of the main problems that results from the mass adoption of the web is information overload, that requires smarter and more fine-grained computer support for networked information, and that has to balance between personal and group data, while simultaneously safeguarding privacy and establishing trust. In other words, the current computing infrastructure does not really support knowledge workers all that well: for example, sending a single file to a mailing list multiplies the cognitive processing effort of filtering and organizing this file times the number of recipients – leading to more and more of peoples' time going into information filtering and organization activities. Current application infrastructures do not support interconnection between separate data items, like the author of a document and her corresponding entry in the address book (S. Decker & Frank, 2004).

Several new technology thrusts have now emerged which could dramatically impact how people interact and collaborate: The Semantic Web, P2P Computing, and Online Social Networking. NEPOMUK is a European IST project aiming at creating a new technical and methodological platform using these technologies: the Social semantic desktop. It enables users to build, maintain, and employ inter-workspace relations in large scale distributed scenarios. New knowledge can be articulated in semantic structures and be connected with existing information items on the local and remote desktops. Knowledge, information items, and their metadata can be shared spontaneously without a central infrastructure. NEPOMUK will realize a freely available open source integration framework with a set of standardized interfaces, ontologies and applications (Sauermann et al., 2005).

High-level architecture of the Social semantic desktop

Figure 3.10 shows the highest-level architecture and connections between social networks and the P2P infrastructure that connects those social networks to individuals' desktops.

Traditional semantics, knowledge representation, and reasoning research are now interacting with other research areas, which not individually but together may have the explosive impact of the original Web (S. Decker & Frank, 2004):

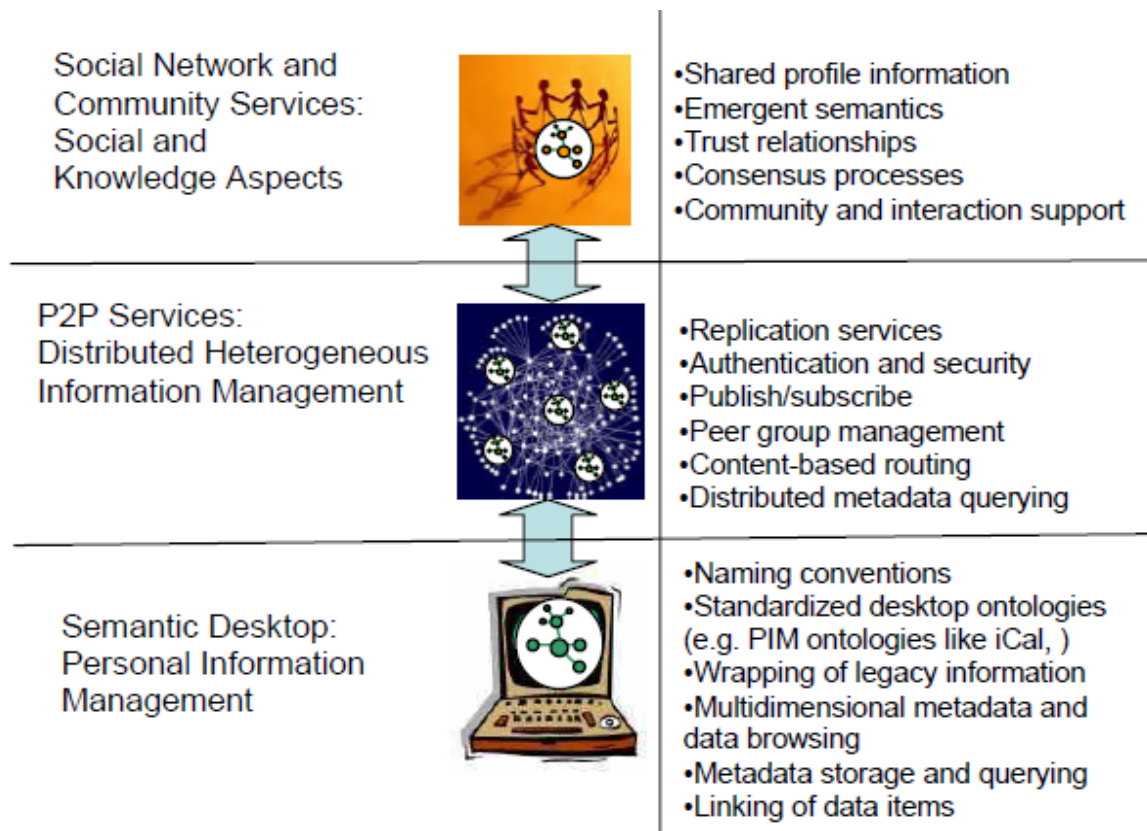


Figure 3.10: High-level architecture of the Social semantic desktop

1. The Semantic Web effort²³ provides standards and technologies for the definition and exchange of metadata and ontologies. Available standard proposals provide ways to define the syntax (RDF) and semantics of metadata based on ontologies (OWL).

²² Jabber Instant Messaging: <http://www.jabber.org/>

²³ <http://www.w3.org/2001/sw/>

Research covering data transfer, privacy and security issues is now also under development.

2. Social software maps the social connections between different people into the technical infrastructure. As an example, Online Social Networking makes the relationship between individuals explicit and allows discovering previously unknown relationships. The most recent Social Networking sites also help form new virtual communities around topics of interest and provide means to change and evolve these communities.
3. P2P and Grid computing develops technology to network large communities without centralized infrastructures, for data and computation sharing. P2P networks have technical benefits in terms of scalability and fault tolerance, but a main advantage compared to central sites is that they allow building communities without centralized nodes of control, thus avoiding for someone to own big, expensive central facilities. Recent research has provided initial ways of querying, exchanging and replicating data in P2P networks in a scalable way.

Authors in (S. Decker & Frank, 2004) argue that next generation Internet applications will support collaboration and information exchange in a P2P network, connecting online decentralized social networks, and enabling shared metadata creation and evolution by a consensus process, the result being the Social semantic desktop.

3.1.9 Comparison of Surveyed Implementations

An overview of the most important features of the surveyed semantic desktops is given in Table 3.1. Several conclusions can be derived while observing the table:

Common features

Most of the semantic desktops share some common features. Almost all of the surveyed systems provide:

- Context-aware navigation.
- Different customizable views.
- Integration of several data types.
- Interrelating data items.

Additionally, there is a clear trend to implement semantic search especially in more contemporary systems like NEPOMUK, Gnowsis, Haystack and IRIS.

Differences

Some unique features are only provided by separate systems. For example, DeepaMehta provides graph visualization and NEPOMUK and Chandler offer means of social collaboration. Additionally, IRIS makes use of advanced artificial intelligence technologies such as machine-learning and inference.

Completeness of supported features

From Table 3.1, it is obvious that the most complete semantic desktop systems in terms of supported features are NEPOMUK, Haystack and IRIS.

	Chandler	DeepaMehta	Fenfire	Gnowsis	Haystack	IRIS	NEPOMUK
context-aware navigation	●	●	●	○	●	●	●
different customizable views	●	●	●	○	●	●	●
graph visualization	○	●	○	○	○	○	○
integration of several data types	●	●	○	●	●	●	●
interrelating data items	●	●	●	●	●	●	●
semantic search	○	○	○	●	●	●	●
social collaboration	●	○	○	○	○	planned	●

	Chandler	DeepaMehta	Fenfire	Gnowsis	Haystack	IRIS	NEPOMUK
suggestion of related items	○	●	○	○	●	●	●
suggestion of related tasks	○	○	○	○	●	●	○

● = supported, ○ = not supported or not obvious from literature

Table 3.1: Overview of surveyed semantic desktops' features

3.2 Semantic Desktops in Software Engineering

There are several approaches for managing knowledge in software engineering. Holz (Holz, 2003b) presents a detailed life-cycle model for POKM (Process-Oriented Knowledge Management) that is specific to software development processes. This life-cycle model for Software Engineering Process-Oriented KM (SEPOKM) is integrated into the life-cycle model performed by the organisation's Process Group and becomes an essential part of a continuous organisational learning process.

Natali and Falbo (Natali & Falbo, 2002) present an infrastructure to deal with knowledge management in software engineering environments (SEEs). This infrastructure is applied to manage product software quality knowledge in ODE, an ontology-based SEE.

Borges (Borges & Falbo, 2002) stores and shares the experience obtained in software process definition. In order to achieve this, they built ProKnowHow, a tool that supports the standard software process tailoring procedure for each project, providing KM support.

Furthermore there are emerging efforts in research towards exploiting semantic web technologies in software engineering in order to manage software development knowledge (Happel & Seedorf, 2006), (Eberhart & Argawal, 2004), (Witte, Y. Zhang, & Rilling, 2007). For example, SemIDE (Bauer & Roser, 2006) is a framework which is based on meta-models in order to describe semantically software engineering processes. OSEE (Thaddeus & Kasmir, 2006) is a tool which aids the construction of a knowledge base which captures software related artefacts.

Ankolekar et al. (Ankolekar, Sycara, Herbsleb, Kraut, & Welty, 2006) provide an ontology to model software, developers, and bugs. Happel et al. (Happel, Korthaus, Seedorf,

& Tomczyk, 2006) present an approach addressing the component reuse issue of software development by storing descriptions of components in a Semantic Web repository, which can then be queried for existing components.

There have also been many previous attempts to design a Relational database schema to store RDF/Owl data (Pan & Heflin, 2003), (Abadi, Marcus, Madden, & Hollenbach, 2007). SeRiDA (Athanasiadis, Villa, & Rizzoli, 2007) combines object-oriented programming and relational databases, with semantic models. Another approach (Noll & Ribeiro, 2007) relies on manual annotation by a user to explicitly associate instance elements in software artefacts with concepts in the ontology. However, this technique implies a considerable amount of effort since there is a significant number of software engineering elements.

The IBM Jazz platform²⁴ provides teams with the ability to: (a) Collaborate via social networking capabilities and automation of the individual and team workflows throughout the software lifecycle; (b) Report via the provision of fast access to fact-based information; (c) Choose their own path via the provision of an open and extensible architecture which offers flexibility to assemble own software delivery platform.

Jazz is a technology platform, which is able to track all aspects of a development project, including events that affect the coding process, potential problems and their causes. Furthermore, software developers are able to discuss their problems and pass code back and forth to fix them, using RSS feeds.

Product offerings that are built on the Jazz platform are able to leverage a rich set of capabilities for team-based software development and delivery. Rational Team Concert family, Rational Quality Manager and Rational Requirements Composer are the first offerings built on Jazz technology.

CollabNet Inc.²⁵, focuses on providing collaboration software and technology to software development teams that are globally distributed. CollabNet provides a set of specialized collaborative tools for software developers in order to manage: software

²⁴ <http://www-01.ibm.com/software/rational/jazz/>

development, knowledge management, communication management, project management, and security and permissions management.

3.3 Semantic Wikis

In this section wikis are introduced and a description of how semantic wikis extend conventional ones is given. Analogously to semantic desktops, a survey of semantic wiki solutions is presented, focusing on the functionality and technologies used. The breadth of offered functionalities by the systems gives an holistic overview of semantic wikis and affected the decision about choosing the proper systems to survey. The surveyed systems are: COW, IkeWiki, Kaukolu, Makna, OntoWiki, OpenRecord, Platypus Wiki, Rhizome, Semantic MediaWiki, SemperWiki, SweetWiki and WikSAR. Finally, the comparison of the different implementations is given, stressing the common features and differences between them.

3.3.1 Wikis

The first Wiki, WikiWikiWeb, was introduced by Cunningham (W. Cunningham & Leuf, 2001). Wikis were increasingly adopted in enterprises as collaborative software. Common uses included project communication, intranets, and documentation. Oren in (Oren, 2005) gives a brief introduction to Wikis. Wikis are interlinked web sites that can be collaboratively edited by anyone. Pages are written in a simple syntax so that even novice users can easily edit pages. The syntax consists of simple tags for creating links to other Wiki pages and textual markups such as lists and headings.

The user interface of most Wikis consists of two modes: reading and editing modes. In reading mode, the user is presented with normal web pages that can contain pictures, links, textual markup, etc. In editing mode, the user is presented with an editing box displaying the Wiki syntax of the page (containing the text including the markup tags) (Oren, 2005).

There is a great evolution in Wiki engines over the last few years. Most of them are open source, thus enabling every single user to setup a Wiki with zero cost. Many

²⁵ <http://www.collab.net/>

organizations and user communities capitalize on Wikis as a mean of collaboration. A typical example is a documentation Wiki of an open source project, where users can collaboratively add documentation about the project. In this way, the heavy task of editing is shared among the members of the whole community, while still allowing anybody to quickly find relevant documentation. A very popular Wiki like Wikipedia²⁶ can grow at very fast rates, since any interested visitor can edit and create pages at will.

A side effect of a large Wiki is that the user can be frustrated when searching for information. Since the largest percentage of the information in a Wiki is textual, the only way to find information is through a keyword-based search. This side effect is the result of unstructured accumulated information, and hence, it poses problems for knowledge management and productivity. The only semantics of Wiki pages lies in the links between pages. Most Wiki engines generate navigational structures from these links: one can see all pages linking to the current one, and go to these related pages. But this navigation through related pages is quite limited, and does not address the need for more advanced information retrieval.

Popular Wiki features

The key features that made Wikis popular and successful were (Kotelnikov et al., 2007):

Collaboration aspect: All information became immediately available for everybody.

Simplicity in information interlinking: To create a link between documents it is sufficient to put the name of the desired document. So creation of a complex graph became easy.

Simplicity of document creation: If a target document does not exist, then it is sufficient to click on the link to start the editing process.

Openness for reading and editing: The information is open to readers as well as to editors. This openness increases considerably the involvement of readers in the process of the information creation and eliminates borders between document creators and their consumers.

²⁶ <http://www.wikipedia.org/>

Openness for experiments: Another factor contributing to the information editing is the absence of fear to destroy existing documents. This is achieved by using modification histories for all the information.

Fine granularity of information: The simplicity of document creation and interlinking encourages creation of individual pages for each topic, term, or word. In Wikis this information can be easily linked and found in different contexts resulting in increased reusability of information.

Refactoring: Simplicity in document creation and versioning encourage refactoring of the information; if a document becomes too large, it can be easily split into many smaller ones.

Immediate rewards: Each modification in documents is published and visible right away.

Popular Wiki engines

A method for ranking Wiki engines is to make Google searches to see how many pages are being served (and indexed by Google) from the different Wiki engines. According to a survey²⁷ conducted in August 2006 using this method, the most popular Wiki engines are:

1. MediaWiki
2. Twiki
3. Confluence
4. TikiWiki
5. PukiWiki
6. JotSpot
7. MoinMoin
8. PBwiki
9. PmWiki
10. Socialtext

Comparing Wiki engines

The Wiki Matrix²⁸ is a tool for comparing Wiki engines. It consists of a searchable database where a very large number of Wiki engine characteristics have been recorded, for

each of the most common engines. There are no evaluations – just listings of features, technical details and other plain facts. A comparison of the above Wiki engines that appear in the Wiki Matrix gives the following insights:

Common properties:

- Most are free and open source.
- Most use PHP or Perl for programming.
- Most run on several different operating systems.

Common features:

- Page history function.
- Preview function.
- Unicode support.
- Full text search.
- Support for more than 10 languages.
- Page permissions.
- All but one support internal comments and CSS.
- RSS feeds.
- “Recent changes” page.

Features which seem to be irrelevant for success:

- WYSIWYG editing.
- There is no correlation between popularity and storage method.
- The CamelCase notation is not the only successful linking style. Some use [square brackets] or similar constructs instead.
- HTML tag support.

²⁷ <http://www.wikicreole.org/wiki/WikiPopularity>

- Math formula support.
- FAQ tags.
- Scripting.
- Feed aggregation.
- Section editing.

3.3.2 Semantic Wikis

Semantic wikis try to overcome the problems related to information retrieval that are mentioned in Section 3.3.1 by combining Semantic Web standards such as RDF/S or OWL with the Wiki paradigm. One idea is to annotate structure in the Wiki by providing metadata for existing features such as links and pages. On the other hand, one can strive to completely represent the Wiki content using instances of the respective ontology language (Malte, 2006).

These annotations are in fact formal descriptions of resources that assist in describing those resources. The difference with a regular Wiki is that a semantic wiki enables users to additionally describe resources in a formal language, instead of just natural language. The authoring effort is similar to that of regular Wikis: semantic annotations are very similar to the layout or structural directives that ordinary Wikis use.

Using the formal annotations of resources, semantic wikis offer additional features over regular Wikis. Users can query the annotations directly (“show me all authors”) or create views from such queries. Also users can navigate the Wiki using the annotated relations (“go to other books by John Grisham”), and users can introduce background knowledge to the system (“all poets are authors; show me all authors”) (Oren, Breslin, & S. Decker, 2006).

The potential benefits of semantic wikis relative to the traditional ones have been well summarized in a recent Project10X report (M. Davis, 2006):

²⁸ <http://www.wikimatrix.org>

- Concept-based rather than language-based searching: queries span vocabularies, languages, and search engines.
- Question answering rather than simple retrieval. Also, overlaying ontologies and knowledge bases can integrate with major web search engines.
- More richly structured content navigation, including multiple perspectives, multiple levels of abstraction, dependency/contingency relationships, etc.
- Easy visualization of content structure (categories, taxonomies, semantic nets, etc.). Direct editing of content structure.
- Mining of semantic relationships in content.
- Wiki content linked to dynamic models, simulations, visualizations.
- Wiki content linked with external repositories and file systems (e.g. personal desktop, enterprise servers, web sources, semantic enabled feeds, etc.).
- Richer user access/rights models, including reputation systems.

The rest of this section gives more details about semantic wikis by examining currently supported features of a representative list²⁹ of available implementations.

3.3.3 COW

COW, Combining Ontologies with Wikis³⁰, is an approach to build a semantic wiki, by bringing together two different concepts: easy content evolution with the help of Wikis, and formal knowledge representation using ontologies. It uses the KAON tool suite (Erol Bozsak et al., 2002) as back-end for an ontology editor and a query processor. A simple text Wiki engine complements the system's functionality. This approach is different from other semantic wikis in two aspects: The ontology data is edited and stored outside the text Wiki, and the implementation of query templates assists inexperienced users (J. Fischer, Gantner, Rendle, Stritt, & Schmidt-Thieme, 2006).

²⁹ For an up-to-date and more complete list of available implementations see http://ontoworld.org/wiki/Semantic_Wiki_State_Of_The_Art

³⁰ <http://www.informatik.uni-freiburg.de/cgnm/software/cow/>

Authors in (J. Fischer et al., 2006) argue that the process of learning and using languages like RDF or OWL would contradict the simplicity and ease of use of Wikis. It would be comparable to forcing the use of full HTML in text Wikis, instead of a more user-friendly and minimalistic syntax. This is the reason why, COW uses the KAON tool suite for ontology management and provides components built on top of it, to support the user in manipulating the ontology without having to be aware of the proper usage of the respective ontology language.

COW provides many features, which are described in the following paragraphs.

Ontology editing and browsing

COW has a slot-based graphical ontology editor and a web-based ontology browser. Using the ontology editor, instances may be created without much expertise (Figure 2.11).

Ontology locking

COW's checks are performed when the user issues changes. These checks guarantee that the result of the check-in is comprehensible for the user. The changes to an entity are refused by the system if the editor view of this entity has changed in the meantime. With this strategy users always know the exact effects of their editing operations. All dependencies causing a change of the edit view are considered, even inferred ones.

Transactions for ontology modification

In RDF-based knowledge representation, the smallest unit of information is an RDF triple. As triples of an editing session depend on each other, changes on the knowledge base should be performed according to the ACID principles³¹. Thus COW offers a transaction mode for ontologies, which guarantees that either all changes are applied or none.

Ontology versioning

Versioning in COW is performed in the whole ontology. If users want to restore old versions, they have to set the complete ontology to the old state. The drawback is that of having to roll-back to an older version of the complete ontology which often implies a lot of changes.

Querying ontologies inside the Wiki

In COW the users specify queries, either using a dedicated page, or by directly embedding them into Wiki pages. The query results, usually instances, provide both hyperlinks to the article pages and to the ontology browser. Normal queries are statements in the KAON query language. Thus, users have the burden of learning the KAON query language. But COW offers an additional possible way of querying ontologies with the usage of query templates. These templates are queries with free variable parameters, which have to be filled-in by the user executing the query.

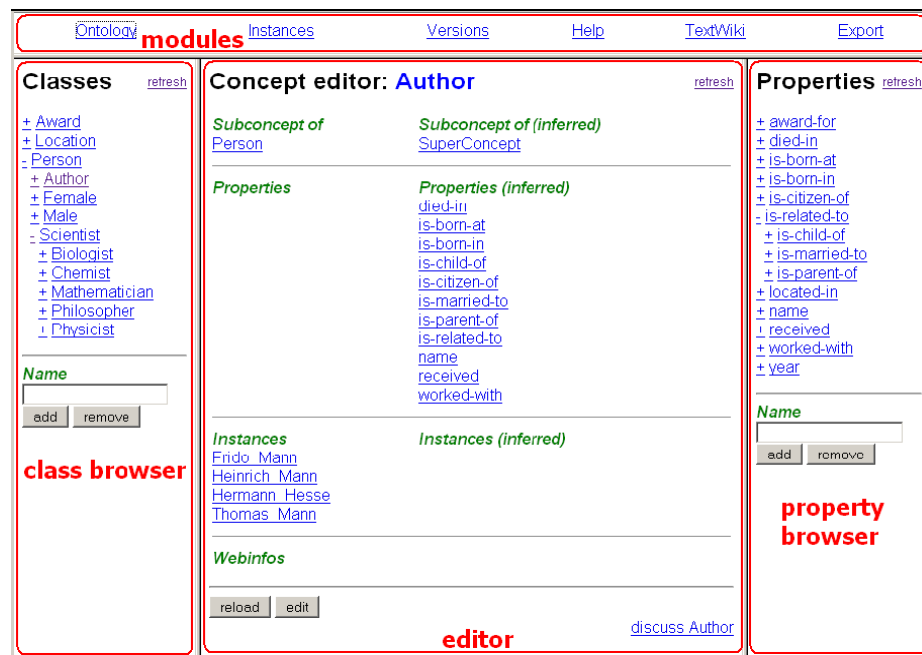


Figure 2.11: COW ontology editor

³¹ Atomicity, consistency, isolation and durability

Persons

Who is from ?

Who was born in ?

Who is related to ?

Who worked with ?

Who is child of ?

Who is parent of ?

Who is citizen of ?

[editieren](#) | [alle Ver](#)

Figure 2.12: COW query template

3.3.4 IkeWiki

IkeWiki (Schaffert, Gruber, & Westenthaler, 2005) is a prototype Wiki system currently being developed at Salzburg Research. IkeWiki serves several purposes:

- It can be used to annotate existing data with semantic terms to improve searching and navigation.
- It can be used to create instance data, based on an existing ontology.
- It can be used as a (limited) tool for creating and editing ontologies themselves.

Users with different roles and different levels of experience can follow at the same time all these purposes.

In the following, the features offered by IkeWiki are described (Schaffert, 2006):

Easy to use, interactive interface

The interface is designed to resemble as closely as possible the Wikipedia interface, which people are familiar with. Furthermore, IkeWiki offers an interactive WYSIWYG editor (using AJAX technology to communicate with the server backend) in addition to the traditional structured text editor, as WYSIWYG editors generally have a better acceptance

among non-technical users. The WYSIWYG editor also supports interactive typing of links and resources. At the same time, IkeWiki offers all the normal editing interfaces (supporting Wikipedia-style structured text) for more experienced users.

In Figure 2.13 a sample article about the “Bilberry” is shown (copied from Wikipedia). Type information is shown below the page title (1). Links to (semantically) related pages are displayed in a separate “references box” on the right hand side (2). The taxonomy box (3) showing the biological classification of the described plant is automatically generated from existing semantic annotations (i.e., Bilberry hasGenus *Vaccinium*) and is an example for context adaptation. Finally, (4) shows interactive typing of links using AJAX technology.

Different levels of experience

Certain advanced functionalities can be hidden from novice users but are available to experienced users.

Different levels of formalisation

IkeWiki supports formalisation of knowledge all the way from informal texts to formal ontologies. This means that parts of the knowledge base might be more formalised than others, and that formal knowledge is in constant evolution.

Reasoning

IkeWiki supports reasoning over the knowledge base, thus allowing the derivation of knowledge that is not explicit.

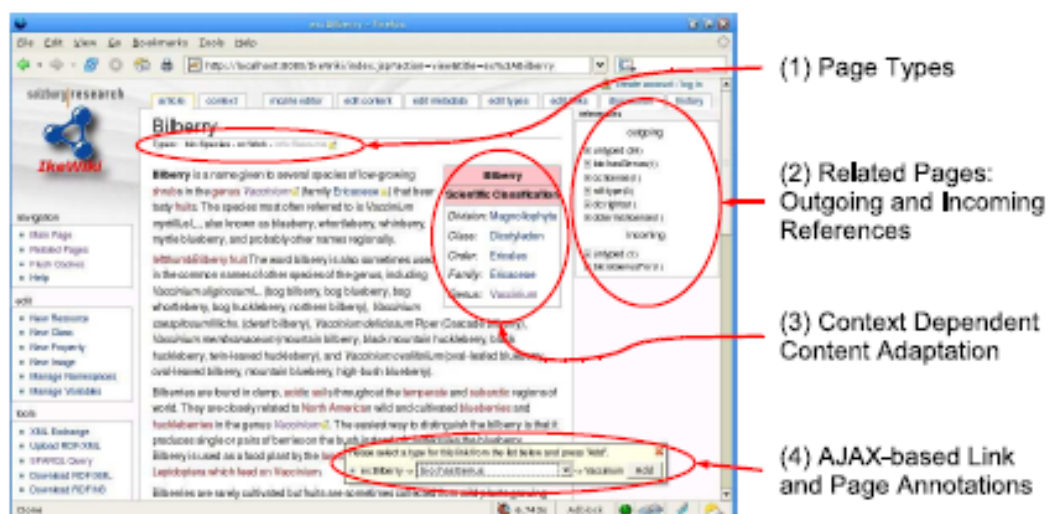


Figure 2.13: IkeWiki user interface

3.3.5 *Kaukolu*

Kaukolu³² (Malte, 2006) builds on JSPWiki³³ and Sesame 2³⁴. Since Kaukolu is built on top of JSPWiki it includes standard Wiki features such as file attachments, access control, plug-in support, or support for multiple back-ends. In addition, it provides the following features:

No restrictions on RDF triples

Kaukolu allows formulating arbitrary RDF on any Wiki page using a slightly extended Wiki syntax. Subjects of RDF triples are not required to represent the URI of the page the triple is located at.

RDF(S) import and export

Being able to associate arbitrary RDF with a Wiki page not only works when formulating RDF but also allows to import RDF. In fact, since RDF Schema is also represented in RDF, one can even import RDFS ontologies to Kaukolu using this method. Imported RDFS ontologies can be used in various ways within Kaukolu. A direct benefit of RDFS ontologies being stored on Wiki pages is that in this way users are able to edit and extend the ontologies used by the Wiki in a straightforward way, using all features a Wiki provides (versioning, collaborative authoring, and viewing diffs). However, changing RDFS using this approach is quite difficult as RDFS has to be changed “manually” without any tool support.

Aliases replacing namespace:localname URIs

Users of Kaukolu are not required to use localnames, labels, or namespaces of RDFS properties in order to express RDF triples using these predicates. For example, typically the user has to write something like (this) *dc:author* “*Author Name*” if he wants to express that the current Wiki page has a Dublin Core author property. In Kaukolu, every RDF instance or

³² <http://kaukoluwiki.opendfki.de/>

³³ <http://www.jspwiki.org/>

³⁴ <http://www.openrdf.org/>

RDFS property may be associated to arbitrary strings (aliases) that can be used instead of the URI/label of the respective property or instance.

Auto-completion for semantic and non-semantic content

Kaukolu provides ontology-based auto-completion support, which proposes aliases based on RDFS range and domains. For example, when typing *Paul knows*, with *Paul* being a *foaf:person*, and *knows* being associated to *foaf:knows*, the system automatically proposes a list of *foaf:persons* defined in the Wiki to complete the RDF triple. Auto-completion works for predicates, too. In case no alias is found in the typed text, Kaukolu assumes that the user does not intend to write triples, and simply proposes names of Wiki pages as auto-completion suggestions, based on the prefix already typed.



Figure 2.14: Kaukolu user interface

3.3.6 Makna

Makna³⁵ aims at providing an easy and user-friendly user interface and powerful Semantic Web technologies in the background. It was developed at the working group Network based Information Systems at the Frei Universität Berlin. The engine implementation is based on JSPWiki and the Semantic Web engine of Hewlett-Packard

³⁵ <http://www.apps.ag-nbi.de/makna/>

Jena³⁶. Thus, Makna preserves the advantages provided by conventional Wiki technology (JSPWiki).

Makna (Dello, Simperl, & Tolksdorf, 2006) provides separate administration and user interface. The administration interface offers the following ontology management and configuration features:

Specify the ontology/ontologies used within the system.

Import external RDF data.

Define shortcuts for a more comfortable usage of ontological primitives.

Configure inference engines and persistent storage systems.

The user interface embeds facilities for creating and using Semantic Web information on the basis of the imported ontologies:

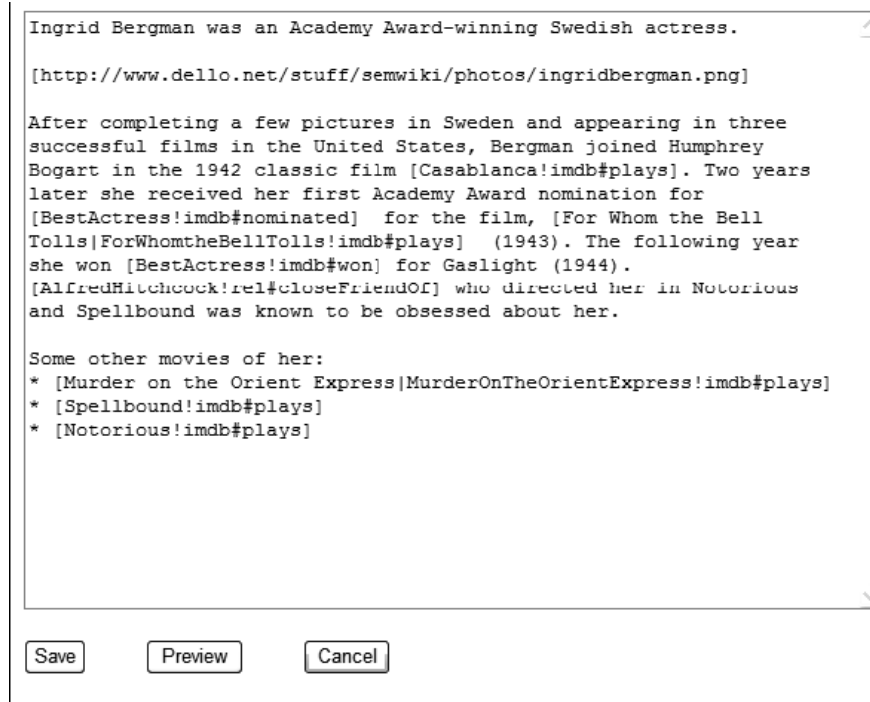
Refer to ontological primitives for annotating Wiki content or defining link types

Wiki users are able to create semantic content (in form of RDF statements referencing pre-configured ontologies) in the classical Wiki manner. They are provided with an extended Wiki syntax (Figure 2.15) and with assistant tools (e.g., predicate assistant, see Figure 2.16) simplifying the interface to the ontologies employed. Further on, users can create, modify and delete RDF statements associated with Wiki pages.

Consistency of the semantic model

Makna ensures the consistency of the semantic model by rejecting and notifying the user to correct his input, if an invalid statement is submitted.

³⁶ <http://jena.sourceforge.net/>



Ingrid Bergman was an Academy Award-winning Swedish actress.

[http://www.dello.net/stuff/semwiki/photos/ingridbergman.png]

After completing a few pictures in Sweden and appearing in three successful films in the United States, Bergman joined Humphrey Bogart in the 1942 classic film [Casablanca!imdb#plays]. Two years later she received her first Academy Award nomination for [BestActress!imdb#nominated] for the film, [For Whom the Bell Tolls|ForWhomtheBellTolls!imdb#plays] (1943). The following year she won [BestActress!imdb#won] for Gaslight (1944). [AlfredHitchcock!rel#closeFriendOf] who directed her in Notorious and Spellbound was known to be obsessed about her.

Some other movies of her:

- * [Murder on the Orient Express|MurderOnTheOrientExpress!imdb#plays]
- * [Spellbound!imdb#plays]
- * [Notorious!imdb#plays]

Save Preview Cancel

Figure 2.15: Makna syntax



Predicate Assistant

Just type anything belonging to the predicate (name,comment,domain,...) and a magical list will appear :-)

Name of predicate: mail

Start typing

- foaf#mbox
- foaf#mbox_sha1sum

Figure 2.16: Makna predicate assistant

Formulate and execute content- and structure-based queries

Makna implements a search interface resorting to form-based search patterns, which allows users to use the ontology in formulating structure-based queries and the underlying inference engines for enabling semantic search. Makna provides several templates for formulating typical content- and structure-based query patterns.

Browse the Wiki contents on a content- or structure basis

When a Wiki page is called the Wiki engine extracts a sub-graph of the semantic model, which contains all statements having the current page either as their subject or their object.

The Makna interface is shown in Figure 2.17. The navigation block on the right side of the screen consists of two parts: the summary of the semantic relations of the

current page (allows the user to quickly navigate to a related topic) and the list of the prepared search requests for related resources (for each property of a page a search link is provided to find other resources with the same property).

Export semantically represented data as RDF or N3



Figure 2.17: Makna user interface

3.3.7 OntoWiki

OntoWiki (Auer, Dietzold, & Riechert, 2006) aims at providing support for agile, distributed knowledge engineering scenarios. OntoWiki facilitates the visual presentation of a knowledge base as an information map, with different views on instance data. It enables intuitive authoring of semantic content, with an inline editing mode for editing RDF content, similar to WYSIWYG for text documents. It fosters social collaboration aspects by keeping track of changes, allowing to comment and discuss every single part of a knowledge base, enabling to rate and measure the popularity of content and honoring the activity of users. OntoWiki enhances the browsing and retrieval by offering semantic enhanced search strategies. Its features are described in the following paragraphs.

Visual representation of semantic content

In order to provide an interface as intuitive as possible OntoWiki regards knowledge bases as “information maps”. Each node at the information map, i.e., each RDF resource, is represented as a Web accessible page and interlinked to related digital resources. These Web pages representing nodes in the information map are divided into three parts: a left sidebar, a main content section and a right sidebar (Figure 2.18). The left sidebar gives the opportunity to the user to select content to be displayed in the main content section. The available selections are the set of available knowledge bases, a class hierarchy browser and a full-text search. The right sidebar offers tools and complementary information specific to the selected content.

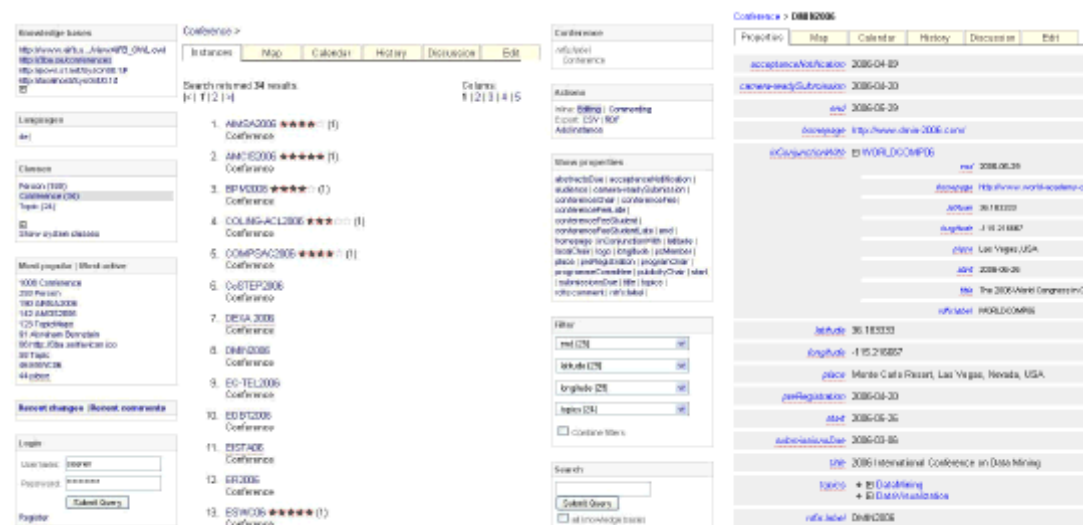


Figure 2.18: OntoWiki user interface

Different views on instance data

OntoWiki facilitates domain specific or generic views on instance data. Domain specific views have to be implemented as plug-ins. Generic views provide visual representations of instance data according to certain property values. Map and calendar views are currently implemented.

Editing modes

OntoWiki supports two complementary edit strategies for the knowledge base:

Inline editing: the smallest possible information chunks (i.e., statements) presented in the OntoWiki user interface are editable for users.

View editing: common combinations of information (such as an instance of a distinct class) are editable in one single step.

Additionally, OntoWiki provides the following editing widgets:

- **Statements:** allows editing of subject, predicate, and object.
- **Nodes:** edit literals or resources.
- **Resources:** select and search from/for existing resources.
- **Literals:** literal data in conjunction with a data type or a language identifier.
- **Widgets for specific literal data types:** e.g., dates, HTML fragments.
- **File widget:** allows uploading of files to the OntoWiki system.

Concept identification and reuse

OntoWiki uses the AJAX technology to interactively propose already defined concepts while the user types in new information to be added to the knowledge base. To realize this interactive search, all URI references and literals are indexed for full-text searches in the statement repository.

Social collaboration

Social collaboration within OntoWiki is supported by:

Change tracking: All changes are tracked. OntoWiki enables the review of changes and these can be narrowed down to a specific context (e.g., changes on a specific instance or made by a distinct user). Additionally, users can subscribe to get the most recent changes by email or RSS/Atom feeds.

Commenting: All statements may be commented. Small icons attached to an object of a statement indicate that such comments exist. Placing the mouse pointer on such an icon will immediately show up a tool tip with the most recent comments; clicking on the icon will display all comments.

Rating: OntoWiki allows to rate instances. Special annotation properties allow the creation of rating categories with respect to a certain class. Instances of the class can then be rated according to these categories.

Popularity: All accesses to the knowledge base are logged thus allowing to arrange views on the content based on popularity. The popularity of content can be measured with respect to a certain knowledge base or fragment of it.

Activity/Provenance: The system tracks who contributed what. This includes contributions to the ontology schema, additions of instance data or commenting.

Semantic search

Facet-based browsing: To enable users to select objects according to certain facets, all property values (facets) of a set of selected instances are analyzed. If for a certain property the instances have only a limited set of values, those values are offered to restrict the instance selection further.

Semantically enhanced full-text search: OntoWiki provides a full-text search for keywords occurring in literal property values. The results are grouped by instances and are ordered by frequency of occurrence of the search string. Search results may be filtered to contain only individuals which are instances of a distinct class or which are described by the literal only in conjunction with a distinct property.

3.3.8 OpenRecord

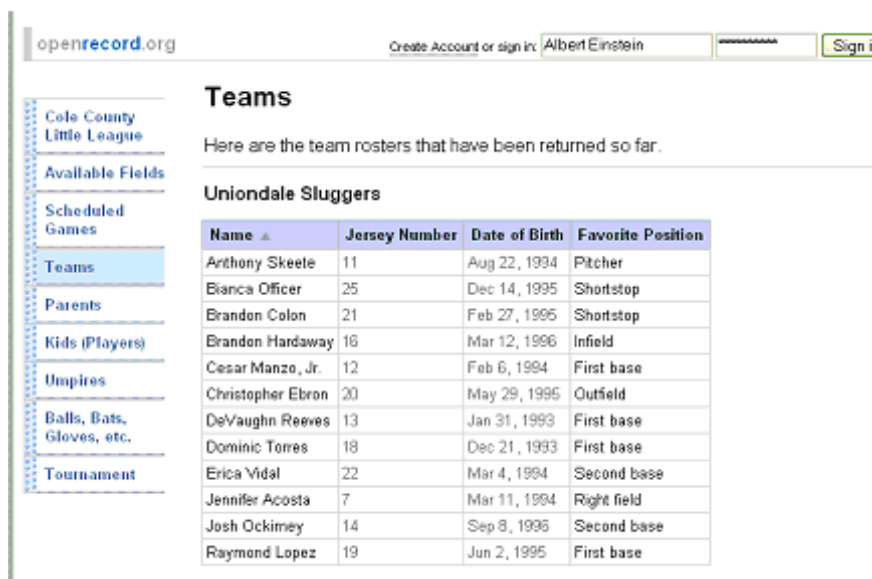
OpenRecord³⁷ is similar to a Wiki, but with some database features added in. In OpenRecord the content is organized as a database of items, or records. Each page on an OpenRecord site can query the database to get some set of items, and those items can be displayed in an editable table, or in an outline format, or in other formats. In time, OpenRecord could grow to incorporate simple spreadsheet features, as well as interactive charting and graphing features, and OLAP³⁸ and pivot table features. OpenRecord is still at an early stage of development. OpenRecord's interface is shown in Figure 2.19. The project goals include:

- Enable a workgroup to work in an unusually open and transparent style.
- Facilitate sharing and collaboration.
- Facilitate tracking of goals, tasks, and work accomplished.
- Facilitate comparison and prioritization.

³⁷ <http://openrecord.org/>

³⁸ On Line Analytical Processing

- Facilitate categorization and structuring of content.
- Make it easy to create new content without first planning how the content will be organized.
- Make it easy to gradually add structure and organization to the content.
- Make it easy to re-organize content in different ways, and have multiple concurrent organization schemes.
- Discourage the growth of a jumbled jungle of rambling discussion.
- Offer features geared towards collecting facts and figures.
- Discourage long pure-text discussion threads.
- Facilitate the creation of valuable reference material.



The screenshot shows the OpenRecord.org website. On the left is a sidebar with a vertical list of links: Cole County Little League, Available Fields, Scheduled Games, Teams (highlighted), Parents, Kids (Players), Umpires, Balls, Bats, Gloves, etc., and Tournament. The main content area is titled 'Teams' and includes a sub-header 'Uniondale Sluggers'. Below this is a table with four columns: Name, Jersey Number, Date of Birth, and Favorite Position. The table lists 13 players with their respective details.

Name	Jersey Number	Date of Birth	Favorite Position
Anthony Skeete	11	Aug 22, 1994	Pitcher
Bianca Officer	25	Dec 14, 1995	Shortstop
Brandon Colon	21	Feb 27, 1995	Shortstop
Brandon Hardaway	16	Mar 12, 1996	Infield
Cesar Manzo, Jr.	12	Feb 6, 1994	First base
Christopher Ebron	20	May 29, 1995	Outfield
DeVaughn Reeves	13	Jan 31, 1993	First base
Dominic Torres	18	Dec 21, 1993	First base
Erica Vidal	22	Mar 4, 1994	Second base
Jennifer Acosta	7	Mar 11, 1994	Right field
Josh Ockimey	14	Sep 8, 1995	Second base
Raymond Lopez	19	Jun 2, 1995	First base

Figure 2.19: OpenRecord user interface

3.3.9 Platypus Wiki

Platypus Wiki (Ademar Aguiar & David, 2005) offers a simple user interface to create Wiki pages including metadata. It uses RDF, RDF Schema and OWL to manage the metadata and create ontologies. All pages are stored in an HTML file with metadata in RDF.

The Platypus Wiki interface is shown in Figure 2.20. The current resource in a page is described by three columns. The first column shows all RDF statements (linkable) that have the current resource as the object. In the central column the main content is presented, which

describes the concept represented by the URI of the current resource. The last column contains all statements, in which the current resource is the subject. Platypus Wiki includes the features described below.



Figure 2.20: Platypus Wiki user interface

Metadata aggregation

Where to find the RDF metadata to merge is specified for every requested Wiki page, enabling metadata aggregation.

Links as RDF properties

In Platypus, Wiki RDF statements construct a directed labeled graph. A node is an RDF resource and a link is an RDF property. When the user clicks on a resource, it becomes the current resource and its RDF metadata is used to construct the navigation and presentation.

Editing

In Platypus Wiki, users can edit the content in a similar way to a normal Wiki. In addition users are able to edit the RDF metadata.

Automatic linking between pages

Platypus Wiki offers the possibility to create “site links”, “namespace links” and “page links”. A “site link” consists of one or more words and a URL. The engine replaces every word specified as a site link with the URL given. Users do not have to specify any links in a page but instead simply write the content, and when the page is published it will be

automatically enriched with site links. If in a particular namespace or page, a word indicated as a site link must have a different URL, “namespace links” or “page links” can be specified. These links behave like site links, but their scope is limited to the current namespace or a particular page.

Content and metadata aggregation

Platypus enables interleaving content and metadata from other Platypus Wikis. For each page a list of URLs is specified from which content is gathered and another list of URLs from which metadata is gathered.

Monitoring user activity

Platypus ranks the results of queries or elements in lists displayed to users. It also gathers explicit ratings for each page from users. Additionally, monitoring which navigation paths are mostly followed is useful in proposing navigational aids and correlating Wiki pages.

3.3.10 Rhizome

Rhizome (Souzis, 2005) is a Wiki-like content management and delivery system that exposes the entire site content, structure, and metadata as editable RDF.

An overview of the features that Rhizome offers is given below.

Custom text formats

Rhizome uses custom text formats in order to facilitate the creation of semantic content:

ZML is a plain text formatting language similar to those found in Wikis. Users, via *ZML*, can create ad hoc structural elements mapped to XML elements and use special formatting conversions to indicate semantic intent.

RxML is a simple alternative format for RDF that aims to aid novice users author and edit RDF metadata.

Access of RDF data stores as a virtual XML document object model

Rhizome Wiki runs on top of Racoon, which is a simple application server using an RDF model for its data store. Racoon uses *RXPath* to translate requests and command line arguments to RDF resources. This allows users to access RDF data stores as a virtual XML

document object model and query them using XPath. Other XML technologies that rely on XPath, such as Schematron, can easily be adopted.

Informal document conversion

This feature is shown in Figure 2.21, which shows a page created in Rhizome Wiki. The figure also illustrates how structured content is created, transformed into and stored as RDF. Rhizome allows transforming informal documents like FAQs into semantically meaningful resources.

Content, metadata and site structure stored as RDF

When a user saves a page of ZML content, Rhizome Wiki converts the ZML into RDF via a process called shredding, before saving it in its data store. Users can edit any of this RDF directly, thus enabling them to change the Wiki application's behavior and structure.

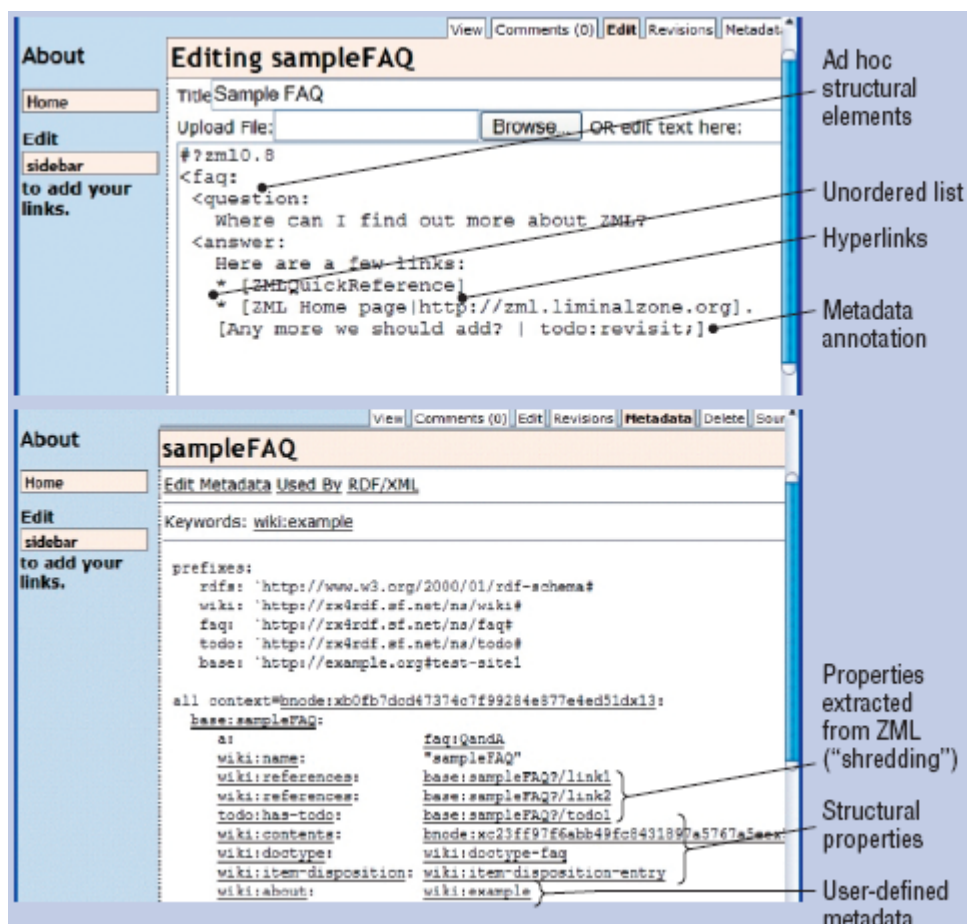


Figure 2.21: Edit mode and metadata view of the Rhizome Wiki user interface

3.3.11 Semantic MediaWiki

Semantic MediaWiki (Krötzsch, Vrandečić, & Völkel, 2006) is an extension of MediaWiki – a widely used Wiki-engine that also powers Wikipedia. In Semantic MediaWiki, each article (Wiki page) describes a concept. Semantic statements can be made, by freely assigning link types to hyperlinks. Additionally, attributes of any concept can be specified in a syntax similar to the link syntax. Instead of referring to another concept, they simply refer to a value.

One of its main targets is to establish the semantic wikipedia. In order to accomplish its objective, Semantic MediaWiki is also available in multiple languages and is designed to easily support further localization, since the most important usage scenario for Semantic MediaWiki are the different versions of Wikipedia.

Semantic MediaWiki addresses core problems of current Wikis, like:

Consistency of content: ensure that information is consistent in different parts of the system.

Accessing knowledge: easily find the right information, bypassing the information overload problem of many large Wikis.

Reusing knowledge: support the reuse of knowledge in separate applications.

Semantic MediaWiki provides the key features described below.

Annotating pages

The collection of semantic data in Semantic MediaWiki is achieved by allowing users to add notes to the article Wiki-text via a special markup. Every article corresponds to exactly one ontological element and every annotation in an article makes statements about this single article. Most of these annotations correspond to simple ABox statements in OWL DL. The following types of annotations are available in Semantic MediaWiki:

Categories are a simple form of annotation that allows users to classify pages. Semantic MediaWiki endows them with formal interpretation as OWL classes.

Relations describe relationships between two articles by assigning annotations to existing links.

Attributes allow users to specify relationships of articles to items that are not articles.

Querying and searching

Users can search for articles using a simple query language that was developed based on the known syntax of the Wiki and which is identical with the syntax used for specifying an annotation.

Dynamic pages

The query functionality of Semantic MediaWiki can be used to embed dynamic content into pages. Moreover the query syntax involves statements for displaying further properties of the retrieved results and for modifying the appearance within the page.

Reusing existing ontologies

Semantic MediaWiki is compatible with OWL DL knowledge model, which means that it is feasible to use existing ontologies within the Wiki. This is possible in two ways, via:

Ontology import, which is a feature that allows creating and modifying pages in the Wiki to represent the relationships that are given in some existing OWL DL document.

Vocabulary reuse, which allows users to map Wiki pages to elements of existing ontologies.

External reuse in practice

OWL/RDF export gives the possibility of external reuse of Wiki data. Some Semantic Web tools have been tested with the RDF output. Semantic MediaWiki works smoothly together with some applications like FOAF Explorer, the Tabulator RDF browser or the Piggy Bank RDF browser extension. Moreover an externally hosted SPARQL³⁹ querying service is provided. The system is based on the RDF Joseki server and is synchronized with the semantic content of the Wiki.

A typical Semantic MediaWiki screen is shown in Figure 2.22, showing the ontoworld.org Wiki – a Wiki for the Semantic Web community.

³⁹ <http://www.w3.org/TR/rdf-sparql-query/>

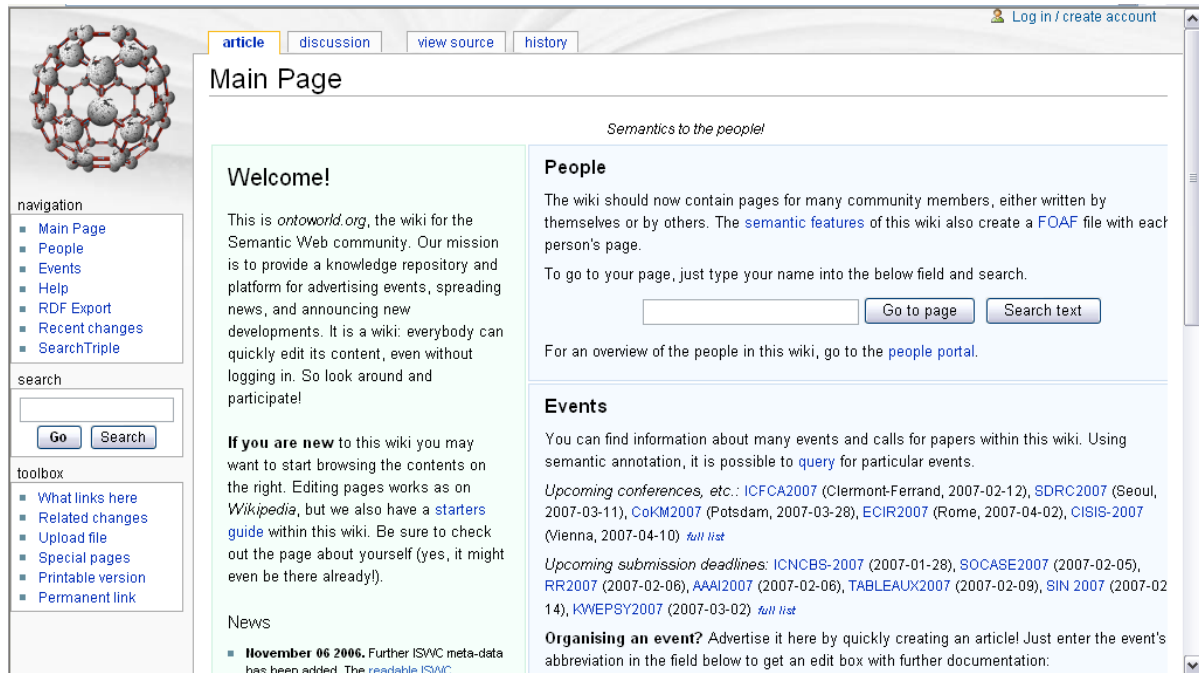


Figure 2.22: Semantic MediaWiki user interface

3.3.12 SemperWiki

SemperWiki (Oren, 2005) is a breed between semantic wikis and personal Wikis and serves as a personal information management system. It combines the ease of use which is characteristic of personal Wikis with the improved retrieval and navigation functionalities of semantic wikis. It can be used as a stand-alone semantic information system or as user interface of a semantic desktop system.

The user interface is depicted in Figure 2.23. On the left side users can edit pages and on the right side there is a navigation bar, which allows users to navigate through the Wiki contents. A page may consist of normal text, links to other pages and websites and semantic annotations.

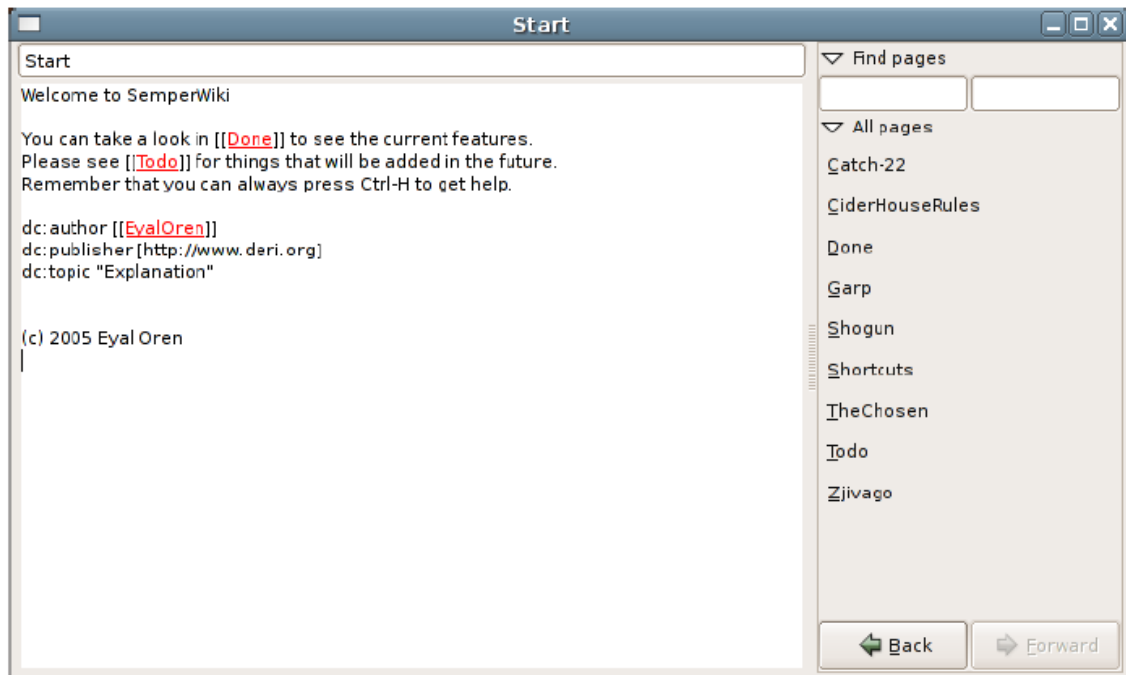


Figure 2.23: SemperWiki user interface

SemperWiki includes the features described below.

Information stored in RDF

All information in SemperWiki is stored in RDF. Each page is identified by a URI, which is formed by prefixing its title with the base URI of the Wiki, defined by the user.

Syntax

For the semantic annotations a simple syntax is used. A statement is used on a line by itself and consists of a predicate followed by an object. Predicates are resources and can be written as their full URIs, with prefix notation, or as Wikilinks.

Navigation

SemperWiki offers various ways to navigate to a page. Someone can jump to a page by:

- clicking on any Wikilink;
- typing the name of the page directly;
- using the history buttons; and
- by typing a predicate and/or object in the find section.

Dynamic Sidebar

The links on the sidebar are generated automatically based on the available semantic information in the system. Most specific pages are shown first and the less corresponding ones last.

Instant Response

All changes to all pages in SemperWiki are saved instantly, without any user interaction. This means that while the user is searching or typing in the pages, information is updated without any need to save or refresh anything on the page.

3.3.13 SweetWiki

SweetWiki (Michel Buffa & Gandon, 2006) relies on the CORESE semantic search engine for querying and reasoning (Corby & Dieng, 2004) and on SEWESE, its associated web server extension that provides API and JSP tags to implement ontology-based interfaces and a set of generic functionalities like security management, ontology editors, web application life cycle, etc.

Some of the main characteristics of SweetWiki are described below.

Ontologies

SweetWiki makes use of two ontologies:

An ontology of the Wiki structure that provides metadata based on an ontology of Wikis.

An ontology of the topics where each Wiki page addresses one or more topics.

The ontology of the Wiki structure is maintained by the developers of the Wiki. The domain ontology is enriched directly by the users and may be restructured by administrators or volunteers of the site to improve the navigation and querying capabilities.

Tagging

SweetWiki uses a standard WYSIWYG editor which is extended to directly support semantic annotations. As shown in Figure 2.24, a user has the possibility to enter some keywords in an AJAX-powered text-field while editing a page. As the user types, an auto-completion mechanism proposes existing keywords by issuing SPARQL queries to the Semantic Web server in order to identify existing concepts with compatible labels. Later on community experts may reposition those keywords in the ontology, edit them etc.

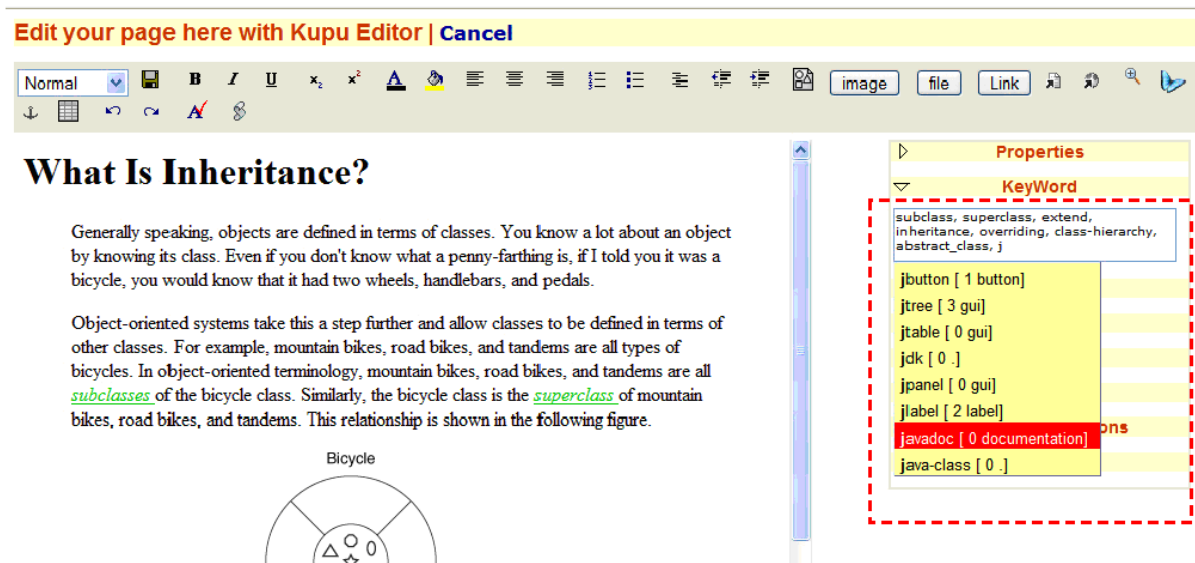


Figure 2.24: Tags suggested in SweetWiki as the user enters keywords

Faceted navigation

The CORESE engine is used to generate faceted navigation widgets: the semantics of the tags is used to derive related topics and query the engine on similar pages using SPARQL queries (Figure 2.25).

Ontology editor

Supervising tools are integrated in SweetWiki relying on SEWESE and are used to monitor the Wiki's activity. Moreover, SweetWiki also reuses web-based editors available in SEWESE in order to maintain and re-engineer the ontologies.

Sweet Wiki Edit this page | login | password | Connect | Register | Semantic Web Enabled Technology Wiki

Workspace What Is Inheritance?

- Main
- Users
- All users
- Sand Box
- Search

Generally speaking, objects are defined in terms of classes. You know a lot about an object by knowing its class. Even if you don't know what a penny-farthing is, if I told you it was a bicycle, you would know that it had two wheels, handlebars, and pedals.

Object-oriented systems take this a step further and allow classes to be defined in terms of other classes. For example, mountain bikes, road bikes, and tandems are all types of bicycles. In object-oriented terminology, mountain bikes, road bikes, and tandems are all **subclasses** of the bicycle class. Similarly, the bicycle class is the **superclass** of mountain bikes, road bikes, and tandems. This relationship is shown in the following figure.

Bicycle

may partially implement the behavior, but much of the class is undefined and unimplemented. Other programmers fill in the details with specialized subclasses.

Keywords

- inheritance(1)
- superclass(1)
- extend(1)
- class-hierarchy(1)
- overriding(1)
- abstract_class(1)
- subclass(1)

See Also

Tags' informations

Category : inheritance, abstraction,

Related tags : subclass, multiple_inheritance, abstract_method, extend,

Page informations

- Author : admin
- Web : Main
- Last changes : 2006-06-11

Figure 2.25: Faceted navigation in SweetWiki

3.3.14 WikSAR

WikSAR (Aum Mueller & Auer, 2005) (Semantic Authoring and Retrieval within a Wiki) facilitates semantic authoring and provides the user with context aware navigation, interactive graph visualization of the emerging ontology, as well as semantic retrieval. Embedding queries into Wiki pages creates views on the information space. Desktop integration includes accessing dates (e.g., reminders) entered in the Wiki via local calendar applications, maintaining bookmarks, and collecting web quotes within the Wiki.

The WikSAR interface consists of a Wiki page and the optional interactive graph visualization. A Wiki page is divided into three parts (Figure 2.26): the form for editing the text, the rendered Wiki text above, and the sidebar on the right containing context-dependent links, constructed from semantic information present on other Wiki pages.

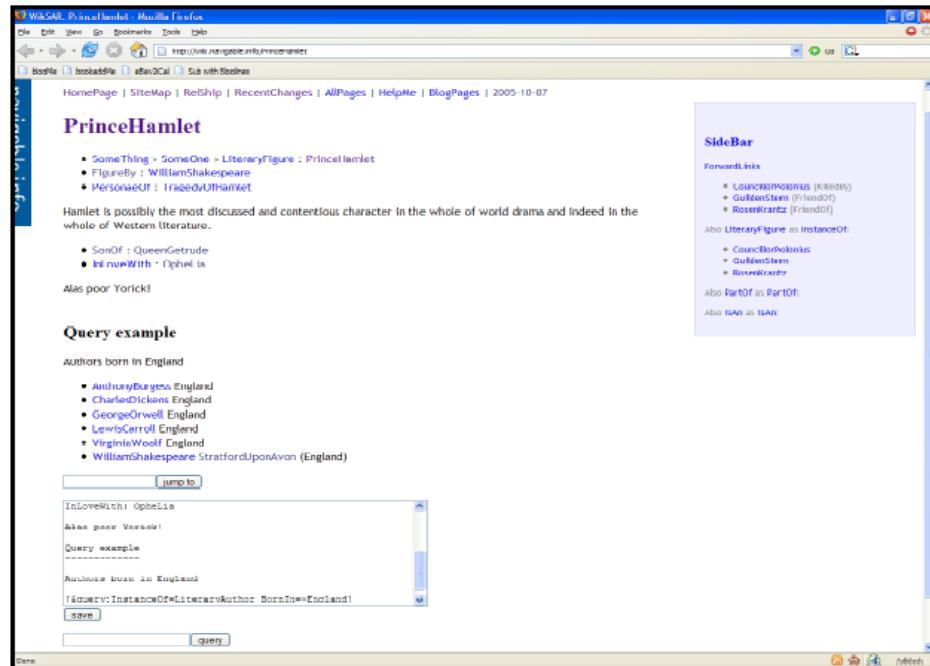


Figure 2.26: WikSAR user interface

In more detail WikSAR offers the features described below.

Semantic authoring

WikSAR uses the WikiWord or CamelCase syntax. WikiWords are not only used to create hyperlinks to other Wiki pages: they are interpreted either as subject, predicate, or object in Semantic Web statements, i.e., RDF triples. The page name of a Wiki page denotes the subject of statements embedded in the Wiki text. An editing assistance can be integrated in the Wiki by suggesting already used WikiWords or vocabulary from external reference ontologies.

Context-aware semantic navigation

The entered statements are used to create links to related pages depending on the current context. Firstly, breadcrumbs inform the user about his position in the Wiki showing the path back to the root of the site or concept. Secondly, the sidebar shows pages or concepts related to the current concept including their type of relationship. Additionally, the semantic annotations are used to automatically generate a class hierarchy or a complete map of the ontology as labeled graph, i.e. a typed site map.

Semantic retrieval

Triples stored in WikSAR are available for semantic queries. The Wiki space can be queried to return distinct concepts (pages). The current query syntax allows filtering by specific predicate-object combinations allowing equality, quantitative comparisons, and regular expressions as operator.

Semantic views and query chaining

WikSAR accepts a variety of proprietary commands embeddable in Wiki pages to generate and include content gathered from all available data. Likewise, queries need not only to produce search results available temporarily to the user – queries can be embedded persistently within Wiki pages.

Desktop integration

WikSAR publishes dates entered in the Wiki as remote calendar entries in the iCalendar format. Such entries can be then imported or subscribed to by desktop calendar applications and furthermore can trigger reminders. Additionally, WikSAR uses “bookmarklets” to put selected pieces of text and/or the URI of a resource onto the Wiki.

Interactive graph visualization and navigation

A quite novel feature of WikSAR is the integration of an interactive graphical representation within the Wiki; navigating through the Wiki space is possible in either the Wiki or the graph, changing focus simultaneously in both views (Figure 2.27).

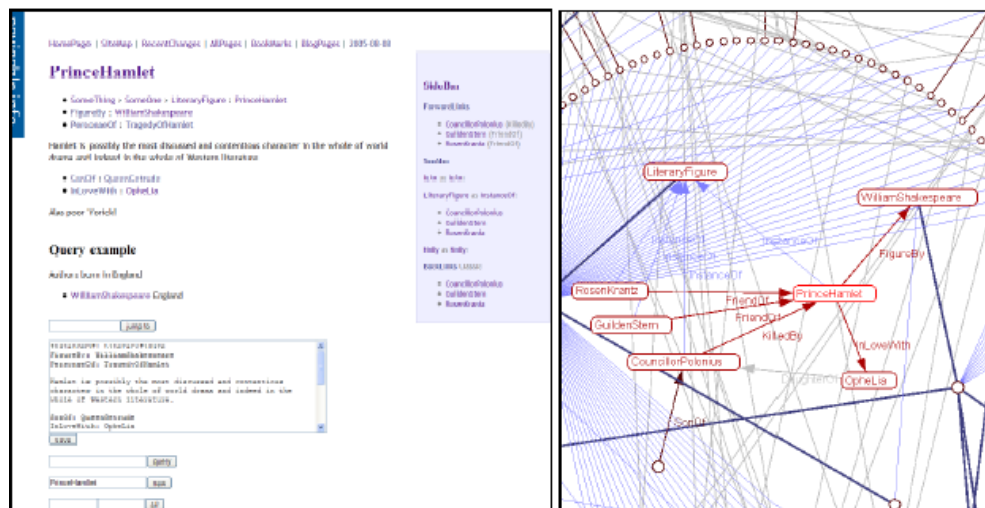


Figure 2.27: WikSAR with interactive graph switched on

3.3.15 Comparison of surveyed implementations

The most important features of the surveyed semantic wikis are summarized in Table 2.2. The features have been classified in five types of activities (authoring, navigation, retrieval, reuse and social collaboration). These types of activities cover the whole operational lifecycle of a semantic wiki. Several conclusions can be derived while observing the table:

Common features

Most of the semantic wikis share some common features. Context-aware navigation and full-text search are implemented in almost every semantic wiki engine. A very important characteristic of most of them is that they provide two or three alternative ways for retrieval. Some of them strive to provide alternative ways for navigation, as well. In addition, a clear trend can be observed about the implementation of the following features:

Auto-completion.

Inference.

Queries supported by query languages.

Ontology export.

Change tracking.

Differences

The surveyed semantic wikis take different approaches concerning authoring, navigation and retrieval. A few of them provide WYSIWYG editors complementing the functionality of regular text editors. Finally, there is a significant lack of means for social collaboration besides OntoWiki.

Completeness of supported features

From Table 2.2, it is obvious that the most complete semantic wiki engines in terms of supported features are the following (starting with the most complete engine):

1. OntoWiki.
2. COW.
3. Semantic MediaWiki.

4. Makna.
5. IkeWiki.

	COW	IkeWiki	Kaukolu	Makna	OntoWiki	OpenRecord	Platypus Wiki	Rhizome	Semantic MediaWiki	SemperWiki	SweetWiki	WikSAR
Authoring												
ACID transactions	●	○	○	●	○	●	○	○	●	○	○	○
auto-completion	○	●	●	●	●	○	○	○	○	○	●	●
ontology editor	●	●	○	○	●	○	○	○	○	○	●	○
WYSIWYG editor	○	●	○	○	●	●	○	○	○	○	●	○
Navigation												
context-aware navigation	●	●	○	●	●	○	●	○	●	●	●	●
different views	○	○	○	○	●	○	○	●	○	○	○	●
faceted browsing	○	○	○	○	●	○	○	○	○	○	●	○
interactive graph visualization	○	○	○	○	○	○	○	○	○	○	○	●
ontology	●	●	○	○	●	○	○	○	○	○	○	○

	COW	IkeWiki	Kaukolu	Makna	OntoWiki	OpenRecord	Platypus Wiki	Rhizome	Semantic MediaWiki	SemperWiki	SweetWiki	WikSAR
browser												
Retrieval												
embedded query	●	○	○	○	○	○	○	○	●	○	○	●
full-text	●	●	●	●	●	●	●	●	●	●	●	●
inference	○	●	○	●	●	○	●	○	○	○	●	○
query templates	●	○	○	●	○	○	○	○	○	○	○	○
using query language	●	○	○	○	○	○	●	●	●	●	○	●
Reuse												
ontology export	●	○	●	●	○	○	○	●	●	○	○	○
ontology import	○	○	●	●	○	○	○	○	●	○	○	○
Social collaboration												
change tracking	○	●	○	○	●	●	○	●	●	○	○	○
commenting	○	○	●	○	●	○	○	●	○	○	○	○
popularity	○	○	○	○	●	○	○	○	○	○	○	○
rating	○	○	○	○	●	○	●	○	○	○	○	○

● = supported, ○ = not supported or not obvious from literature

Table 2.2: Overview of surveyed semantic wikis' features

3.4 semantic wikis in Software Engineering

In this section, the investigation of the potentials of the wiki technologies in the domain of software engineering is presented. First, a survey of the usage possibilities of conventional wikis in various software development activities is presented. Then, a discussion of new opportunities and challenges resulting from the emergent semantic wikis is presented.

3.4.1 Wikis in Software Engineering

3.4.1.1 A short history of Wikis in Software Engineering

The short history of Wikis is tightly coupled to the domain of software engineering. In fact, the invention of the Wiki took place in a software development context, when Ward Cunningham coined the term in 1995 for the website of the “Portland Pattern Repository”⁴⁰. This first ever Wiki is dedicated to collecting and discussing design patterns in software engineering.

After that, Wikis were increasingly adopted in further communities. Especially distributed teams in the open source community appreciated Wikis as a lightweight solution for collaborative knowledge exchange. In those highly dynamic environments, Wikis act as a flexible “glue” in the tool landscape, complementing mailing lists and configuration management tools.

The rising availability of stable open source Wiki engines also fostered the adoption of Wikis in company settings. Again, tech savvy development teams started to favour flexible Wiki technology against awkward traditional content management and groupware systems. Often, the rollout of Wiki software in those intranet settings started as a grassroots initiative without explicit management attention (M. Buffa, 2006) (TWiki.org, n.d.).

Throughout the time, software engineering has remained one of the dominant application areas of Wiki technology (Louridas, 2006). Wikis are often introduced by a small number of early adopters and then used by an increasingly amount of employees. Besides open source development teams and small companies, also big players such as SAP, Novell

and Yahoo are using Wikis internally (TWiki.org, n.d.). Shashi Seth from Google even claims that “This company runs on Wikis”⁴¹.

Currently, especially Wikis’ technical drawbacks are addressed by commercial vendors, e.g., poor usability support and low integration with other applications (like spreadsheets). Besides those general purpose “enterprise Wikis” offered by companies such as JotSpot or SocialText, some solutions focus on the software engineering domain. This includes integrating Wikis in IDEs (John, Jugel, S. Schmidt, & Wloka, 2005), source code documentation (Ademar Aguiar & David, 2005), bug tracking⁴² or in collaborative software development platforms⁴³.

3.4.1.2 Applications of Wikis in Software Engineering

The big success of Wikis and Wiki-based systems in the Internet and in open source communities resulted in a fast adoption of this technology in professional environments. Software development companies as well as research projects conducted many experiments to tap the potential of Wikis. Indeed the application of Wikis in the field of software engineering may vary from brainstorming to documentation over to bug reporting and might even include team coordination and collaboration. In the following, a presentation of promising examples of those applications is given.

Documentation: Central documentation practices often cause many problems for software developers and for project managers, as well. In general, the main challenge is to keep the information up-to-date, while change occurs. In addition, especially in large distributed projects with overlapping responsibilities, information quickly gets redundant and inconsistent. Wikis present a convenient platform to address both information up-to-datedness and consistency. Firstly, the easy-to-use update mechanism enables all stakeholders to update information as soon as changes occur. In such way the documentation process also gets distributed and simultaneous to the development and management

⁴⁰ <http://c2.com/cgi/wiki?WikiHistory>

⁴¹ <http://google.wikia.com/wiki/Goowiki>

⁴² <http://trac.edgewall.org/wiki/TracGuide>

⁴³ E.g., CodeBeamer (<http://www.codebeamer.com>)

activities. Secondly, the easy information linking in Wikis reduces inconsistencies by keeping the information in a central place (e.g., a Wiki page) and referencing it if required. WikiDoc (R. Wolf & Zhao, 2007) and XSDoc (Ademar Aguiar, Gabriel David, & Manuel Padilha, 2003), (Ademar Aguiar & David, 2005) are two concrete examples that clearly demonstrate the benefits of Wikis for software documentation. In WikiDoc java code documentation can be added via a Wiki interface – also from non-programmers. XSDoc enables the creation and annotation of software framework documents as well as the integration of different content types (text, models and source code). In (Ademar Aguiar et al., 2003) the authors present a usage example of XSDoc for producing part of a document for JUnit testing framework. Using Wikis as a documentation infrastructure might be inefficient, if paper-based high quality documentation is needed (e.g., user manual). In this case, the organic (due to the natural and non-predefined growth) structure of the Wiki documentation and the informality of the content might affect the quality of the document. Wiki documentation needs to be restructured, serialized and adapted to the target reader. However, Wikis can still serve as an important help for document writers. Bachmann and Merson (Bachmann & Merson, 1998) represented their lessons learned from using a Wiki-based tool for architecture documentation. They argue that the advantage of being able to create and maintain architecture documentation in a dynamic and collaborative way seems to outweigh any other disadvantages.

Bug tracking: Wikis have also been used to capture and manage bugs and issues of software systems, especially in distributed open source projects. In fact, bug tracking is a collaborative, a knowledge-intensive and an interdisciplinary task, which can perfectly be supported by Wikis. Again, the linking feature and the ease of posting are the key success factors for this Wiki application. System developers as well as users are able to post, comment, link, describe and collaborate in order to detect and fix issues. Trac⁴⁴ is an example of a Wiki-based bug-tracker that allows relating Wiki pages to issues and vice versa. Furthermore, the code of a project can be integrated as read-only documents.

Requirements and traceability management: Traceability enables to trace requirements back to the stakeholders or forth to the design concepts, source code and test cases (Dutoit &

Paech, 2003). Indeed, traceability is not excluded to requirements. Especially in large distributed projects where change occurs frequently, developers and other system stakeholders often need to trace artefacts and decisions to other artefacts, decisions, models and participants. It is often useful to trace a test case to the related components that should be tested. A design goal might be also traced to non-functional requirements. A management decision might be traced to a risk or to a design decision. Wikis offer an easy and intuitive way to associate “everything with everything”. The links between Wiki pages might be interpreted as associations between “data objects” (the pages). This makes information traceable to other information pieces across the project. Using XSDoc (Ademar Aguiar et al., 2003), in (Silveira, Faria, A. Aguiar, & Vidal, 2005) the authors present a Wiki-based requirements and traceability management approach especially for customizable software. The relationship between specific requirements and generic product characteristics is established mainly via configuration parameters and associated configuration questions. Linking and viewing facilities support traceability analysis.

Reuse support: Wikis can be considered as an easy to use platform for exchanging reusable knowledge and artefacts between and within software projects. Wiki support for reuse goes beyond its origin for supporting pattern reuse as stated before. Indeed the linking features enable the reuse of requirements specifications, documentations, testing instructions, lesson learned etc. Although conventional Wikis provide a general history tracking mechanism, this often does not supply the strong common need for elaborated configuration management mechanisms, which support reuse. Histories of a Wiki page present a linear sequential thread of changed content of the page. The creation and management of parallel branches, a basic feature of each configuration management system, is not supported by conventional Wikis.

Communication and collaboration: Wikis support both planned and unplanned communication in a distributed development context. A successful example of supporting planned communication is presented in (Mullick, Bass, Houda, Paulish, & Cataldo, 2006), where a Wiki is deployed as a management tool for meeting protocols in real life projects. Decker et al. (B. Decker, Ras, Rech, Klein, & Hoecht, 2005) state four main advantages of

⁴⁴ <http://trac.edgewall.org/>

Wikis for supporting collaboration and knowledge exchange in the course of software development projects:

- One place publishing: a central place for publishing information often simplifies the communication between participants. Moreover the content of the communication is stored locally and can be used for other reasons (planning, documentation etc.)
- Simple and safe communication: the rules of Wikis are easy, especially to software developers who are used to source code editing. Simple access right mechanisms make the communication more secure.
- Easy linking: this is a major strength of Wikis, especially when applied in the domain of software engineering. Indeed, a raising need in software projects is to be able to associate “everything with everything”. Especially in highly dynamic domains where changes occur frequently, software engineers often need to associate participants and artefacts (such as design models, communication objects or a change request) with each other.
- Description on demand: a user might link an object to a target, although the target does not exist at the “linking” time. Therefore it is possible to create the target object whenever it is needed. Linking becomes easier.

Agility support: Wikis provide a perfect tool support for agility in development projects. On the one hand, Wikis facilitate informal communication, which is a main success factor for agile methodologies. On the other hand, Wikis make up a flexible and quick mechanism of carrying out changes. Unlike classical methodologies that strive toward minimizing and controlling changes (e.g., through change control boards, strict change rules) agile methodologies consider changes as a main condition during and after development. Additionally, Wikis provide a convenient implementation of motivating participants – a core principal of agile practices. The developers’ motivation is higher if they personally propagate their knowledge and are responsible for it. Unlike central communication mechanisms with tight control that limits potential opportunities for expertise sharing and collaborative knowledge creation among distributed team members, in a Wiki everyone is responsible for his/her content. This increases the information quality and the motivation of the developers. As a matter of fact many recent studies have demonstrated the usefulness of Wikis in agile projects (Silveira et al., 2005), (Thomas Chau & Frank Maurer, 2005), (T. Chau & F.

Maurer, 2006). In (T. Chau & F. Maurer, 2006) the authors present the results of an exploratory case study. They argue for the need of knowledge sharing tools that support not only structured but also unstructured knowledge representation. Wiki-based informal knowledge authoring tools can be used for sharing content about problem understanding, instrumental, projective, social, expertise location and content navigation purposes. The authors also observed self-organized maintenance of the repository content among the ordinary repository users as a result of the open-edit nature of the Wiki-based repository.

In the following table, a recapitulation of the main advantages and issues of wikis' applications in the software engineering domain is shown.

Application	Main Advantages	Main Issues
Documentation	Instantly information update when change occurs Redundancy avoidance through linking capabilities	Serialisation and structuring of content
Bug tracking	Easy collaboration of different stakeholders Linking and referencing of other bugs	Integration of other applications and content types (e.g., IDE, configuration management)
Requirements and traceability management	Association of requirements to other information Traceable requirements	Lack of link semantics Lack of structured content
Reuse	Central knowledge repository Referencing and linking of content	Narrow support for non-textual content (e.g., diagrams, code etc.) Narrow versioning support

Communication and collaboration	One place publishing Simple and safe communication Easy linking Description on demand	No instant communication No possibility of central organisation
Agility support	Change support Motivation of participants	Narrow support for structured knowledge

Table 2.3: Applications of Wikis in software engineering

When talking about the advantages of Wikis in a software engineering setting, one has to talk about its alternatives. Since Wikis are a very flexible tool, they compete with a range of generic applications such as e-mail, instant messaging and groupware/CMS systems.

Compared to classical groupware and CMS systems, Wikis have the advantage of flexibility and easy access. The main success factors for introducing Wikis at SAP were easy accessibility, low entry barriers and to “convince the hackers”. Users are allowed to edit most pages while the versioning mechanism avoids vandalism – social protocols replace technical controls like access rights. Chau and Maurer (Thomas Chau & Frank Maurer, 2005) found out, that the paradigm of decentralized contributors adding unstructured knowledge improves knowledge sharing in contrast to structured knowledge provided by a centralized team. In opposite to instant messaging/chat, Wikis allow asynchronous communication and an evolution of knowledge at a central place, and thus complements (or may even partially replace) such classical group collaboration tools.

3.4.2 Semantic wikis in Software Engineering

While classical Wikis are well suited for working collaboratively on unstructured information, they are limited concerning fine-grained structured content. Obviously, conventional Wikis tender bounded opportunities for storing machine-interpretable knowledge and for answering structured queries.

Besides Wikis that specialize in dealing with a certain kind of structured data, semantic wikis offer a general approach for the acquisition and evolution of structured knowledge. In this section an analysis of specific opportunities of semantic wikis when applied in the

domain of software engineering is presented. Afterwards, a discussion of some concrete application scenarios that add real value to core software engineering processes is presented.

3.4.2.1 Opportunities of semantic wikis for Software Engineering

Semantic wikis entail various advantages for software engineering projects. They enable an incremental formalisation of the underlying knowledge across the various activities of software engineering. Thereby, they increment results from the underlying ontologies that can be extended and customized in the different contexts. Unlike a development or a collaboration infrastructure with a fixed scheme (e.g., a data base for bug tracking system where bugs must occur in a version), semantic wikis provide a simultaneously growing scheme and data. Obviously, a higher flexibility for project rules is supported.

In addition, linking capabilities of semantic wikis enable the association of various artefacts and system models. While these associations are untyped in classic Wikis they are typed in semantic wikis. In conventional Wikis the only semantics of the link between two pages is that these pages are “related to” each others. In real life, developers might want to define a design pattern that *traces* a non-functional requirement, a change request that *changes* a core feature, or a design concept that *implements* a requirement. Annotation of data and links between data in semantic wikis satisfy such developers’ needs. Thereby, rules might be defined based on these typed associations as well as the information pieces, which acquire types and semantic specifications by themselves. Conventional Wikis lack machine-understandable semantics. Thus structuring the Wiki content by machine reasoning is difficult. These semantics are especially valuable for linking the various data objects.

The acquired collaborative formal knowledge rather complements informal human communication than replacing it. Rule-based systems, reasoners and declarative query languages might be used for a semiautomatic processing of such knowledge and creation of new one. New content such as implicit associations between artefacts can be generated. Additionally, flexible and declarative model-checking and -validation mechanisms become easy to implement. When common ontologies are deployed, project knowledge can be much easier imported from and exported to other projects and repositories. This is obviously valuable for software development organisations and communities. Reuse and knowledge exchange becomes more technology, organisation, process and methodology independent. A

user-story in Extreme Programming (XP) can be considered, for example, as a use case instance in Rational Unified Process (RUP).

Finally, in a report on metadata management state of the art (Happel et al., 2007), there is a discussion of the advantages of annotations and metadata for system models and artefacts produced in software engineering. Ontology based semantic wikis perspicuously provide powerful and extensible annotation mechanisms that can be applied to system models and software artefacts when deployed in software development projects.

3.4.2.2 Example Scenarios

The advantages of semantic wikis as described before are quite generic and apply to a large number of possible application areas. Additionally, since mature and usable implementations of semantic wikis are still rare, there are no established applications in software engineering scenarios yet. However a short grasp of concrete application scenarios is provided by describing the application of semantic wikis in requirements engineering and SOA maintenance in the following.

Requirements engineering

Decker et al. present an application scenario of semantic wikis in requirements engineering (B. Decker et al., 2005). They augmented the Wiki with application and solution domain specific ontologies. This provides means to manage the organic growth implied by the Wiki approach. Annotating the knowledge created in requirements elicitation based on ontologies enabled Wikis to offer relevant recommendations. For instance, explicitly “telling” the Wiki that an object is an *actor* and that an actor *instantiates* a *use case*, allows Wikis to recommend possible actors. Moreover consistency checks (e.g., each actor should initiate at least one use case) and cross project offering based on the capture of the user context might add value to classical Wikis.

SOA maintenance

Service-oriented architectures (SOA) are complex systems consisting of a large number of heterogeneous artefacts and documents. Their documentation and maintenance involves diverse stakeholders such as developers and business experts. The situation of both rather technical (e.g., interfaces specifications) and textual documentation leads to scattering of SOA information into different information spaces, which causes substantial management effort.

Semantic wikis can serve as a foundation for a lightweight solution to document and maintain service-oriented architectures. They allow for informal documentation with low entry barriers, while providing means for specifying formal relations among concepts where needed. Additional information can be derived by combining the asserted knowledge with existing specification documents taken from external service repositories.

The concrete idea is importing the structure of technical artefacts such as interface descriptions into a knowledge base that serves as the basic structure for browsing the service oriented architecture, providing a better overview of the SOA for developers and business experts. Through semantic connections between services, business objects and domain concepts, it is possible to gain a quick overview, e.g., which services are using a certain business object. Since this enables an easy access to explore the list of available services, it might also foster the reuse of existing services.

Semantic wikis enable an informal style of documentation while also supporting to incorporate machine-interpretable knowledge about technical descriptions. Thus, they allow collecting relevant information about a service-oriented architecture at a central place that has so far been maintained separately.

3.4.2.3 Implementations

There are attempts to use semantic wikis in software engineering. The RISE (Reuse in Software Engineering) project (B. Decker et al., 2005) introduces the Riki Wiki combined with the Wikitology paradigm (B. Decker et al., 2005), (Klein, Hoecht, & B. Decker, 2005). With Wikitology, it is possible to derive semi-automatically an ontology needed for information retrieval from the pages in the Riki itself. Such a Wikitology automatically updates “itself”, i.e., it reflects the changes of contained knowledge, changed views of the users accounts, new projects, customers, rules, trends, and ideas inside the Wiki. By considering the Riki as the ontology, the ontology will be always collectively edited up-to-date and will automatically reflect all changes.

Hyena (Rauschmayer, 2005) aims at providing a platform that makes RDF’s advantages available to software engineering without diverging too far from the concepts and tools that are familiar in that domain. It provides editing, GUI and publishing services for implementing editors of RDF-encoded custom models and it is implemented as an Eclipse plug-in. It includes some built-in ontologies for many basic software engineering tasks.

These ontologies have themselves become part of the platform, because they provide services that will be useful for other ontologies.

4 APPROACH

This thesis strives to solve *knowledge management problems in software development* by using *social semantic desktop* technologies. Its approach yields a knowledge management system that aims to assist developers in their daily work. In this chapter the approach of the thesis is described. In section 4.1 the problems related to managing knowledge in software development (in current state of practice) are described. Then, in section 4.2 the knowledge management lifecycle with regard to the thesis' approach is discussed. The discussion in section 4.2 leads to research challenges which are depicted in section 4.3. Finally, the proposed system is introduced – namely KnowBench and the contributions of the thesis are summarized.

4.1 Introduction

Working in distributed teams rapidly gains increasing importance for professional software development – for instance, in distributed or virtual organizations, in nearshore / offshore settings, when working simultaneously in the software organization and with the customer, when cooperating with a software component supplier, or also in open-source projects. In such settings, communication and diffusion of information becomes more difficult, and sharing the expertise for efficient production of high-quality software requires more time and effort. The problem lies not only in the geographic distribution of people, but also in often associated phenomena like different working times, different prior experience or working culture, different technical languages or organization internal slang, different working / programming styles, or psychological effects like a stronger “not invented here” attitude when working with external colleagues.

But, apparently, not only the emergence of communication bottlenecks through far-distance collaboration is faced; the situation is also made worse because there is obviously a growing need for extensive communication and knowledge sharing. Clearly, the “construction by configuration” approach is becoming the predominant paradigm in software development in the next years (Component Frameworks and Service Oriented Architecture are two examples for this approach). This means that reuse happens at a higher level of abstraction and complexity than earlier; correct integration of a complex software

component, effective working with a powerful software framework, successfully employing a design pattern – all this requires significant background and experience knowledge about the used software artefact, sometimes even context knowledge, e.g., about the application domain. Further, since the globalization of software production enables new alliances and more effective software creation value chains, the turn-around time for knowledge is decreasing, i.e., new versions come faster, and the experience with old version ages rapidly.

Lindvall & Rus (Lindvall & Rus, 2002) identify the following major fields of knowledge which are critical for the success of software projects: (1) Knowledge about technologies; (2) Domain knowledge; (3) Knowledge about local policies and practices; and (4) “who knows what” knowledge. The authors also emphasize the importance of team collaboration and knowledge sharing in distributed groups.

Altogether, effective experience sharing and efficient re-learning of new features becomes key to success: In a situation where the reuse of all kind of artefacts will be the focal point in software development, not only the semantic description of the functionalities or characteristics of an artefact, but also the suitable methods to understand and make appropriate use of the artefact will be the key factor for achieving efficiency and ensuring quality of software production.

On the other hand, *existing approaches for sharing software development knowledge focus mostly on the implementation of the Experience Factory concept in the form of knowledge repositories, which has been proved as suitable only for large software companies*, since small- and medium-sized ones usually cannot afford to create centralized organisational units for organisational learning. Moreover, for a distributed and for open-source software development, such a centralised approach is certainly not an optimal solution. Finally, *although the Experience Factory is a useful solution for sharing general projects’ experience, even large companies did not achieve the increase in the quality of software by introducing Experience Factories*. Indeed, Experience Factory embodies the assumption that, to share and exploit knowledge, it is necessary to implement a process of knowledge-extraction-and-refinement, whose aim is to eliminate all subjective and contextual aspects of knowledge, and create an objective and general representation that can then be reused by other people in a variety of situations (like an experience package). However, *the software implementation knowledge is too divergent and implicit so that by mapping it into predefined packages a part of its characteristics that can be important for*

reuse is lost. For example, the knowledge about how to use a class encompasses the web site the developer visited, the input he posted in the wiki, the errors he made by instantiating variables, etc., in other words, any situation (i.e. working context) in which a user was involved in order to clarify (explicitly or implicitly) the meaning of that class. However, **it is difficult to define a universal representation of knowledge**, since not only the last action a user performed in resolving a problem, but moreover all actions he performed in the context of resolving the problem are relevant for the knowledge sharing. Indeed, due to very individual perception of knowledge, it might be the case that the information from a web page was not useful for a user, but it can be useful for another user who resolves the same problem, but probably in a different context (or he has different preferences). These interdependencies between information sources and their implicit relations to the given problem in a given context are exactly what have been lost in packaging knowledge. Therefore, instead of monolith, closed packages with distilled knowledge stored in a central repository, flexible structures that link raw information locally (at a user's desktop), that can be easily accessed and further processed in a certain personal context are needed. Moreover, expecting that a user will perform all these structuring of knowledge manually is not at all realistic.

Hence, a new, lightweight, more decentralized concept for sharing software development knowledge is needed. Due to the heterogeneity of a distributed community (e.g., regarding its domain knowledge, experience, used vocabulary, etc.), **the new concept requires a proper abstraction mechanism in order to facilitate an efficient communication.** Finally, **the concept should be more seamlessly integrated into software development environments in order to enable knowledge creation with minimal overhead.**

4.2 Knowledge Management Lifecycle and the Thesis' Approach

The remaining of this section analyzes the thesis' approach. This analysis, leads to research challenges (denoted in this section as RC_x where x is a sequential number) that need to be addressed. Several requirements are extracted (denoted as RQ_x) that lead to a knowledge management system which supports software developers.

Natali and Falbo (Natali & Falbo, 2002) note that knowledge management in software engineering might be exploited to capture software engineering knowledge and experience produced during the execution of projects. Even if software differs from one project to

another there is still similar knowledge that can be reused in order to fasten the development process. This can assist in the avoidance of problems based on the experience gathered inside an organization.

Exactly this need of managing knowledge in software development is the motive of this thesis. Thus:

RC1 the thesis approaches ***managing knowledge in software development based on the entire knowledge management lifecycle***

Furthermore, as already mentioned in chapter 1, another motive for the thesis is the:

RC2 ***enhancement of knowledge management in software development by exploiting social semantic desktop technologies***

In the rest of this section a typical knowledge management systems' lifecycle is depicted proposed by Probst (Probst, 1998), and the respective steps in this lifecycle that should be supported (and how these steps are supported). This is shown in Figure 3.1. The building blocks represent activities that are directly knowledge-related. The inner cycle consists of the building blocks of identification, acquisition, development, distribution, preservation, and use of knowledge. The outer cycle consists of all these activities plus goal-setting and measurement. This feedback cycle clarifies the importance of measuring the measurable variables in order to focus on goal-oriented interventions.

Many knowledge problems occur because organizations neglect one or more of these building blocks and thus interrupt the knowledge cycle. For example, if the steps of an important problem-solving process are not documented, they may disappear from the organization's memory, making successful repetition of the process impossible.

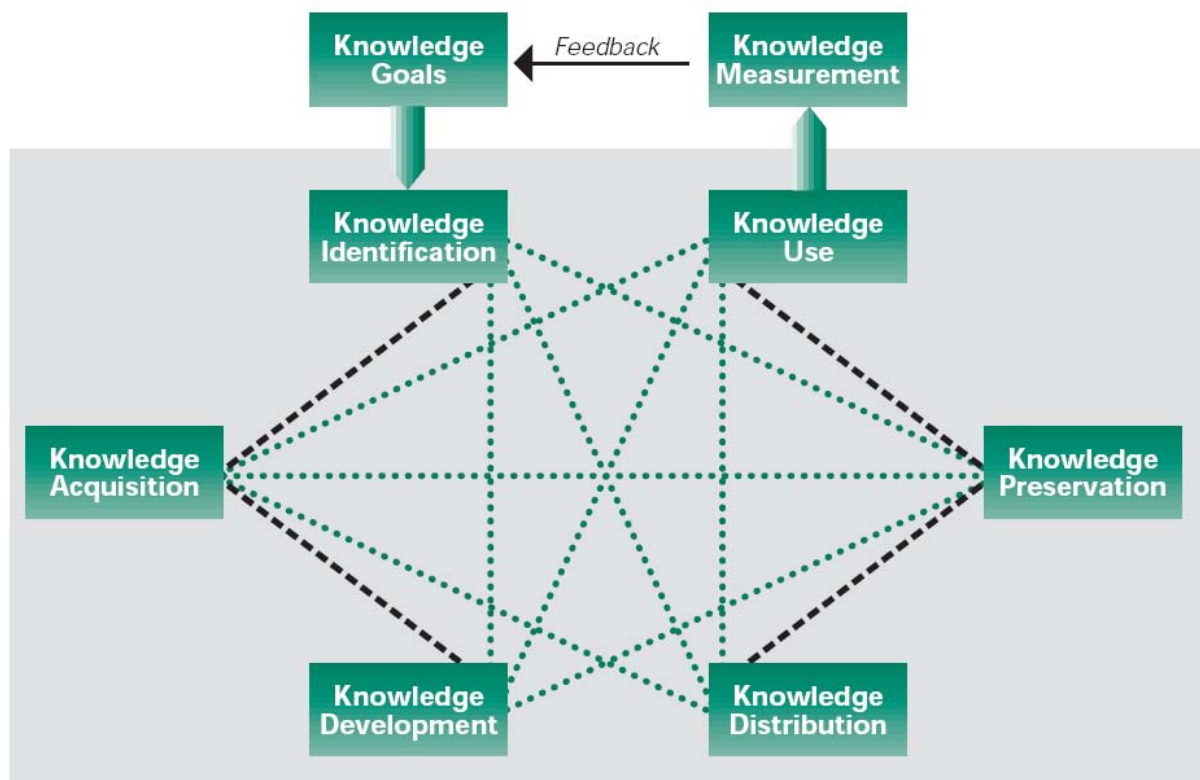


Figure 3.1: The building blocks of knowledge management systems (Probst, 1998)

4.2.1 Knowledge Goals

Knowledge goals point the way for knowledge management activities. This building block deals with the creation of a “knowledge-sensitive” corporate culture, in which sharing and development of know-how create the preconditions for effective knowledge management. *It also ensures that knowledge goals will be translated into action.* For example, a typical knowledge goal might be the accessibility of all internal documents in the company via a suitable intranet.

The thesis’ approach leads to the development of the KnowBench knowledge management system which supports the coding phase in the software development process. The main goal of the KnowBench system is to enable better and faster capability of sharing and exploiting knowledge.

4.2.2 Knowledge Identification

Knowledge is information combined with experience, context interpretation and reflection. It is a high-value form of information that is ready to be applied in decisions and actions. Knowledge can be viewed as formal and informal knowledge. Formal knowledge

can be expressed in a structured form, and easily communicated and shared. Informal knowledge is highly personal and hard to formalize, making it difficult to share with others. It is embedded in individual experience and involves intangible factors such as personal belief, perspective and value.

Before making effort in the development of new capabilities, it should be known what knowledge exists both inside and outside an organisation. Most organisations lose track of their internal and external data, information, and knowledge. This lack of transparency leads to inefficiency, uninformed decisions, and redundant activities. Effective knowledge management creates sufficient internal and external transparency and supports employees in their knowledge-seeking activities.

In the context of software development, formal knowledge includes software engineering methods, document templates, components, software artefacts, and so on. Examples of informal knowledge are lessons learned that are gained as a result of the work of the organization itself and that describe both successful reports and problems. Reuse of lessons learned from past software projects promotes good software development practices and prevents the repetition of mistakes. Thus, modelling, capturing and supporting the creation and usage of all types of knowledge described above should be supported.

Some of the sources of knowledge (e.g., the artefacts) are stored by default in electronic form, by the very nature of software development, so the use of knowledge management is facilitated in the software industry. However, this source can be viewed as raw data and information. Knowledge management seeks to turn data into information, and information into knowledge. ***The most eminent problem is, however, that just a fraction of all knowledge related to software is captured and made explicit. The majority of knowledge is tacit, residing in the brains of the employees.***

In order to increase knowledge availability:

RC3 *ontology-based capture of tacit knowledge* is supported by:

RQ1 *exploiting and integrating existing information and collecting new knowledge*

Ontologies

which facilitate communication and information exchange, support systematic access to parts of the organizational knowledge base, connect various types of knowledge, improve use of knowledge, etc.

The ontologies constitute the glue that binds knowledge management activities together, allowing a content-oriented view of knowledge management. They model the critical knowledge relevant for software development process, which includes:

existing knowledge artefacts (e.g. source code, issue tracking entries, commit messages, documents, etc.); and

organizational entities such as users, projects and resources.

It should be noted here that the ontologies should model formal and informal knowledge related to software development processes in a machine understandable form, which enables provision of a qualitatively new level of services, such as gap analysis (although, it is out of scope of the thesis). Additionally, the aim is not to replace existing artefacts, but assist the augmentation and leverage existing knowledge.

4.2.3 Knowledge Acquisition

The explosive growth and simultaneous fragmentation of knowledge have made it all but impossible for companies to build up the know-how they need for market success by themselves. Instead, companies have to capture critical capabilities or to buy them, often from several knowledge markets.

Role of search in knowledge acquisition

Knowledge artefacts, such as source code, components, documentation, issues from tracking systems, etc. typically contain knowledge that is rich in both structural and semantic information. Providing a uniform ontological representation for various knowledge artefacts it is possible to utilize semantic information conveyed by these artefacts and to establish their traceability links at the semantic level.

One of the roles of search is to automatically create an initial knowledge base by “importing” already existing information. It is important not to replicate existing information, but rather provide an explicit model which helps augmenting and leveraging the existing information.

Knowledge acquisition can be done by:

RQ2 extracting metadata about knowledge artefacts from structured content (e.g. code structure) and from natural language text (e.g. comment in code or knowledge documents)

RQ3 *linking knowledge artefacts - having generated metadata for various kinds of artefacts - it is necessary to relate them semantically.* For example, identifying concepts that appear in issue reports is essential for knowledge management. Certain concepts, such as variable names, function names and log messages, are likely to appear in issue/bug reports and provide valuable hints about the code that is involved in producing the bug.

In this way it is possible to automatically acquire existing knowledge that could be useful for the most important tasks of software development. This approach does not require human processing/understanding, and more importantly provides support for integration.

4.2.4 *Knowledge Development*

Knowledge development consists of all the management activities intended to produce new internal or external knowledge on both the individual and the collective level. The process of individual knowledge development relies on creativity and on systematic problem solving methods. Collective knowledge development involves the learning dynamics of teams. Moreover, there must be an atmosphere of openness and trust to allow the intensity of communication that makes collective learning results superior to individual ones.

One of the targets of the thesis is to support individual knowledge development as well as collective knowledge development.

The metadata⁴⁵ development needs guidance from ontologies. In order to allow for sharing of knowledge, newly created metadata must be consistent with the ontologies.

Thus:

RC4 *individual and collective knowledge articulation*

is needed. This can be supported by:

RQ4 *annotating knowledge artefacts by using a unified vocabulary that ensures unambiguous communication*

⁴⁵ The term metadata is used as a synonym for an ontology individual and its property instantiations

RQ5 *manual metadata development:* ontologies are used in order to guide developers towards creating relational metadata. An analysis (Hands Schuh & Staab, 2003) has shown that subjects have more problems with creating relationship instances than with creating attribute instances. Without the ontology they would miss even more cues for assigning relationships between class instances. The user interface should support manual metadata development by dynamically creating dialogs whose content is generated based on an entity from the ontologies. An efficient visualization of the ontologies would assist in choosing correctly the most appropriate classes for instances.

RQ6 *semi-automatic metadata development:* creating metadata manually is very time-consuming. Thus, this process should be automated as much as possible by taking advantage of information extraction techniques to propose annotations. It is evident that the effort of the developer to define metadata and to annotate knowledge artefacts can be significantly decreased by using above mentioned semi-automatic methods.

RQ7 *ontology-based metadata editing* which allows: (i) browsing ontologies; (ii) looking at their individuals and (iii) following their relationships.

RQ8 *lightweight knowledge articulation supported by semantic wiki.* Its purpose is to provide assistance to software developers in accomplishing their tasks in a more flexible fashion and shorten their total effort in terms of time. In order to achieve this goal, the user interface approaches software development documentation and related problem solving by combining lightweight yet powerful wiki technologies with semantic web standards. Wikis provide significant support in knowledge articulation by providing lightweight and flexible mechanisms while semantics add structured formulation to the wiki contents. This combination gives software developers the opportunity to capture their knowledge related with a software development artefact exploiting a structured formalism. This knowledge will be then reusable, more easily shared and retrieved in the future by the semantic wiki's resilient user interface.

4.2.5 Knowledge Distribution

In making knowledge available and usable across the whole organization, the critical questions are: *Who should know what, to what level of detail, and how can the organization support these processes of knowledge distribution?* Not everyone needs to know everything.

Technical knowledge distribution infrastructures can support efficient knowledge exchange within organizations and connect formerly separated experts through an electronic network. Relevant technologies are groupware, modern forms of interactive management information systems, and all instruments of computer-supported cooperative work – CSCW (K. Schmidt & Bannon, 1992).

Knowledge sharing should be fostered in any approach that aims at improving CSCW processes of software developers. It assists communities of practice and interests (represented as a P2P network) in speeding up the learning curve.

Thus, *lightweight metadata-based P2P services* are required which should provide:

RQ9 *easy, rapid access to distributed knowledge in order to stimulate people to share and use shared knowledge*

Since knowledge artefacts are represented as ontology individuals with all their (data and object) properties' instantiations, different strategies can be applied for publishing properties' instantiations, such as publishing only data properties instantiations, publishing all properties without publishing target individuals, etc.

4.2.6 Knowledge Use

Knowledge use – meaning the productive deployment of organizational knowledge in the software development process – in fact is the purpose of knowledge management. ***The successful identification and distribution of critical knowledge does not ensure its daily use.*** Without consistent use, there is a high probability that new knowledge management systems will decay in quality, and the investment will be wasted. The potential user of knowledge has to see a real advantage in order to change his or her behaviour and “adopt” the knowledge.

To gain developer acceptance ***the envisioned system should be integrated into the software development organization's process (e.g. in the used IDE)***, allowing collecting and storing relevant knowledge as this is generated while working. The main advantage of integrating knowledge management into an IDE is that knowledge management is put into software engineers' workflow, since software development activities occur inside the computational environment rather than in the external world. Consequently, the respective system, i.e. the user interface should also be integrated into the existing work environment.

Visualisation

Since “information visualisation is the use of computer-supported, interactive, visual representation of abstract data to amplify cognition” (Card, Mackinlay, & Shneiderman, 1999), the

- RC5** *multifaceted knowledge representation based on semantic visualization of data* can assist in understanding the domain and in resolving problems. This research challenge can lead to four requirements:

RQ7 (see above) *ontology-based metadata editing*

RQ8 (see above) *lightweight knowledge articulation supported by semantic wiki*

RQ10 *graphical representations of ontology-based data*

In order for this to be achieved the user interface can combine graphically (transparently and intuitively) metadata with ontologies.

RQ11 *representation of different aspects of the underlying information - easy and flexible presentations of the same information in different ways.*

Searching

People in software organizations spent a lot of their time in searching for and accessing different types of information related to their projects. Thus, it is prominent that *intelligent search* should be provided.

Knowledge management can support information access through searching. A developer could search for any kind of knowledge:

RQ12 *formal knowledge search (i.e. ontology-based entities in the metadata repository)* or

RQ13 *informal knowledge search (i.e. indexes in “integrated” index storage)*

Consequently, two types of search should be supported: structured-based and keyword-based. While keyword-based search represents the standard model for search interfaces, structured search allows more precise queries which in turn yield more precise results. The latter is especially useful for software developers, since they are skilled enough to formulate structured queries and parts of their problem domain can be precisely described (“Give me all bugs for a certain component”). Additionally, both approaches could be complemented by:

RQ14 semantic search which exploits underlying ontologies and background knowledge (metadata-based). The ontologies can be used for query formulation as well as for query processing (e.g. ranking). Furthermore, search can be performed on two levels:

RQ15 local level search, where a search for relevant knowledge within the local knowledge base is performed; and

RQ16 distributed (P2P) level search, where a search for relevant knowledge across the P2P network is performed.

Dealing with information overload (too many results) as well as no answers to problems requires:

RQ17 query refinement, relaxation and similarity

Query refinement should be supported: i.e. suggestions for query reformulation could reduce the long time it takes to find information. Query refinement strives to decrease the size of a too large result set by extending the initial query. Query relaxation proposes to dismiss certain constraints expressed in the initial query, in order to yield additional results that were previously dismissed. The goal of query similarity is to retrieve similar queries, based on a certain similarity measure. It is important to note here, that the provision of all above mentioned services are based on extensive usage of ontologies.

User feedback should be taken into account in the search process to refine information needs. Moreover, one of the problems addressed is the fact that no software project is like another. Knowledge artefacts matching the reuse needs are rarely found. Therefore, a good reuse approach must find similar knowledge artefacts and let modifications on selected artefacts.

4.2.7 Knowledge Preservation

After knowledge has been acquired or developed, it must be carefully preserved. Many companies complain that in the process of reorganization they have lost part of their corporate memory. To avoid the loss of valuable expertise, companies must shape the processes of selecting valuable knowledge for preservation, ensuring its suitable storage, and regularly incorporating it into the knowledge base.

To support the knowledge management process in general, a knowledge management infrastructure should be provided. The knowledge repository lies at the core of this

infrastructure. Arranged around the knowledge repository, knowledge management services will actively provide useful information to developers working on knowledge-intensive tasks. These knowledge management services correspond to the activities of the knowledge management process: creation, capture, retrieval, access, dissemination, use, and preservation of the organization's knowledge.

Local (private) knowledge (not available to others) and public (distributed) knowledge (available to others) should be distinguished and stored transparently by using an abstract layer.

All artefacts guiding a software project and developed during a software project could be represented formally and explicitly based on ontologies. This means, that these explicit assets of the software organization can be made not only:

RQ18 *machine readable but also machine understandable knowledge*, i.e. ontology-based representation can be (manually or automatically) created.

These assets directly support the core business and must be managed efficiently in order not to be lost. The problem of transferring knowledge from experts to novices will be also facilitated since the knowledge is readily captured, stored, and organized in an ontology-based repository.

It is important to prevent loss of knowledge and enhance the accessibility to organizational knowledge by providing a well-structured knowledge repository (i.e. provide a framework for creating, modifying, querying, and storing ontologies and metadata). Thus:

RQ19 *knowledge retaining* is important.

Public knowledge should be available to others via a P2P network. Thus, a distributed knowledge management system is mandatory: (i) each peer provides all the services needed by a knowledge node to create and organize its own local knowledge, and (ii) protocols of meaning negotiation are introduced to achieve semantic coordination (e.g., when searching knowledge from other peers).

Ontologies are the main candidate for knowledge representation. They have shown to be the right answer to knowledge structuring and modelling by providing a formal conceptualization of a particular domain that is shared by a group of people in an organization. However, ***knowledge management systems based on centralized ontologies need a long development phase and are difficult to maintain***. Thus, the only ontologies

used for annotation (i.e. domain ontology and software engineering ontology) should be subject to change. Each peer should be allowed to share their own knowledge space within a network of autonomous peers, to make a part of knowledge in this space available to other peers, and to search relevant knowledge from the knowledge space of other peers. Thus, the P2P network should be viewed as a way of enabling distributed control, customization, and redundancy. On the other hand the knowledge repository should ensure central control, standardization, high capacity, and robustness.

4.2.8 Knowledge Measurement

Evaluation and measurement of values of organizational knowledge presents the biggest challenge in the field of knowledge management. There is no tested tool box of accepted indicators and measurement processes. Additionally, the subject that needs to be measured is particularly elusive. Knowledge and capabilities can rarely be tracked to a single influencing variable. Furthermore, the cost of measuring knowledge is often seen as too high or socially unacceptable. Nevertheless, knowledge measurement holds considerable potential value.

Knowledge artefacts do not automatically create organizational capabilities. In most cases, their potential can be realized only through human action and usage. Thus, the expected advantages of a system improving quality of the software, reducing time-to-market, increasing productivity, etc. can be seen in the long-run. The main reason for that is that knowledge management is a process implemented over a period of time, which has much to do with human relationships as it does with business practices and information technology. Thus, the system resulting from the thesis' approach combines tools and technologies to provide support for capturing, accessing, reusing and disseminating knowledge, which in turn generates benefits for the software organization and its developers. Thus, performing *evaluation of the usability of the system* is mandatory. Summative evaluation focuses on the importance of knowledge artefacts and how they fit with the company's goals.

4.3 Research Challenges overview

Figure 4.2 depicts a bird's eye view of the research challenges that are addressed by the thesis (see also section 4.2). Research challenges RC1, RC2 are more high level than RC3, RC4, RC5 and thus are depicted in the upper part of figure 4.2.

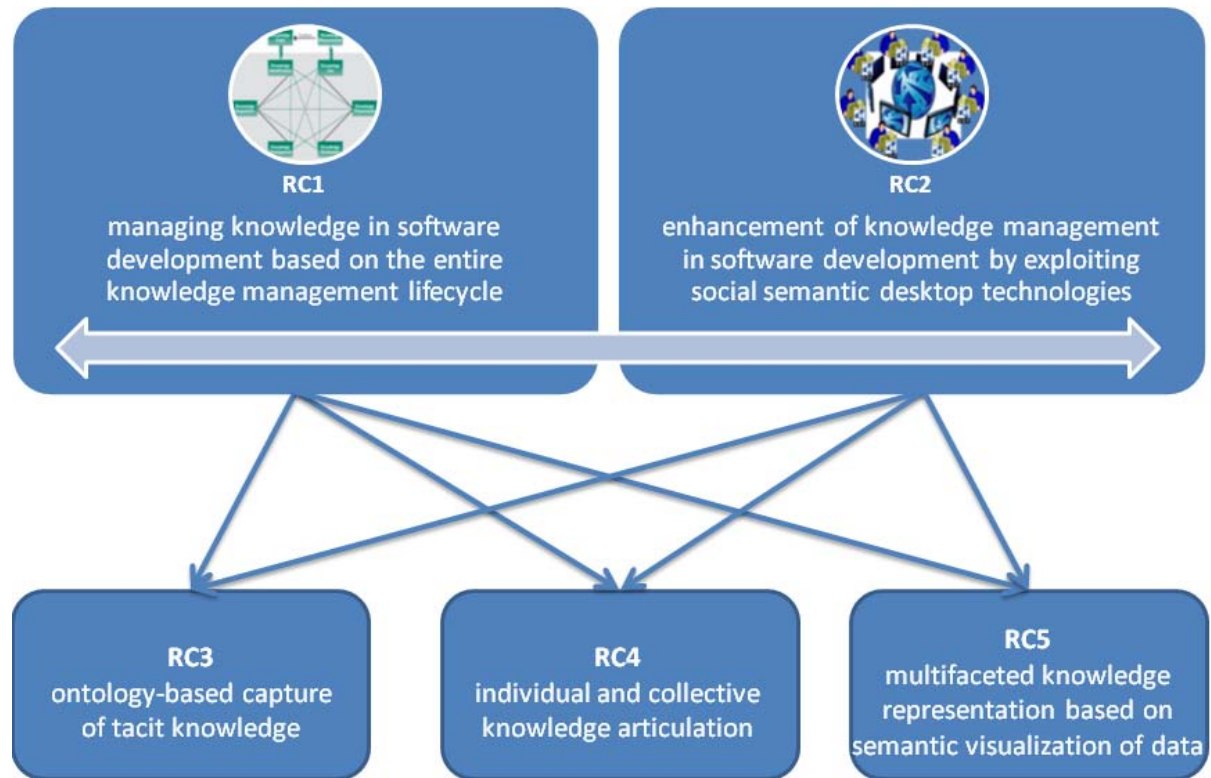


Figure 4.2: Research challenges of the thesis

Table 4.1 summarizes all the research challenges (and the respective knowledge processes), as well as the positioning of the thesis' approach in relation to state-of-the-art.

Research challenge	Knowledge process	Positioning of the thesis' approach	Related work
RC1 managing knowledge in software development based on the entire knowledge management lifecycle	whole KM lifecycle	There is no approach in the literature that provides holistic (in all processes of the KM lifecycle) knowledge management support in software development. Instead, current approaches focus only on partial support of the KM lifecycle.	(Basili et al., 1994), (Basili et al., 2001), (Basili et al., 2002), (Holz, 2003a), (Ag, 2008), (<i>Skillscape Competence Manager</i> , n.d.), (<i>SkillSoft Skillview</i> , n.d.), (Ye & G. Fischer, 2002), (Henninger, 1991), (Holmes & Murphy, 2005), (Leitao, 2004), (T. Chau & F. Maurer, 2006), (Cook & Churcher, 2006)

RC2 enhancement of knowledge management in software development by exploiting social semantic desktop technologies	whole KM lifecycle	State-of-the-art approaches do not support knowledge management in software development powered by social semantic desktop technologies. These approaches either provide isolated semantic desktop or semantic wiki solutions and thus, there is no integrated solution providing both.	(Natali & Falbo, 2002), (Happel & Seedorf, 2006), (Eberhart & Argawal, 2004), (Witte et al., 2007), (Bauer & Roser, 2006), (Thaddeus & Kasmir, 2006), (Happel et al., 2006), (Pan & Heflin, 2003), (Abadi et al., 2007), (Athanasiadis et al., 2007), (Noll & Ribeiro, 2007), IBM Jazz platform ⁴⁶ , (B. Decker et al., 2005), (Klein et al., 2005), (Rauschmayer, 2005)
RC3 ontology-based capture of tacit knowledge	knowledge goals, knowledge acquisition, knowledge preservation	Current approaches provide only ontology-based capture of explicit, instead of tacit knowledge. The thesis' approach provides both kinds of knowledge capturing. Furthermore, state-of-the-art solutions support only manual interlinking of knowledge artefacts. In the thesis' approach, automatic interlinking is	(Natali & Falbo, 2002), (Happel & Seedorf, 2006), (Eberhart & Argawal, 2004), (Witte et al., 2007), (Bauer & Roser, 2006), (Thaddeus & Kasmir, 2006), (Happel et al., 2006), (Pan & Heflin, 2003), (Abadi et al., 2007), (Athanasiadis et al., 2007), (Noll & Ribeiro, 2007), IBM Jazz platform, (B. Decker et al., 2005), (Klein et al., 2005), (Rauschmayer, 2005), (Kapor, 2005), (Richter et al., 2005), (Sauermann et al., 2006), (Karger

⁴⁶ <http://www-01.ibm.com/software/rational/jazz/>

		supported in order to facilitate tacit knowledge capturing.	et al., 2005), (Cheyer et al., 2005), (S. Decker & Frank, 2004), (Schaffert et al., 2005), (Malte, 2006), (Dello et al., 2006), (Auer et al., 2006), (Krötzsch et al., 2006), (Oren, 2005), (Michel Buffa & Gandon, 2006), (Aumueller & Auer, 2005)
RC4 individual and collective knowledge articulation	knowledge development	The advantage of the thesis' approach in relation to the state-of-the-art is that individual as well as collective knowledge articulation are supported. This is achieved by using semantic annotations (in manual and semi-automatic ways), as well as by allowing developers to articulate knowledge inside various metadata-based editing environments. Current approaches do not focus on both types of articulation, but rather emphasise either on individual or collective articulation.	(Natali & Falbo, 2002), (Happel & Seedorf, 2006), (Eberhart & Argawal, 2004), (Witte et al., 2007), (Bauer & Roser, 2006), (Thaddeus & Kasmir, 2006), (Happel et al., 2006), (Pan & Heflin, 2003), (Abadi et al., 2007), (Athanasiadis et al., 2007), (Noll & Ribeiro, 2007), IBM Jazz platform, (B. Decker et al., 2005), (Klein et al., 2005), (Rauschmayer, 2005), (Kapor, 2005), (Richter et al., 2005), (Sauermann et al., 2006), (Karger et al., 2005), (Cheyer et al., 2005), (S. Decker & Frank, 2004), (Schaffert et al., 2005), (Malte, 2006), (Dello et al., 2006), (Auer et al., 2006), (Krötzsch et al., 2006), (Oren, 2005), (Michel Buffa & Gandon, 2006), (Aumueller & Auer, 2005)
RC5 multifaceted	knowledge use	Current approaches in literature do not suggest	(Kapor, 2005), (Richter et al., 2005), (Sauermann et al., 2006),

knowledge representation based on semantic visualization of data		multifaceted knowledge representation based on semantic visualization of data. Only few of them provide graphical representations of ontology-based data but there are limitations as the same ontological data are not represented in several facets of the same information. Instead, the thesis' approach suggests this multifaceted knowledge representation in order to facilitate several views of the data, thus achieving knowledge usage in an easier fashion.	(Karger et al., 2005), (Cheyer et al., 2005), (S. Decker & Frank, 2004), (Schaffert et al., 2005), (Malte, 2006), (Dello et al., 2006), (Auer et al., 2006), (Krötzsch et al., 2006), (Oren, 2005), (Michel Buffa & Gandon, 2006), (Aumüller & Auer, 2005)

Table 3.1: Research challenges and positioning of the thesis' approach in relation to the state-of-the-art

4.4 Conclusions

Software development is a collective, complex, and creative effort. To deal with complex software processes, it becomes essential to provide computer-based tools to support software developers to perform their tasks.

The thesis' approach leads to the realization of a *knowledge management system* powered by *social semantic desktop* technologies – namely KnowBench which *supports developers during the software development process* to produce better quality software. The

goal of KnowBench is to collect knowledge the developers gain during software development in order to avoid mistakes and leverage success in future projects.

Although benefits can be derived from individual tools addressing separate software development activities, the real power of the KnowBench system lies in supporting the whole knowledge management process. Indeed, KnowBench can be considered as integrated collections of services. These services, facilitate software development activities by applying the knowledge management paradigm in capturing, storing, disseminating, and reusing knowledge that is created during these activities as well as in integrating existing sources.

Knowledge management requires both a shared language and a good fit with concepts that already exist in the organization. In the context of the KnowBench knowledge management system, the KnowBench ontologies define the shared vocabulary that will be used to facilitate communication, search, storage, and representation. The ontology-background of the KnowBench system enables to move from a document-oriented view of knowledge management to a content-oriented view, where knowledge artefacts are interlinked, combined, and used. Furthermore, the KnowBench ontologies constitute the glue that binds the KnowBench components together. Since all knowledge handled has a formal meaning (semantics) associated, it will be accessible not only to human developers, but also to automated components/tools. The key idea is to have software development data/knowledge defined and linked in such a way that its meaning is explicitly interpretable by software tools rather than just being implicitly interpretable by human developers. In KnowBench's knowledge management approach, ontologies are used to structure the metadata repository, as well as to support the main knowledge services, such as creation, usage, search, reuse and sharing of knowledge artefacts.

4.4.1 Summarization of Thesis' Contributions

This thesis contributes in current research by providing an approach which strives to resolve problems during *managing knowledge in software development* (supporting developers in the *whole knowledge management lifecycle*) by exploiting *social semantic desktop* technologies.

Furthermore, the approach provides:

- ***explicit and tacit knowledge capturing*** - automatic interlinking is supported in order to fasten tacit knowledge capturing.

- ***individual as well as collective knowledge articulation*** - this is achieved by using semantic annotations (in manual and semi-automatic ways), as well as by allowing developers to articulate knowledge inside various metadata-based editing environments.
- ***multifaceted knowledge representation based on semantic visualization of data*** - the thesis' approach suggests multifaceted knowledge representation in order to facilitate several views of the same data, thus achieving knowledge usage in an easier fashion.

Another contribution of the thesis is the design and development of a semantic-based knowledge management system (KnowBench – presented in the following chapters 5, 6 and 7). KnowBench is developed as a proof of concept of the thesis' approach. It is constituted by the following parts:

- Manual semantic annotation
- Semi-automatic semantic annotation
- Knowledge base editing
- P2P services
- Semantic search
- Software Development semantic Wiki (DevWiki)
- Knowledge base graph-based browsing

5 KNOWBENCH FUNCTIONALITY

This chapter presents KnowBench. First, the KnowBench functionality overview is presented and then a mapping between the requirements and the desired KnowBench functionality (see table 5.1 – the section which describes every functionality is shown in the table). Finally, a more detailed analysis of the functionality is given in section 5.3.

5.1 KnowBench at a Glance

One of the results of this thesis is the design and implementation of functionality to assist software development work inside the Eclipse IDE exploiting the power of Semantic-based technologies (e.g. manual/semi-automatic Semantic Annotation, semantic wiki, Knowledge Base graph-based browsing, etc.).

An overview of the whole KnowBench system functionality is given in the following list:

Manual Semantic Annotation of source code with desired granularity (package/class/method/variable level etc.) – multiple views of semantic annotations are available (either in KB editor, KB graph browser, semantic wiki or annotations view) and extension of the annotation ontology is feasible (domain and software engineering concepts).

Semi-automatic Semantic Annotation of source code corpora (both annotating one file and a directory including several files is supported). This kind of annotation is supervised by the user and results in new concepts and sometimes in their classification inside the annotation ontology hierarchy tree. These new or existing concepts of the annotation ontology are then used to annotate semantically the source code corpora. This procedure assists and shortens the time consuming task of manually annotating source code.

Knowledge base editing provides access to the knowledge base in order to allow the software developer to browse, insert, modify or remove software engineering knowledge objects. The concepts presented to the user by the KB editor (as well as by other portions of the KnowBench such as Wiki or KB graph browser) are easily configurable in a configuration file. However, it was decided that this selection of concepts should not be available through KnowBench's UI due to the fact that only specific persons (administrators)

with special access rights or knowledge of the KnowBench system and the ontologies themselves should change the concepts that can be browsed and manipulated (insertion, deletion and modification of instances). The knowledge base editor does not allow the modification of the ontology schemas. The only schema that is allowed to be changed is the annotation ontology schema and only through one entry point, which is the manual/semi-automatic Semantic Annotation UI.

Knowledge base graph-based browsing facilitates knowledge base browsing using graphs (designed and developed using the open source JPowerGraph graphics library⁴⁷ developed by the Digital Enterprise Research Institute (DERI)⁴⁸). The KB graph browser provides several advanced features such as zooming in/out, animation, filtering, auto-layout, expansion, collapse of nodes, etc. This functionality is provided as a standalone RCP application independent of the whole KnowBench installation as well as an Eclipse plug-in.

P2P services. KnowBench provides P2P metadata-based services. In this way, developers can share their knowledge and exploit knowledge created by other developers. The UI for joining and publishing metadata to a specific P2P network is provided. However, it does not provide UI for creating and configuring P2P networks as this is a very delicate and specialized procedure that targets system administrators who will configure the system as per needs of each application environment. The creation and configuration of P2P networks can be done using external tools and the resulting configuration files can be distributed to the users by the administrators.

Semantic search is provided allowing the user to issue a query and consult the respective results of the search. KnowBench facilitates the expression of keyword, structured and semantic search and enables the user to refine or relax/constrain the query depending on the satisfaction of his knowledge need.

Software development semantic wiki provides access to the knowledge base and gives an alternative way to browse, insert or modify instances of the KB. It uses a lightweight and flexible editor with auto-completion and popup support to assist software developers in the

⁴⁷ <http://sourceforge.net/projects/jpowergraph>

⁴⁸ <http://www.deri.ie/>

articulation of knowledge. Browsing through knowledge is done like surfing through a conventional wiki using the Semantic links between different knowledge artefacts. This browser is available inside the Eclipse IDE so that the software developer does not have to switch to another external browser.

Preferences and configuration of subsystems is allowed through the standard Eclipse preferences mechanism. The user is allowed to configure the search subsystem, to define policy rules for sharing knowledge to other peers, etc.

Help subsystem provides assistance and guidelines to software developers in order for them to be able to use KnowBench in a proper way and effectively.

5.2 KnowBench Functionality Extraction from Requirements

Table 5.1 describes the mapping of requirements extracted in chapter 4 to desired KnowBench functionality.

Requirement	Functionality	Sections
RQ1 - exploiting and integrating existing information and collecting new knowledge	Knowledge base editing, semi-automatic semantic annotation	5.3.2, 5.3.7
RQ2 - extracting metadata about knowledge artefacts from structured content (e.g. code structure) and from natural language text (e.g. comment in code or knowledge documents)	semi-automatic semantic annotation	5.3.7
RQ3 - linking knowledge artefacts - having generated metadata for various kinds of artefacts - it is necessary to relate them semantically	Manual semantic annotation, knowledge base editing, software development semantic wiki, semi-automatic semantic annotation	5.3.1, 5.3.2, 5.3.5, 5.3.7
RQ4 - annotating knowledge artefacts by using a unified vocabulary that ensures unambiguous	Manual and semi-automatic semantic	5.3.1, 5.3.7

communication	annotation	
RQ5 - manual metadata development	Manual semantic annotation, knowledge base editing, software development semantic wiki	5.3.1, 5.3.2, 5.3.5
RQ6 - semi-automatic metadata development	semi-automatic semantic annotation	5.3.7
RQ7 - ontology-based metadata editing	knowledge base editing	5.3.2
RQ8 - lightweight knowledge articulation supported by semantic wiki	software development semantic wiki	5.3.5
RQ9 - easy, rapid access to distributed knowledge in order to stimulate people to share and use shared knowledge	P2P services	5.3.3
RQ10 - graphical representations of ontology-based data	Knowledge base graph-based browsing	5.3.8
RQ11 - representation of different aspects of the underlying information - easy and flexible presentations of the same information in different ways	Knowledge base editing, software development semantic wiki, knowledge base graph-based browsing	5.3.2, 5.3.5, 5.3.8
RQ12 - formal knowledge search (i.e. ontology-based entities in the metadata repository)	Semantic search	5.3.4
RQ13 - informal knowledge search (i.e. indexes in “integrated” index storage)	Semantic search	5.3.4
RQ14 - semantic search which exploits underlying ontologies and background knowledge (metadata-based)	Semantic search	5.3.4
RQ15 - local level search	Semantic search	5.3.4

RQ16 - distributed (P2P) level search	Semantic search	5.3.4
RQ17 - query refinement, relaxation and similarity	Semantic search	5.3.4
RQ18 - machine readable but also machine understandable knowledge	This is achieved through the KnowBench ontologies	–
RQ19 - knowledge retaining	Knowledge base editing, software development semantic wiki	5.3.2, 5.3.5

Table 4.1: Mapping of requirements derived from chapter 4 to desired KnowBench functionality

5.3 KnowBench Functionality

5.3.1 Manual Semantic Annotation

An important aspect of the KnowBench is the ability to annotate semantically source code. The standard Eclipse JDT editor is extended to add this possibility. The software developer is able to annotate source code with semantic annotation tags that are available or define new tags and extend the annotation ontology. When extending the annotation ontology the user can define which annotation tag he extends by assigning to the new tag an existing annotation tag as its parent or alternatively create a new “tag family” which is under Software Engineering or Domain Entity.

The manual semantic annotation of source code provides granularity regarding the respective source code fragment to be annotated. This granularity level is restricted by the underlying Eclipse platform itself as the `IJavaElement` interface is used to map between source code fragments and metadata. This limits the selected source code fragments to be annotated in the nearest Java elements that surround the source code fragment at hand (e.g. package/class/method/variable etc.). However, if this Eclipse mechanism is updated or extended in the future, KnowBench’s manual semantic annotation capabilities can be extended as well in order to reflect the new more fine-grained granularity of Eclipse source code structure management.

The above mechanism has been introduced to address the synchronization of the selected source code fragment with the annotation ontology. Else, this could cause problems in cases where a user typed in some source code inside an already annotated region because the annotation should be changed accordingly to include the new source code part. This could lead to performance bottlenecks as well, if the synchronization of the ontology with the source code file was used every time the software developer was about to change something in the code. Thus, it is not necessary to synchronize the metadata with the actual source code file, since the metadata refer to java elements actually providing a link to the file at that specific point regardless its position and its enclosed text and synchronization is not required as such.

Additionally with using `IJavaElement` a semantic annotation mechanism that allows the annotation of several objects is provided, such as Java packages, classes, methods, etc. Furthermore, the semantic annotations view provides detailed information such as the code entity type (class, method, variable, etc.), code entity name and whether the annotation at hand is a software engineering or a domain annotation.

When a manual semantic annotation is added the following metadata are added to the knowledge base automatically:

Code entity instance – the type of the code entity (i.e. the corresponding subclass of the Code entity concept in the ontology) depends on the auto-detected Java element that surrounds the selected source code fragment. “Is about” or “Is related to SE entity” and “Is contained in” object properties are instantiated with respective values. “Is about” refers to a domain annotation while “Is related to SE entity” refers to a software engineering annotation. “Is contained in” refers to the relevant source code file that contains the code entity. The ID/URI of the code entity instance is created using a relative path to the Eclipse workspace of the class containing the source code fragment, without the workspace path itself in order to avoid references to nonexistent paths when sharing the metadata to the P2P network, followed by the signature of the code entity (e.g. `\HelloWorld\src\helloworld\editors\XMLDoubleClickStrategy.java~boolean selectComment(int)` as in Figure 5.1).

Source file instance – the source file that contains the code entity. “Contains” and “Located in” object properties are instantiated with corresponding values. The “Contains” property is the inverse property of “Is contained in” and indicates that a source file contains code

entities. When multiple annotations are added in other code fragments then it is apparent that the source file instance will have as many instances of the “Contains” property as the annotated code entities of the file. “Located in” indicates the physical file resource of the source file.

File instance – the physical file instance that a source file is located.

Annotation class – in the case that the software developer chooses to extend the Annotation ontology and add a new concept then a new class is added in the ontology as well as the corresponding “SubclassOf” relation at the desired level of the ontology hierarchy tree.

It is obvious that annotations in the same source code file will not reproduce all the above mentioned metadata if they already exist. For example, if the software developer adds an annotation to a source file, the first time will result in the creation of all the previous metadata and the second time only instances of the Code entity concept and “Is about” or “Is related to SE entity” properties will be created. Additionally, it is possible to annotate the same code entity with multiple annotations – in this case only “Is about” and “Is related to SE entity” properties are instantiated.

A typical example of using the manual semantic annotation is given below:

A software developer wants to annotate the method “boolean selectComment(int)” of a Java class named “XMLDoubleClickStrategy” which is part of a generic XML editor application and define that this method is used only for banking applications. He annotates the method with a newly created annotation tag (concept) called “Bank” under the domain entity concept in the Annotation ontology. As a result the following metadata statements are created in the knowledge base (Figure 5.1):

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://www.knowbench.eu/ontologies/org.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:j.1="http://www.knowbench.eu/ontologies/team.owl#"
  xmlns:j.2="http://www.knowbench.eu/ontologies/ka.owl#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:j.3="http://www.knowbench.eu/ontologies/content.owl#" >
  <rdf:Description
    rdf:about="http://www.knowbench.eu/ontologies/knowledgebase.owl#\HelloWorld\src\hel
loworld\editors\XMLDoubleClickStrategy.java~boolean selectComment(int)">
```

```

    <j.3:isAbout
rdf:resource="http://www.knowbench.eu/ontologies/Annotation.owl#Bank"/>
    <j.2:isContainedIn
rdf:resource="http://www.knowbench.eu/ontologies/knowledgebase.owl#SourceFile\Hello
World\src\helloworld\editors\XMLDoubleClickStrategy.java"/>
    <rdf:type rdf:resource="http://www.knowbench.eu/ontologies/ka.owl#Method"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.knowbench.eu/ontologies/knowledgebase.owl#SourceFile\HelloWorld\src\helloworld\editors\XMLDoubleClickStrategy.java">
    <j.2:contains
rdf:resource="http://www.knowbench.eu/ontologies/knowledgebase.owl#\HelloWorld\src\helloworld\editors\XMLDoubleClickStrategy.java~boolean selectComment(int)"/>
    <j.1:locatedIn
rdf:resource="http://www.knowbench.eu/ontologies/knowledgebase.owl#File\HelloWorld\src\helloworld\editors\XMLDoubleClickStrategy.java"/>
    <rdf:type rdf:resource="http://www.knowbench.eu/ontologies/ka.owl#SourceFile"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.knowbench.eu/ontologies/knowledgebase.owl#File\HelloWorld\src\helloworld\editors\XMLDoubleClickStrategy.java">
    <rdf:type rdf:resource="http://www.knowbench.eu/ontologies/org.owl#File"/>
</rdf:Description>
</rdf:RDF>

```

Figure 4.1: Semantic Annotation resulting metadata in RDF/XML notation

5.3.2 Knowledge Base Editing

A single Eclipse view provides the user interface for performing addition, modification and browsing of the knowledge base. It was decided to extend the “org.eclipse.ui.editors” Eclipse extension point and provide the knowledge base editor.

An overview of the functionality is given below:

Intuitive guidance of the user through the semantic relationships between several instances. This is addressed by adding the range of each object property in parenthesis after each property name which actually indicates what type of information the property is about. Furthermore, the capability of browsing through these semantic links is added, thus enabling the user to explore the knowledge base by double-clicking on the target of an object property which will result in opening the target instance and display its properties in the knowledge base browser.

Intuitive labels and avoidance of displaying URIs in the user interface. “Comprehensible” labels for ontology concepts and properties are created and provided in order to guide the

user in browsing or entering the required information while articulating or modifying an instance. Additionally, tooltips in concepts and properties which describe them are included. The tooltips are presented to the user when hovering above a term and they display the value of the “`rdfs:comment`” property of the respective RDF resource.

The configuration of the root and moreover the annotation concepts presented in several portions of the KnowBench UI (e.g. wiki, KB editor/graph browser, annotation dialogs, etc.) is easily configurable in the way depicted in Figure 4.2 as already mentioned. Figure 4.2 shows the default configuration of the concepts.

```

ONTOLOGY_TREE_ROOT_LABEL = "Knowbench Concepts";

rootClasses = {"http://www.knowbench.eu/ontologies/ka.owl#CodeEntity",
               "http://www.knowbench.eu/ontologies/content.owl#Comment",
               "http://www.knowbench.eu/ontologies/org.owl#Competence",
               "http://www.knowbench.eu/ontologies/ka.owl#KnowledgeArtefact",
               "http://www.knowbench.eu/ontologies/org.owl#Organization",
               "http://www.knowbench.eu/ontologies/ps.owl#Problem",
               "http://www.knowbench.eu/ontologies/org.owl#Project",
               "http://www.knowbench.eu/ontologies/org.owl#Resource",
               "http://www.knowbench.eu/ontologies/ps.owl#Solution",
               "http://www.knowbench.eu/ontologies/org.owl#Team",
               "http://www.knowbench.eu/ontologies/Annotation.owl#
                SoftwareEngineeringEntity",
               "http://www.knowbench.eu/ontologies/Annotation.owl#
                DomainEntity"};

annotationRootClasses = {"http://www.knowbench.eu/ontologies/Annotation.owl#
                          SoftwareEngineeringEntity",
                          "http://www.knowbench.eu/ontologies/Annotation.owl#
                          DomainEntity"};

```

Figure 4.2: Configuration of the root concepts and/or annotation root concepts presented to the KnowBench UI

5.3.3 P2P Services

The software developer can join and connect to several P2P networks and consequently retrieve knowledge from or publish knowledge to. When he connects to a network he can choose to publish all metadata that he created while being at offline mode

and that match the policy rules that are currently configured. Policy rules are defined by the user using the KnowBench UI which extends the standard Eclipse preferences mechanism (see section 5.3.6). Policy rules declare what kind of knowledge to share to which P2P networks. After this configuration, knowledge is shared on the background if it matches the user criteria and at the time that this knowledge is created and sent to the metadata store component by the KnowBench (e.g. when articulating new knowledge items in KB editor, semantic wiki or annotating objects). For example, a policy rule can be defined by the user to share all solutions related to a specific knowledge artefact.

5.3.4 Semantic Search

KnowBench supports advanced methods for knowledge search through a semantic search engine (Giesbrecht, Stojanovic, & Tran, 2008) by taking into account three different types of search, namely keyword, structured and semantic search.

Keyword search component realizes a Lucene-based traditional paradigm of search, with an advanced functionality of indexing different sources of relevant information for software developers, such as SVN repositories or JIRA issues.

While keyword-based search represents the standard model for search interfaces, structured search allows more precise queries which in turn yield more precise results. “Structured search” is the search defined through ontology entities. In the structured search, users do not need to have any background knowledge on the ontology as they can directly define through the KnowBench GUI the specific classes and the properties that they are looking for. For example, one can choose to search only for the problems which have been identified by the user John.

Structured search promises to provide more accurate results than present-day keyword search.

Semantic search in contrast is the search which automatically translates keyword queries into formal logic queries so that developers can use familiar keywords to perform structured search without having to navigate in advance through the ontology tree.

To realize this, the engine uses a novel approach of adapting keywords to structured queries: the approach automatically translates keyword queries into formal logic. The translation consists of three major steps: keyword mapping, graph exploration and query ranking. The generic graph-based approach is followed to explore the connections between

ontology nodes that correspond to keywords in the query. In this way, all interpretations that can be derived from the underlying RDF graph can be computed.

The ontologies are used in the process of answering queries. Thereby, implicit knowledge (e.g. stemming from information about hierarchical relations or inferred by rules) is taken into account for matching results. However, this does not naturally provide any indication about different degrees of relevancy of a match – i.e. a knowledge artefact is either a result or it is not. In order to rank the results of a query, context-based information is used to capture the meaning intended by the users and thus, the search space is further constraint.

Moreover, to deal with information overload (too many results) as well as no answers to problems, the engine creates suggestions for query reformulation that could reduce the long time it takes to find information. Several possibilities are proposed, including creation of semantic index from semi-structured sources, according to the ontologies. Additionally, semantic relations between different artefacts are drawn based on implicit mutual dependencies.

Semantic search UI facilitates the user in issuing a query and consulting the respective results of the search. KnowBench provides the ability to express keyword (in crawled resources), structured search (locally or in the P2P network) and semantic search that creates automatically structured query based on given keyword and enables the user to refine the query depending on the satisfaction of his knowledge need when viewing the search results. The search is integrated with the standard Eclipse search mechanism and as such in the Eclipse search menu.

Keyword search is the simplest way to perform an index-based by entering free text keywords. These keywords are sent to the search component which performs the respective search. A list of results is returned which is presented by KnowBench to the user.

Structured search can be performed by defining several criteria for the search. The user can define in which concept he wants to search and restrict the query by determining property values that are valid for that specific concept.

Semantic search in contrast is the search which automatically translates keyword queries into formal logic queries so that developers can use familiar keywords to perform structured search without having to navigate in advance through the ontology tree.

5.3.5 *Software Development semantic wiki*

A crucial part of the KnowBench is the wiki. Its purpose is to provide assistance to software developers in accomplishing their tasks in a more flexible fashion and shorten their total effort in terms of time. In order to achieve this goal, software development documentation and related problem solving is approached by combining lightweight yet powerful wiki technologies with semantic web standards.

An overview of its functionality is given below:

Validation of the user input when incorrect values for selected concepts, properties, related instances or property values are introduced. The semantic wiki informs the user about the wrong input value and assists him/her in that way to track down the problem and resolve the issue.

Auto-complete facilities provide assistance to the user for selecting the right concept, properties and related instances of an object property as a value.

Intuitiveness – the software developer is able to understand what is required as input using the auto-complete facilities of the semantic wiki.

Syntax colouring – the software developer is assisted with syntax colouring to easily determine what kind of information is recognized by the wiki engine in order to avoid mistyping.

Multipage editor consisting of the semantic wiki editor and the semantic wiki browser which facilitates navigation of the knowledge base in an HTML-fashion inside the Eclipse environment.

Navigation through the semantic wiki is enabled via following an instance's semantic links to other related instances.

On the fly synchronization with the knowledge base. The new semantic wiki is on the fly which means that no files are kept anywhere and all information is persisted directly to the knowledge base. This one more advantage besides better performance and consistent synchronization with the knowledge base: policy rules that apply to knowledge items are directly applied to a newly saved instance created in the wiki, thus it is immediately shared to the P2P network.

The new KnowBench semantic wiki provides better and more flexible knowledge articulation facilities than the first prototype and gives the software developer an opportunity to use an advanced tool in order to document and develop knowledge required for the successful usage of the KnowBench system.

5.3.6 Preferences and Configuration of Subsystems

The standard Eclipse preferences extension point is extended in order to provide preference pages where configuration (semantic search and P2P policy rules) can be performed.

An overview of the available preferences of KnowBench components is given below:

Source to be crawled by the search component can be added or removed and configured to crawl several repositories as per needs of the user in order for the search system to index the appropriate resources

P2P policy rules can be defined in order to define what kind of knowledge should be shared in P2P networks as already described.

5.3.7 Semi-automatic Semantic Annotation

The need for assisting the user in annotating semantically source code (either one single file of code or a batch of files residing in the same directory) is evident. This is achieved by supporting semi-automatic semantic annotation of source code. The notion of semi-automatic comprises a user-controlled process with the system acting as a mediator (proposing semantic annotation concepts/taxonomies of concepts) and the user having the final decision of acceptance or rejection or revision and acceptance of the system's propositions. Thus, the time consuming task of manually annotating source code can be significantly decreased.

In order to achieve this goal in KnowBench ontology learning (Maedche & Staab, 2004) and information extraction techniques (D. H. Cunningham, Maynard, Bontcheva, & Tablan, 2002) were examined. Ontology learning is needed in order to extract ontology concepts from source code corpora that can be used for annotating it. On the other hand, information extraction is needed in order to instantiate the annotation ontologies with individuals, either of newly proposed and created or existing ontology concepts.

Developing ontologies by hand is clearly a tiresome task and its accuracy and maintainability can not be guaranteed. Researchers strive to address effectively such kind of knowledge bottleneck problems in ontology engineering. There is a lot of ongoing research in the area of (semi-) automatic ontology development, i.e. ontology learning. Ontology learning is concerned with knowledge acquisition and in the context of KnowBench the focus is on knowledge acquisition from text corpora and especially from source code corpora. Towards that goal, the need to adopt automatic or semi-automatic tools to build ontologies is obviously imperative as well as the need to construct respective systemic frameworks in order to support multiple iterations of ontology learning and ensure their consistency.

The ontology development task is a complex process. According to (P. Cimiano, 2006), in order to be able to estimate the state-of-the-art in ontology learning, the ontology development task should be broken down into subtasks that together constitute the complex task of ontology development (either manual or with any level of automatic support). The first fundamental subtask comprises extracting terms about the domain of interest. Then ontology development is primarily concerned with the definition of concepts and relations between them, but always keeping a connection to the terms and the knowledge that is used to refer to them. In other words this implies the acquisition of linguistic knowledge about the terms that are used to refer to a specific concept in text and possible synonyms of these terms. An ontology further consists of a taxonomy backbone (is-a relation) and other, non-hierarchical relations. Finally, in order to derive also facts that are not explicitly encoded by the ontology but could be derived from it, also rules should be defined (and if possible acquired) that allow for such derivations.

All of these aspects of ontology development can be organized in a layer cake of increasingly complex subtasks, as illustrated in Figure 4.3 (derived from (P. Cimiano, 2006)).

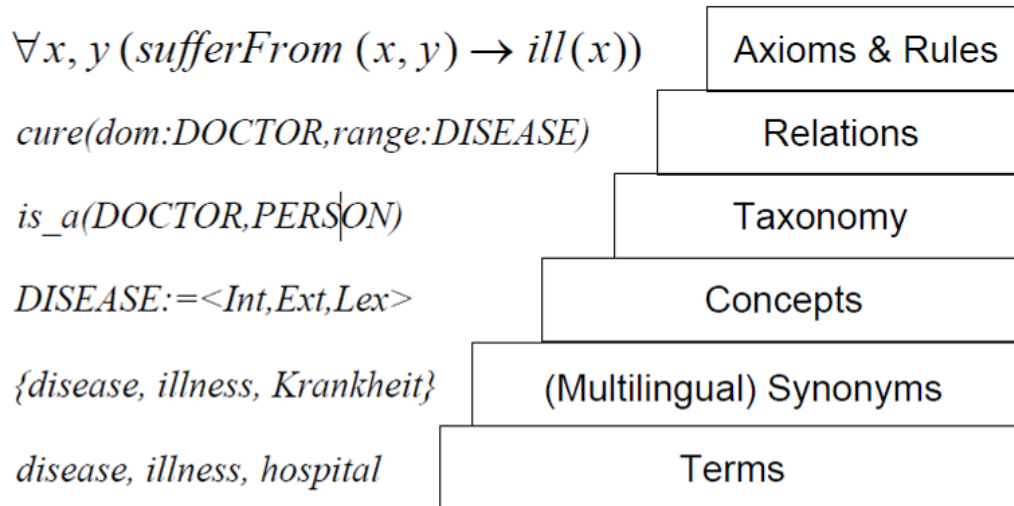


Figure 4.3: The ontology learning layer cake

Several ontology learning frameworks have been designed and implemented in the last decade. The Mo’K workbench (Bisson, Nédellec, & Canamero, 2000), for instance, basically relies on unsupervised machine learning methods to induce concept hierarchies from text collections. OntoLT (Buitelaar, Olejnik, & Sintek, 2003) is an ontology learning plug-in for the Protege ontology editor. It is targeted more at end users and heavily relies on linguistic analysis. The framework by Velardi et al., OntoLearn (Velardi, Navigli, Cuchiarelli, & Neri, 2005), mainly focuses on the problem of word sense disambiguation, i.e. of finding the correct sense of a word with respect to a general ontology or lexical database. TextToOnto (Maedche & Staab, 2004) is a framework implementing a variety of algorithms for diverse ontology learning subtasks. In particular, it implements diverse relevance measures for term extraction, different algorithms for taxonomy construction as well as techniques for learning relations between concepts (Maedche & Staab, 2000). The focus of TextToOnto has been so far on the algorithmic backbone with the result that the combination of different algorithms as well as the interaction with the user had been neglected so far. The successor (in 2005) Text2Onto targets exactly these issues by introducing the POM as a container for the results of different algorithms as well as adding probabilities to the learned structures to facilitate the interaction with the user. In the rest of this section Text2Onto is presented as well as the reasons for selecting it for implementing the specific functionality in KnowBench.

In general, Text2Onto is a framework for data-driven change discovery by incremental ontology learning. It uses natural language processing and text mining techniques in order to extract an ontology from text and provides support for the adaptation of the ontology over time as documents are added or removed. Explicit modelling of all kinds of changes and an

explanation component guarantee maximum transparency and traceability of the ontology learning process.

The architecture of Text2Onto is centred around the Probabilistic Ontology Model which stores the results of the different ontology learning algorithms. A Probabilistic Ontology Model (POM) as used by Text2Onto is a collection of instantiated modelling primitives which are independent of a concrete ontology representation language. The obvious benefits of defining primitives in such a declarative way are twofold. On the one hand, adding new primitives does not imply changing the underlying framework thus making it flexible and extensible. On the other hand, the instantiated primitives can be translated into any knowledge representation language given that the expressivity of the primitives does not exceed the expressivity of this target language. The modelling primitives used in Text2Onto are provided just below. The name of the corresponding primitive of Gruber's Frame Ontology is shown in parenthesis where applicable:

- concepts (CLASS)
- concept inheritance (SUBCLASS-OF)
- concept instantiation (INSTANCE-OF)
- properties/relations (RELATION)
- domain and range restrictions (DOMAIN/RANGE)
- mereological relations
- equivalence

As far as Natural Language Processing is concerned, many existing ontology learning environments focus either on pure machine learning techniques (Bisson et al., 2000) or rely on linguistic analysis (Buitelaar et al., 2003), (Velardi et al., 2005) in order to extract ontologies from natural language text (information extraction). Text2Onto combines machine learning approaches with basic linguistic processing such as tokenization or lemmatizing and shallow parsing. Since it is based on the GATE framework (D. H. Cunningham et al., 2002) it is very flexible with respect to the set of linguistic algorithms used, i.e. the underlying GATE application can be freely configured by replacing existing algorithms or adding new ones such as a deep parser if required.

In KnowBench the first two modelling primitives are supported, i.e. concepts and concept inheritance. In the respective modelling primitives Text2Onto employs different algorithms in order to calculate the probability for an instantiated modelling primitive. In particular the following measures are calculated:

Relative Term Frequency (RTF), TFIDF (Term Frequency Inverted Document Frequency), Entropy and the C-value/NC-value method in (Philipp Cimiano & Völker, 2005). For each term, the values of these measures are normalized into the interval [0..1] and used as corresponding probability in the POM.

Regarding subclass-of relations, in Text2Onto various algorithms have been implemented using different kinds of sources and techniques following the approach in (P. Cimiano, Pivk, Schmidt-Thieme, & Staab, 2004). In particular, algorithms exploiting the hypernym structure of WordNet (Fellbaum, 1998), matching Hearst patterns (Hearst, 1992) in the corpus as well as in the World Wide Web and applying linguistic heuristics mentioned in (Velardi et al., 2005). The results of the different algorithms are then combined through combination strategies as described in (P. Cimiano et al., 2004).

Moreover, in addition to the aforementioned core functionality of Text2Onto a graphical user interface featuring different views for the configuration of the ontology learning process and the presentation of the results is available. Once, an ontology has been extracted from the corpus the different modelling primitives are displayed to the user, who can interact with the POM by giving feedback to individual learning results. However, for the purpose of KnowBench a customized user interface was developed for controlling the parameters of the semi-automatic semantic annotation functionality.

Finally, in addition to the graphical user interface, Text2Onto features a java-based API which provides users and developers with programmatic access to the complete functionality of the ontology learning framework. This programming interface allows for integrating Text2Onto in other software applications and was used to wrap up Text2Onto into KnowBench. Furthermore, the selection of several Text2Onto algorithms is allowed to be used as well as different combinations which match different needs of each software developer while deriving annotations.

Concluding, in the context of KnowBench and for the purpose of realizing a semi-automatic semantic annotation of source code functionality in KnowBench the work was based on Text2Onto. The main reasons for doing so were:

1. The API documentation was really detailed and comprehensible thus the extension/modification of Text2Onto was greatly facilitated.
2. All previously developed ontology learning frameworks lack an explanation component helping the user to understand why something has changed in the underlying POM.
3. Most tools do not indicate how certain a learned object actually is, thus making it more difficult for the user to select only the most reliable findings of the system.
4. The results of different algorithms can be combined in order to increase the preciseness of the learned structures and thus providing better reliability.

5.3.8 Knowledge Base Graph-based Browsing

Knowledge base graph-based browsing facilitates knowledge base browsing using graphs and provides an alternative choice to the use of the semantic wiki or the knowledge base editor by augmenting the available means the software developer has to visualize the knowledge base. The KB graph browser provides several advanced features such as zooming in/out, animation, filtering, auto-layout, expansion, collapse of nodes, etc.

This functionality is provided as an independent standalone RCP application (however the metadata store component is required to run the standalone version) as well as an Eclipse plug-in.

An overview of the provided functionality by this KnowBench sub-component is given below:

Visualization of concepts, instances and their relationships using graph nodes (different kind of nodes for concepts or instances) and arrows respectively

Zooming in/out the whole graph

Rotation of the graph by 90, 180, 270 degrees

Large/small icons of the graph nodes

Smooth floating animation of the graph while dragging around a node to focus on

Auto-layout of the graph when it is moved by dragging a node

Filtering of all instance or concept nodes

Expansion/collapse of concept nodes will show/hide instances that do not have outgoing semantic links to other instances (object property instantiations)

Legend of the graph nodes (i.e. concepts and instances)

Concept clusters indicate how much “loaded” is a concept in terms of its instances and in comparison with the overall KB load – this is indicated with a 0 to 10 scale

Tooltips on every node indicating its name and type (concept or instance)

6 TECHNICAL IMPLEMENTATION OF KNOWBENCH

The KnowBench system is designed in a modular fashion meaning that several components correspond to different kind of functionality (e.g. search, semantic wiki, annotation, Semi-automatic annotation, etc.). The benefits from such a modular design are obvious:

More flexible and less effort required for the source code maintenance – interfaces are defined between the KnowBench components which makes it easy for a future maintenance release to focus only on the desired functionality (i.e. if a future release targets better visualization of the annotation mechanism, only the annotation plug-in has to be updated leaving the overall KnowBench's behaviour intact).

All functionality can be modularized as a result, taking its respective GUI along – this is very important in cases that users of KnowBench want to use specific functionality of the software and configure only the desired features that they want to have. For example, they can decide to use only the annotation and search functionality of the KnowBench system without search, semantic wiki, etc.

New functionality can be developed easily and in an autonomous way as an Eclipse plug-in using Eclipse PDE⁴⁹ and then integrated into KnowBench with minimum effort – the components resulting out of this conceptual model are implemented as separate Eclipse plug-ins, all connecting to a thin core model of the KnowBench, which is lightweight and acts as a dynamic plug-in manager of the KnowBench. New implemented plug-ins have to conform to a minimal set of certain conventions defined in this core model.

The design of KnowBench is generic enough so that it can be reused in any target development environment as long as the latest uses Object Oriented Programming languages and technologies (e.g. Java, Visual Basic, C++, C#, etc.) such as Microsoft .NET Framework⁵⁰, Visual Studio⁵¹, IntelliJ IDEA⁵², Borland JBuilder⁵³, etc. This constraint is

⁴⁹ <http://www.eclipse.org/pde/>

⁵⁰ <http://www.microsoft.com/.NET/>

introduced due to the fact that Object Oriented Analysis and Design Using UML⁵⁴ was used. For an introduction to OOP and UML design visit <http://www.codeproject.com/KB/cpp/oopuml.aspx>. This kind of conceptual design gives the potential of implementing KnowBench in different kinds of platforms and porting to other environments, depending on the required application at hand. As a proof of concept an Eclipse IDE specific implementation will be developed using this generic conceptual design.

6.1 High-level KnowBench Technical Architecture

Figure 6.1 depicts the high level architecture of the KnowBench. In this high level architecture, the components of the KnowBench are shown:

- Semantic annotation API
- Shallow NLP and IE API (wrapping up and customizing Text2Onto (Philipp Cimiano & Völker, 2005))
- Knowledge base graph-based browser
- Knowledge base editor
- semantic wiki editor and renderer (DevWiki)
- Semantic search
- Global metadata store

The semantic annotation API is used for both manual and semi-automatic semantic annotation of source code and interacts with the global metadata store component to perform operations on the knowledge base. It is responsible for automated generation of annotation related metadata, when it is provided with an annotation concept and a respective source code element. Manual annotation is triggered by the software developer via his/her KnowBench using its source code editor to annotate a specific source code fragment.

⁵¹ <http://msdn.microsoft.com/en-us/vstudio/default.aspx>

⁵² <http://www.jetbrains.com/idea/>

⁵³ <http://www.codegear.com/products/jbuilder>

On the other hand, semi-automatic annotation uses a source code corpora as input in order to extract new annotation concepts and if feasible their classification inside the annotation ontology hierarchy tree. The output of the semi-automatic annotation is then passed to the semantic annotation API to add the actual annotation metadata inside the knowledge base in the same manner as in the manual annotation process. The shallow natural language processing (NLP) and information extraction (IE) API is used to achieve the purpose of annotating source code corpora in a semi-automatic supervised by the software developer procedure. It is build on top of the Text2Onto API⁵⁵ and provides a wrapper with customizations (such as java keyword stop-words which are ignored, e.g. import, package for, while, etc.) which adapt it to the software development domain since Text2Onto performs information extraction in generic document corpora. The usage of stop-words was necessary in order for the system to ignore java keywords and not to propose them as new annotation concepts, taking into consideration that in a typical source code file these keywords appear in multiple occurrences and do not refer to its domain or software engineering entity concepts.

The knowledge base graph-based browser interacts with the global metadata store component in order to visualize the knowledge base by exploiting the graph metaphor. Likewise, the knowledge base editor, semantic wiki editor and renderer all communicate with the global metadata store component in order to operate directly on the knowledge base. The difference between the KB graph browser, semantic wiki renderer and the knowledge base and semantic wiki editors is that the first two only fetch information from the knowledge base, as opposed to the rest which perform additional operations such as creating, updating/modifying and removing knowledge base items (removal is supported only in KB editor).

KnowBench communicates with the semantic search API in order to fetch knowledge. The semantic search API in turn uses the global metadata store to retrieve this knowledge located either locally or in the P2P network.

⁵⁴ <http://www.ratio.co.uk/W1.html>

⁵⁵ http://ontoware.org/frs/?group_id=14

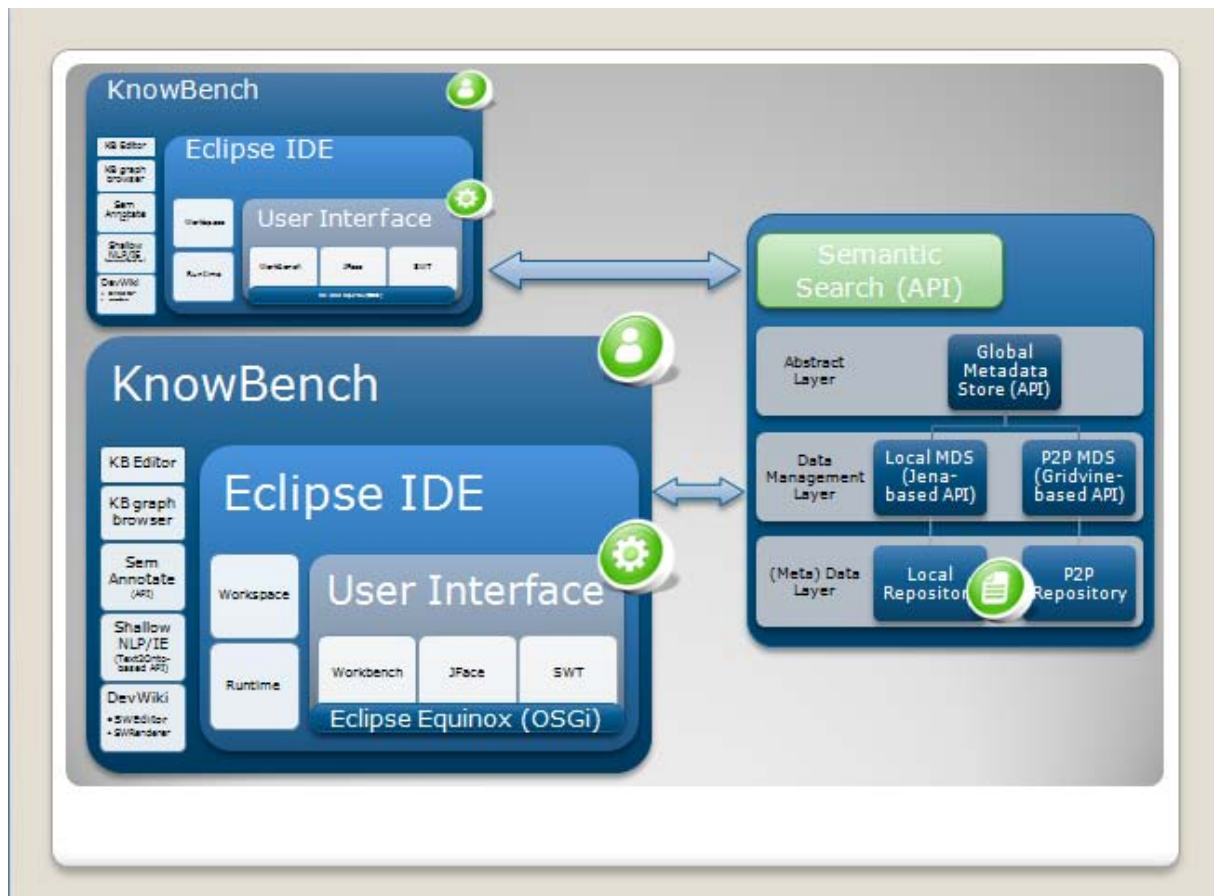


Figure 5.1: KnowBench high-level Architectural Diagram

The set of Eclipse plug-in components which compose KnowBench is given in the list below:

org.knowbench.eu.annotation

org.knowbench.eu.annotation.semiauto

org.knowbench.eu.commons

org.knowbench.eu.core

org.knowbench.eu.kbeditor

org.knowbench.eu.kbgraph

org.knowbench.eu.kbgraph.plugin

org.knowbench.eu.metadata

org.knowbench.eu.metadata.p2p

org.knowbench.eu.model

org.knowbench.eu.search

org.knowbench.eu.widgets

org.knowbench.eu.wiki

In the rest of this chapter a short implementation description of each KnowBench component is given.

6.2 Core Component

The Core component serves as a thin and lightweight kernel and plug-in manager for every other KnowBench component (plug-in). All other plug-ins connect to the core component and adhere to a certain minimal set of conventions. Each time a new plug-in is required some specific interfaces have to be implemented.

The Core component consists of the following classes:

Activator - The activator class controls the plug-in life cycle

Constants – Constants used in the core component and its dependencies

UriHelper – Class containing methods for creating well-formed URIs as well as managing them

IActivable - This interface should be implemented by any activator that wishes to use `InitHelper#ensureIsActive`

InitHelper - This class provides the mechanism to allow other bundles to activate OSGi services

KnowBenchActivator – Abstract base class to be extended by other plugin Activators

KnowbenchPreference - This class represents a preference page that is contributed to the Preferences dialog. By subclassing `FieldEditorPreferencePage`, the field support built into `JFace` can be used that allows creating a page that is small and knows how to save, restore and apply itself. This page is used to modify preferences only. They are stored in the preference store that belongs to the main plug-in class. That way, preferences can be accessed directly via the preference store.

AbstractServiceProvider – Abstract base class that handles OSGi services

IServiceProvider – Extension interface of the IActivable

Services - This enumeration provides keys for all the services the KnowBench handles separately. The general contract is to have a two letter prefix specifying the bundle and the name of the service follows.

6.3 Model Component

The Model component is used to model ontology concepts to UI elements following the Model-Viewer-Controller (MVC) paradigm. It facilitates the transformation of ontology related knowledge to visualized UI elements in the KnowBench.

The Model component consists of the following classes:

OntologyRuleDescriptor – Describes P2P policy ontology rules

OntologyTreeNodeDescriptor – Describes an ontology tree node

6.4 Commons Component

This component provides access to libraries used commonly by other KnowBench sub-components. The main advantage of this kind of design is that the maintenance of the common libraries is central and in case of upgrade, it is done only once while in the contrary having to maintain multiple copies of the libraries in every component that uses them might raise inconsistencies and extra effort for maintenance.

6.5 Annotation/Annotation.semiauto Components

These two components provide the semantic annotation API as well as the related UI for manual and semi-automatic semantic annotation of source code files.

They consist of the following classes:

Activator - The activator class controls the plug-in life cycle

AnnotationDialog – Manual annotation popup dialog

ProcessDocuments – Processes source code documents for semi-automatic annotation

`SemiAutomaticAnnotationEditor` – Semi-automatic annotation editor which allows the modification and selection of the annotations proposed by the semi-automatic annotation engine to be added into the `KnowBench` knowledge base

`SemiAnnotateAction` – The action contributed to the set of available Eclipse actions, which is associated with the semi-automatic annotation process

`AddAnnotationAction` - The action contributed to the set of available Eclipse actions, which is associated with the addition of a manual annotation

`RemoveAnnotationAction` - The action contributed to the set of available Eclipse actions, which is associated with the removal of an annotation

`ViewAnnotationsAction` - The action contributed to the set of available Eclipse actions, which is associated with the view of annotations

`AnnotationsView` – The annotation view presenting the annotations of the current source code file

6.6 Kbeditor Component

This component provides the UI for the knowledge base editor functionality of the `KnowBench` which facilitates the creation, browsing, modification and deletion of knowledge base instances. It interfaces with the metadata store API to perform operations on the knowledge base.

`Activator` - The activator class controls the plug-in life cycle

`KBAction` – The action contributed to the set of available Eclipse actions, which is associated with the knowledge base editor

`KBEditor` – The knowledge base editor

`AddInstanceComposite` – The add instance form of the knowledge base editor

`ModifyInstanceComposite` – The remove instance form of the knowledge base editor

`ObjectPropertyComposite` – A utility widget used for assisting the modification/addition of an object property

6.7 Kbgraph Component

This component facilitates the knowledge-base graph browsing functionality of the KnowBench which assists in navigating the knowledge base in a graphical representation fashion using the graph metaphor. It is designed as an independent and standalone application that runs without the whole KnowBench installation.

The Kbgraph component consists of the following classes:

Activator - The activator class controls the plug-in life cycle

ApplicationActionBarAdvisor – Application action bar advisor

ApplicationWorkbenchAdvisor – Application workbench advisor

ApplicationWorkbenchWindowAdvisor – Application workbench window advisor

KBGraphApplication – The knowledge base graph-based browser application

KBGraphPerspective – The knowledge base graph-based browser perspective

KBGraphView – The knowledge base graph-based browser view inside the above perspective

KBGraphModel – The knowledge base graph model

ClassNode – Node representing an ontology concept inside the graph representation

InstanceNode - Node representing an instance inside the graph representation

KBNode - Node representing the knowledge base inside the graph representation

6.8 Kbgraph.plugin Component

This component provides the same functionality as the Kbgraph component described in Section 6.7 but as an Eclipse plug-in available through the KnowBench.

The Kbgraph.plugin component consists of the following classes:

Activator - The activator class controls the plug-in life cycle

KBgraphAction - The action contributed to the set of available Eclipse actions, which is associated with the knowledge base graph-based browser

KBGraphBrowser – The knowledge base graph-based browser

Graph – Class representing the graph

6.9 Global Metadata Store Component

The global metadata store consists of two components (APIs) – the LocalMDS and P2PMDS and two managers – KeyStore Manager and Policy Manager. It provides an abstract layer for handling these and its purpose is to manage knowledge stored either locally or in the P2P network.

The P2PMDS is based on the GridVine/P-Grid system (Aberer, Datta, Hauswirth, & R. Schmidt, 2005) and is customized for KnowBench in order to realize an access control aware P2P metadata store using the services of KeyStore and Policy managers.

The KeyStore Manager is responsible for providing up-to-date status of various P2P groups and the corresponding credentials. The P-Grid secured communication infrastructure needs a set of Certificate Authority's credentials to process various certificates passed to peers in regular communication with other peers.

The Policy Manager provides services to configure and update the access control policy. The developer defines what is shared to whom through the P2P repository by assigning certain policies to different types of knowledge using criteria and the desired developer groups where this knowledge is to be shared. Examples of these criteria could be: “problems that are written by John”, “solutions that are solving problem X”, etc. The criteria are built by constraining properties of ontology concepts to certain values, thus allowing the possibility to create complex rules about what stays inside the local repository or is shared to other peers.

The LocalMDS is built on top of the well known open-source Jena framework in order to provide a handy API for handling the local knowledge base. The main reason for abstracting Jena to KnowBench and building a new API on top of it is that bulk operation was necessary (executing multiple knowledge base transactions at once) and easier/more powerful handling as well as aggregated operations that are not provided by Jena (Jena methods are of high granularity for the purposes of KnowBench – the encapsulation of several Jena methods was necessary in order to provide a higher level of functionality to KnowBench API calls).

6.9.1 LocalMDS

This component provides an entry point to all of the rest KnowBench components in order to access the metadata store API. It serves as a mediator between KnowBench and the knowledge base. It is introduced in the conceptual model since it is necessary to have a central place where the related metadata store services are instantiated through the OSGi mechanism. Additionally, it provides metadata store related UI factories for other KnowBench components such as the visualization of certain knowledge inside the knowledge base e.g. concepts, properties and serves as the “Viewer” in the MVC paradigm.

The LocalMDS component consists of the following classes:

Activator - The activator class controls the plug-in life cycle

OntologyTreeFactory – Factory for building ontology tree hierarchies

PropertyValue - This class acts as a holding class for the values selected by a user in the StructuredQueryDialog

OntologyPropertiesUI – Class for displaying property data sheets needed for their restriction to specific values

OntologyWidget - This is a decorator for Control which allows the rest of the classes using the OntologyWidgetFactory to set and get values from the controls in a controlled and contracted manner

OntologyWidgetFactory – Factory for building ontology related widgets

PropertyWidget – Utility class for creating property widgets as well as manipulating data associated to property widgets

PropertyWidgetEvent - This is the event dispatched by any PropertyWidget that wishes to be removed from the list

PropertyWidgetListener - This listener is implemented by any class which wishes to act on the desire of a PropertyWidget to remove

6.9.2 P2PMDS

This component provides all the P2P related UI such as the ability to connect to a P2P network, publish knowledge or configure policy rules used for sharing knowledge.

The P2PMDS component consists of the following classes:

Activator - The activator class controls the plug-in life cycle

RuleDecorator – Rule decorator for the P2P rules preferences page

P2PPreferences – P2P preferences page

GroupsControl – Widget for controlling the groups associated with a P2P policy rule

PropertiesControl - Widget for controlling the property restrictions associated with a P2P policy rule

RulesControl - Widget for adding/removing the rules associated with a P2P policy rule set

IRuleSetListener – Rule set listener interface

RuleSetEvent – Rule set event

6.10 Search Component

This component provides all types of search (i.e. keyword, structured and semantic search) related UI such as the articulation of a query, viewing its results, refining queries and requesting further details on the desired retrieved knowledge item.

The Search component consists of the following classes:

Activator - The activator class controls the plug-in life cycle

SearchPage – The search page integrated into the Eclipse search

QuerySeviceProvider – Provides the query service of the search component to the KnowBench through OSGi

SemanticSearchSeviceProvider – Provides the semantic search service of the search API to the KnowBench through OSGi

NullStructuresValues - This class provides a holding place for values returned from the search composite

ResultValue - This immutable data type holds the information of a single search result

StructuresValues - This class provides a holding place for values returned from the search composite

SearchResultsView – Search results view displays the search result to the KnowBench UI

KeywordSearchComposite - This is the composite which provides the UI for the keyword search functionality in the KnowBench

MySearchHolder – This class came out of the necessity to create a composite which disabled the setLayoutData method since it is being overridden in the SearchDialog class

ResultComposite – Result composite holds a single search result inside the corresponding search view

StructuredSearchComposite – This composite provides the UI for the structured query search

KnowbenchSearchComposite - This is the master composite which holds both KeywordSearchComposite and StructuredSearchComposite, and displays them in a tab folder

6.11 Widgets Component

The Widgets component provides a reusable widget factory for all KnowBench components, actually being a utility component. It provides several widgets that are needed in more than one KnowBench components.

The Widgets component consists of the following classes:

DefaultSelectedOntologyValue – Provides the default selected ontology value

SelectedOntologyValue – Interface for the selected ontology value

OpenWebBrowser – Opens an external web browser

ResourceProvider - This provides an access point to the images in the bundle

SWTUtils – SWT utility class

Activator - The activator class controls the plug-in life cycle

OntologyFilterTreeControl - This widget is inspired from the Outline View of Eclipse. All the viewer methods are delegated to an underlying TreeViewer, this code just encapsulates the filter text, and its implementation

OntologyLabelProvider – Provides labels for ontology concepts

OntologyTreeCombo – Ontology tree hierarchy combo box

OntologyTreeControl – Ontology tree control widget

6.12 Wiki Component

The Wiki component provides the semantic wiki functionality of the KnowBench. It is a crucial part of the whole KnowBench system as it serves as a flexible knowledge articulation facility assisting in software development knowledge, navigation and preservation.

The Wiki component consists of the following classes:

Activator - The activator class controls the plug-in life cycle

WikiOpenEditorAction - This action is exposed by the KnowBench plugin. It allows any component to open a wiki editor

ColorManager – This class creates an internal cache of colours, which will be seamlessly returned when asked for

NonRuleBasedDamagerRepairer – Non-rule based damager repairer class

KnowBenchDocumentProvider – KnowBench document provider class

KnowBenchEditorPartitionScanner – KnowBench editor partition scanner class

KnowBenchEditorTagScanner – KnowBench editor tag scanner class

KnowBenchEditorWhitespaceDetector – KnowBench editor whitespace detector class

WikiBrowser – Wiki browser class

WikiEditor – Wiki editor class

WikiEditorConfiguration – Wiki editor configuration class

WikiMultipleEditor – Wiki multiple editor class encapsulating wiki editor and browser

NullValidator – Null validator class

WikiOntologyAssistProcessor – Wiki ontology assist processor used for auto-complete facilities

WikiMr – This class provides the connection for the wiki representation of an ontology instance. It will delegate most work to the metadata store API, but it contains some helpful methods, for extracting information from wiki representation

WikiMrAutocompletionConnector - The purpose of this class is to connect the metadata store API to the editor so that auto-completion can be provided

WikiEditorInput – Wiki editor input class

WikiEditorInputFactory – Wiki editor input factory class

WikiStorage – On-the-fly wiki storage mechanism

FirstLineRegexLabelRule – First line regular expression label rule class

RegexLabelRule – Regular expression label rule class

TextSequenceRule - An implementation of IRule capable of detecting words. Word rules also allow for the association of tokens with specific words. That is, not only can the rule be used to provide tokens for exact matches, but also for the generalized notion of a word in the context in which it is used. A word rules uses a word detector to determine what a word is

RulesStore – A class facilitating an in-memory store for rules

Tokens – Tokens enumeration

TokenStore - A class facilitating an in-memory store for tokens

AbstractTextRegionMatcher – Abstract text region matcher class

ColourManager – Colour manager class

PatternMatcher – Matches a region of text using a regular expression

ConceptWikiValue – Concept wiki value utility class

LabelWikiValue - Label wiki value utility class

PropertyObjectWikiValue – Object property wiki value utility class

WikiException – Wiki exceptions are described in this class

WikiOntRepresentationValue – Ontological wiki representation values class

MD5 – Utility class

WikiOntologyExtractor – This class does the same as the partitioner, except that it is simpler, and used for a very specific purpose. Extracts information which will be used to add or update an instance into the model

7 KNOWBENCH WALKTHROUGH

In this chapter the usage of the KnowBench system is described. Since the KnowBench system is a knowledge management system for software developers, this section is organised according to the knowledge management cycle.

7.1 Overview

The KnowBench is tightly integrated into Eclipse developing environment. Thus, its actions, menus and preferences are located in the respective locations within the working environment. More specific, the KnowBench functionalities are embedded in:

1. Toolbar consisting of several buttons (see left -upper part on Figure 7.1);
2. KnowBench menu in the Eclipse menu (see right-upper part on Figure 7.1);
3. Eclipse Preferences dialog (see Figure 7.2);
4. Eclipse Search dialog (see Figure 7.3);
5. Eclipse Perspective (see Figure 7.4);
6. Eclipse status information (see Figure 7.5).

Toolbar

The software developer is able to use the toolbar of KnowBench in order to access the functionality of the system (Figure 7.1).

Menu

The menu provides another access point besides the toolbar for the KnowBench functionality (Figure 7.1).

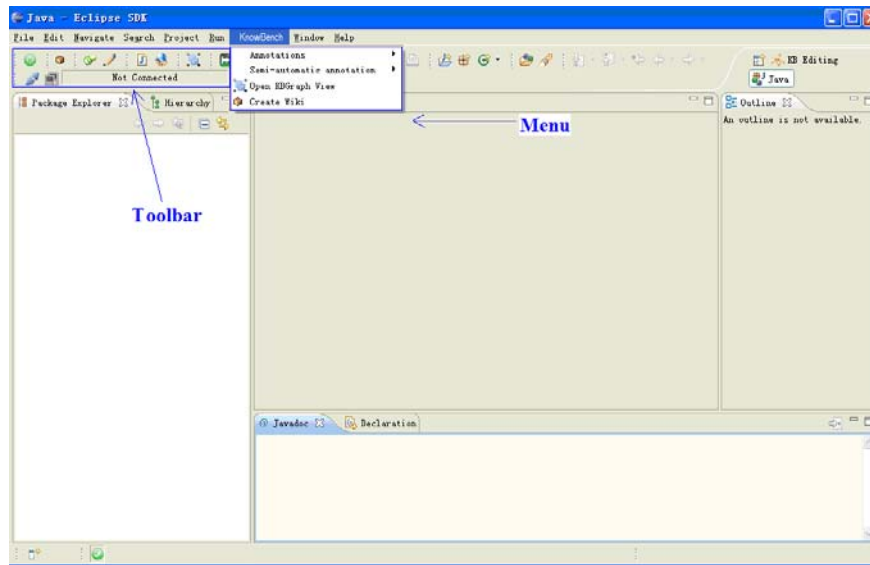


Figure 6.1: KnowBench Menu and Toolbar Eclipse Integration

Preferences

The software developer is able to configure the KnowBench system via the standard Eclipse preferences dialog (Figure 7.2). The user can configure the following:

1. P2P and
2. Search
 - a. Crawl Repository

See section 7.2 for details.

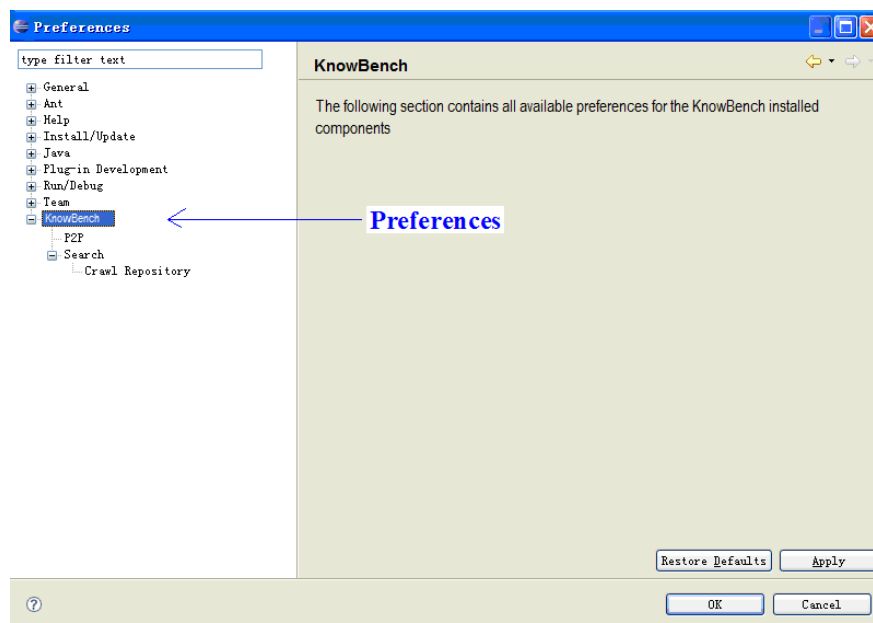


Figure 6.2: KnowBench Eclipse Preferences Integration

Eclipse search integration

The KnowBench search is integrated inside the standard search dialog of Eclipse (Figure 7.3).

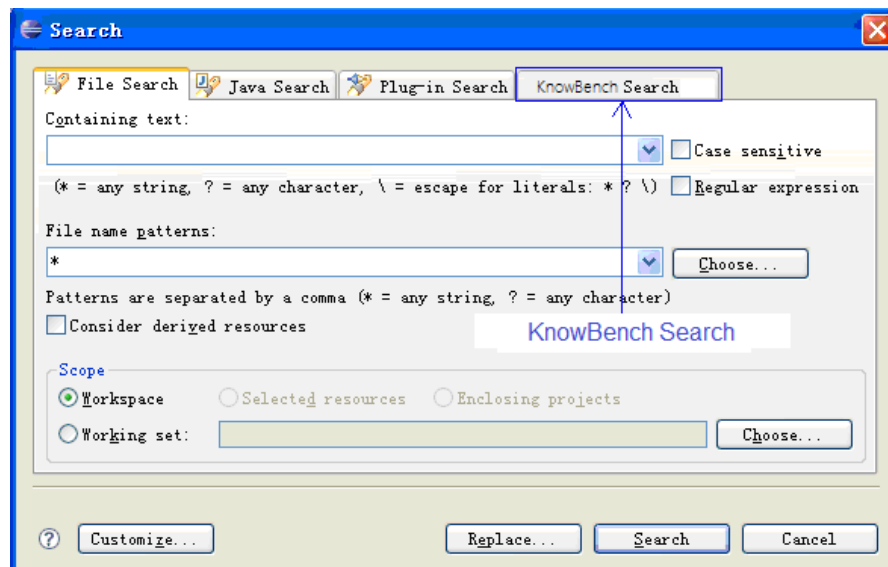


Figure 6.3: KnowBench Eclipse Search Integration

KnowBench perspective

The developer is able to switch to various KnowBench perspectives (Figure 7.4) in order to see the available KnowBench views in which he/she can interact with the system.

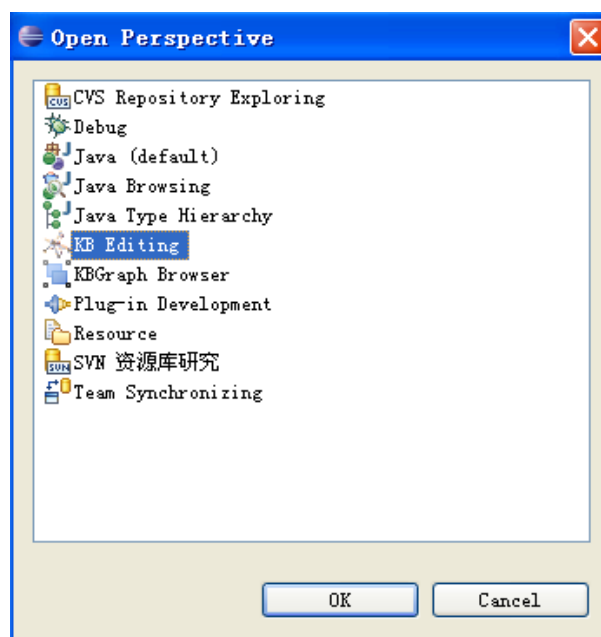


Figure 6.4: KnowBench Eclipse Perspective Integration

KnowBench status

The developer can view the status of the system by clicking on the respective icon (Figure 7.5). He/she can see for example, the status of the P2P network.

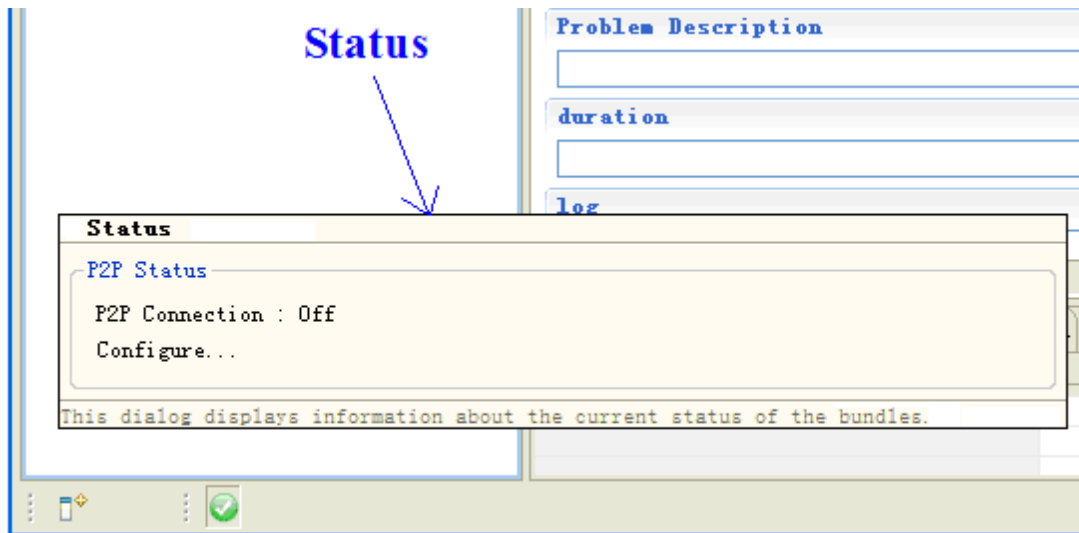


Figure 6.5: KnowBench Eclipse Status Integration

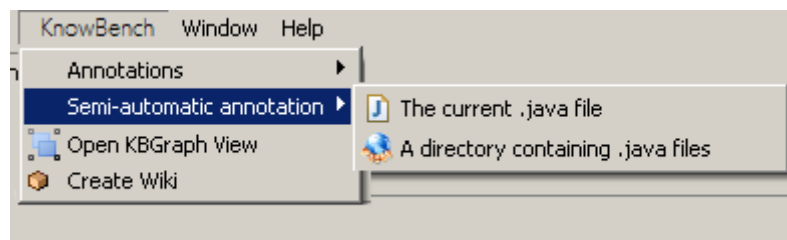
Depending on which functionalities are installed, some of the toolbar buttons (see Figure 7.6, Table 7.1) and menu items (see Figure 7.7) will not be present. All toolbar button functionalities can also be found in the menu.



Figure 6.6: KnowBench Toolbar

Toolbar button	Description
	Create Wiki
	Semi-automatic annotation of the current .java file
	Semi-automatic annotation of source code corpora contained in a directory
	Open graph-based representation of the knowledge base
	Join a KnowBench Network / Disconnect from the network
	Buttons indicating publishing status: not active./Stop Publishing/ Start Publishing

Not Connected Connected Connected and Publishing	Status buttons indicating different operation done over the P2P network: (1) not connected to the network, (2) connection established and (3) connection established and knowledge published
	View or remove annotations of the current source file
	Add a new annotation of a code fragment

Table 6.1: Description of KnowBench toolbar**Figure 6.7: KnowBench Menu**

7.2 Configuration

The KnowBench system can be configured using the Preferences dialog, which can be started from the Window menu as shown in Figure 7.8.

**Figure 6.8: Eclipse Preferences**

KnowBench configuration settings are divided in sections. As already shown in Figure 7.2 configurations sections are:

3. P2P and

4. Search

a. Crawl Repository

These options are subsequently described.

7.2.1 Peer-to-Peer Configuration

In the KnowBench system, the user can control the access to information which is stored in a P2P network. In particular, as in the KnowBench system information on the P2P network is stored in the form of RDF triples, the KnowBench system controls access to RDF triples. In this context, access restrictions means that the data is published in the P2P network in such a form that only the authorized users can access it.

The access control part of the KnowBench system allows users to publish the resources *selectively* to a certain set of users in the network. When users search the P2P network, they should get the results which are only accessible to them.

The KnowBench system only supports READ access rights; WRITE and UPDATE rights are not considered here. Only positive access rights are described by the policy language. A resource is only accessible for users who can present a credential explicitly listed in the policy.

The resources for which access rights can be defined are ontology instances. The KnowBench system also enables to consider a group of instances, which can be further constrained by (one or more) conditions on the class properties (e.g. all the instances of Problem class with a property “isIdentifiedBy” set to a particular user’s name). This means that the developers are able to specify a publishing criterion in configuration files, where in, the system automatically detects new instances in the local store which meet this criterion and publishes all such instances.

A typical access control policy looks like: “All the problems with severity 1 are published so as to be accessible to every user of the network”.

This policy file⁵⁶ can be managed using the KnowBench Peer-to-peer preferences in the Eclipse environment (see Figure 7.9). It should be noted here that in the case that a user is not connected to a peer-to-peer network, the form on the right-hand side in Figure 7.9 will be empty. It means that in order to see the P2P publish policies, the user needs to be connected to a network.

A policy is a set of rules. Each rule contains an object and a (list of) subject(s). A rule expresses that READ access rights should be granted to the object for the holders of the credentials listed in the rule.

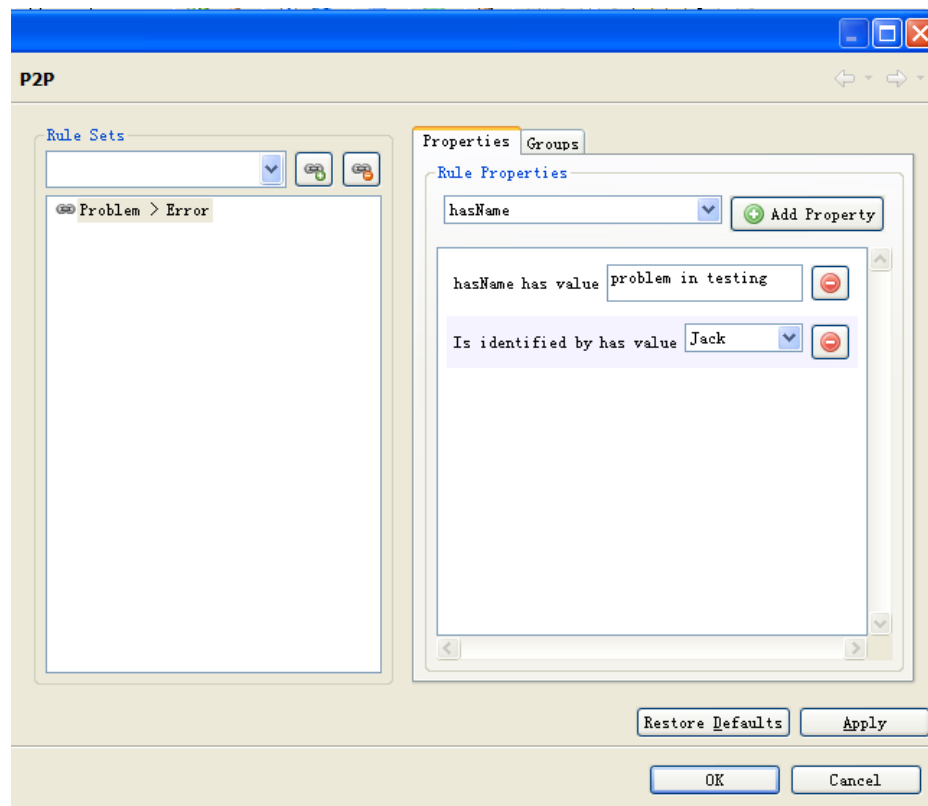


Figure 6.9: KnowBench P2P Preferences

In order to add a rule the user selects a class from the combo box⁵⁷ and presses the “Add Rule” button (see Figure 7.10). This will add the class to the “Rule Sets” list shown in Figure 7.11 and allow the user to make modifications to the rule which is below described. The user can also delete a selected rule by clicking the “Delete Rule” button.

⁵⁶ The actual implementation uses XML syntax for the policies.

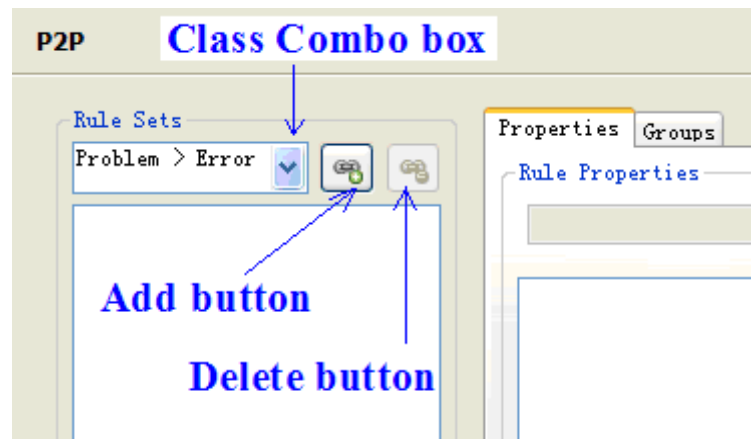


Figure 6.10: Adding a Rule to P2P Policy File

The example shown in Figure 7.10 means that all individuals of the class Error should be shared. To further specify what should be published to the network, the user can define constraints for the selected class. This can be done by selecting properties that should be considered as well as value pairs for those properties. All available properties for the selected class (in Rule Sets) are shown in Rule Properties combo box (see Figure 7.11).

To define a constraint for the rule, the user can add a property by selecting it in Rule Properties combo box and clicking on the Add Property button. The chosen property will appear in Property Value Pairs shown in Figure 7.11. The user should define values for each selected property or remove them.

⁵⁷ The combo box shows all classes defined in the KnowBench ontologies in a form of a tree.

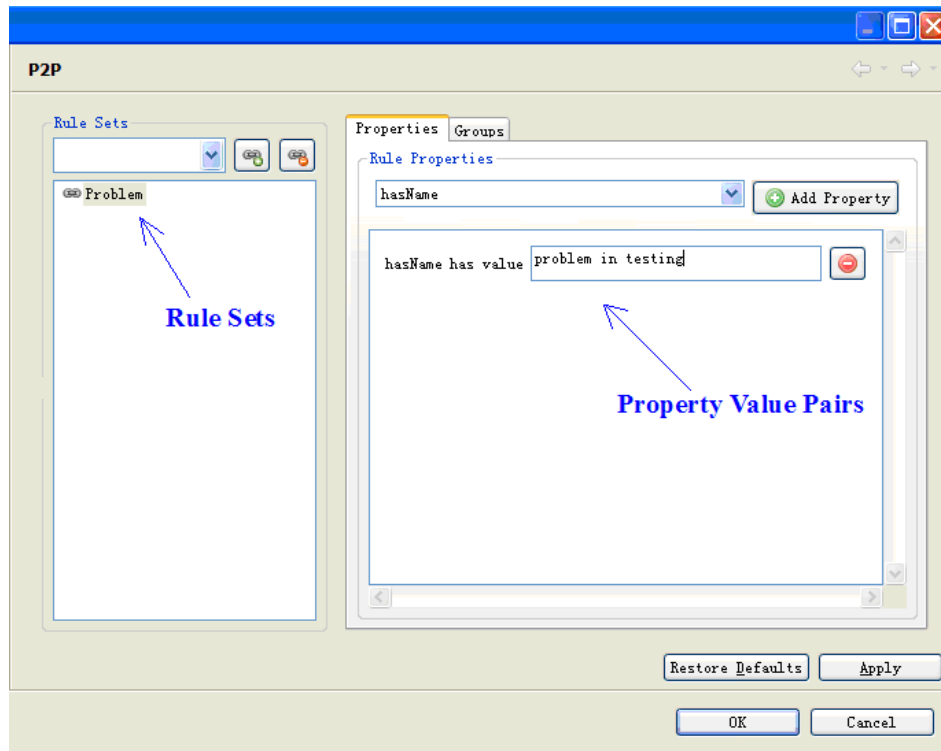


Figure 6.11: Adding Properties to P2P Rule

The example shown in Figure 7.11 should be interpreted as follows:

All problem individuals with the hasName property that contains “problem in testing” will be shared over the P2P network.

In the case that the selected property is an object property, the user should specify constraints by selecting an existing individual. As shown in Figure 7.12, this can be done by clicking the corresponding combo box.

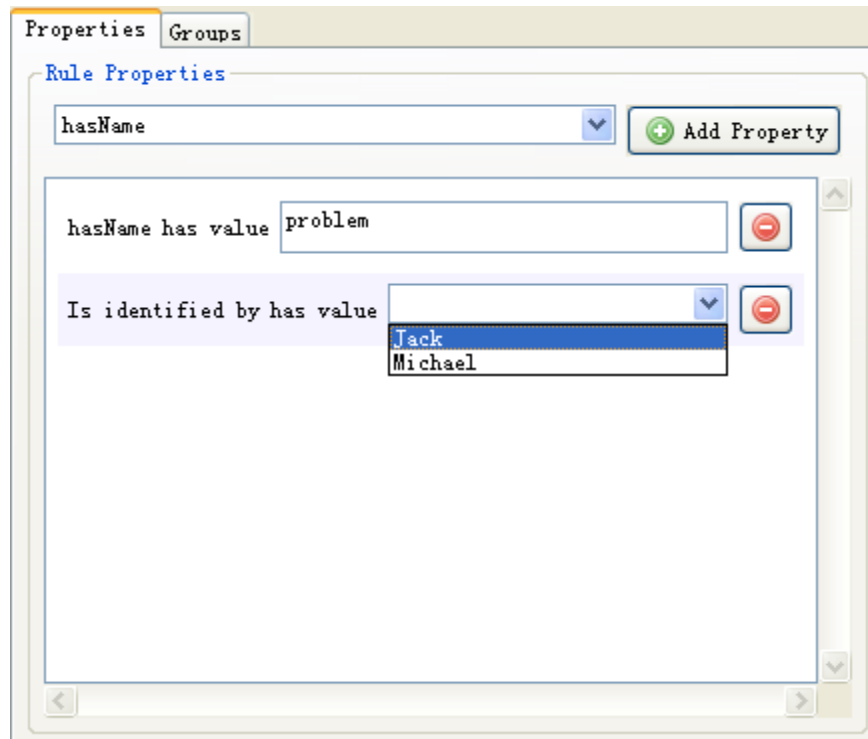


Figure 6.12: Adding Constraints for Object Properties of P2P Rule

A rule can also define the groups that this rule applies to. By accessing the “Groups” tab the user can add or remove available publish groups from the “Publish Groups” list. This is shown in Figure 7.13. In order to add a group, the user needs to select a group in the “Available Groups” list and then click on an arrow pointing toward the “Publish Groups” list. To remove a group user needs to select a group in the “Publish Groups” list and then click on an arrow pointing toward the “Available Groups” list.

There is a default group "Entire-KnowBench-Network", which means an open network. Anybody in the network can access data published to this group.

Creation of new groups is very simple as shown in Table 7.2. It should be noted here that the ontology data is not integrated with this data. The group creator can share the keys with any user to make him member of the group.

Generating a group-subject and its credentials

```
keytool58 -genkeypair -alias group1 -keystore KnowBenchKeyStore -storepass nram123 -keyalg RSA
What is your first and last name?
[Unknown]: KnowBench:n.ram@yahoo.co.in:Group1
What is the name of your organizational unit?
[Unknown]: EPFL
What is the name of your organization?
[Unknown]: EPFL
What is the name of your City or Locality?
[Unknown]: Lausanne
What is the name of your State or Province?
[Unknown]: Lausanne
What is the two-letter country code for this unit?
[Unknown]: CH
Is CN=KnowBench:n.ram@gmail.com:Group1, OU=EPFL, O=EPFL, L=Lausanne, ST=Lausanne, C=CH
correct?
[no]: yes
Enter key password for <group1>
(RETURN if same as keystore password):
```

Table 6.2: Generating a group-subject and its credentials

An arbitrary user willing to join a group must communicate with the group's administrator (the user who created such a group). Once he obtains the group-subject's credentials in the form of a keystore, he should import this subject's credentials to its own KnowBenchKeyStore.

It should be also noted that the used tools (e.g. keytool in Table 7.2) are independent of the KnowBench system. Users are encouraged to use their own tools to produce the credentials required by the subjects as long as they can be parsed by the KeyStore Manager.

⁵⁸ Keytool is distributed with Sun's Java's distribution so does not need any special installation. However, OpenSSL needs to be installed on only the peer which assumes CA's responsibilities. It is assumed that in a collaborative system, independent autonomous bodies exist which can run their own CA.

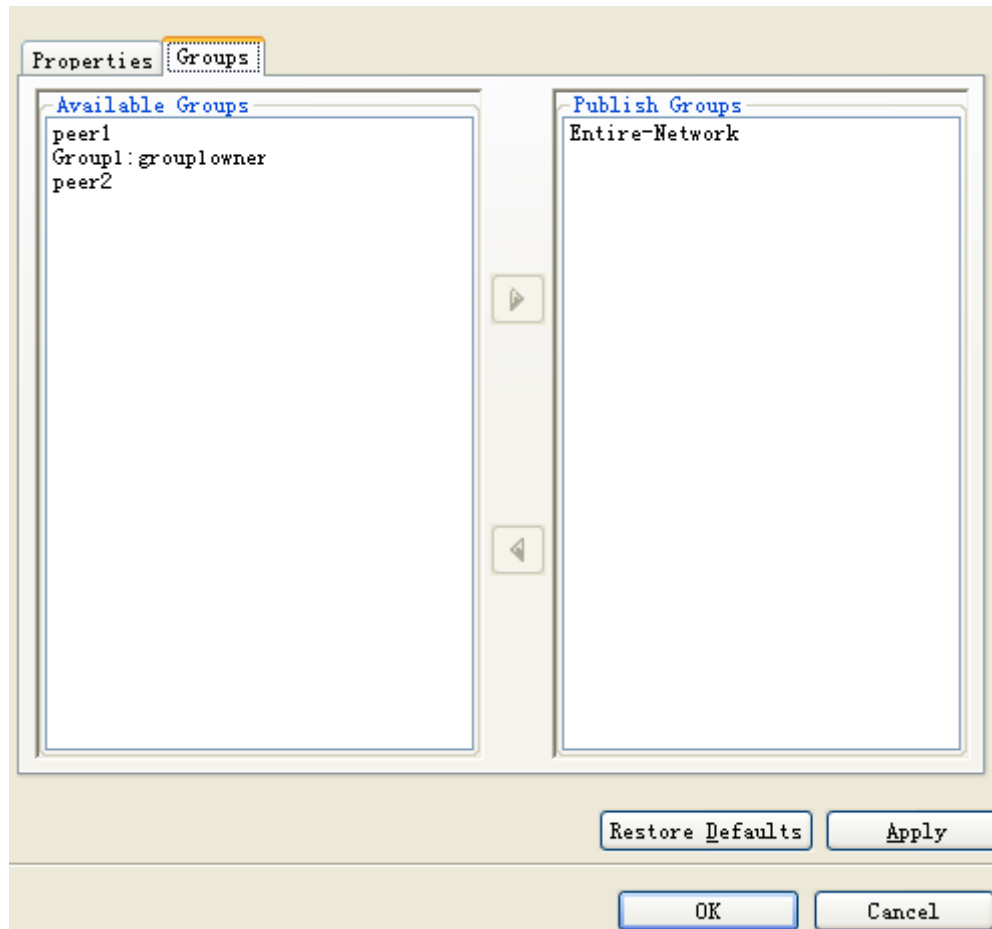


Figure 6.13: Adding Groups to P2P Rule

Changes done to the rule sets are only persisted once the “Apply” button has been pressed. The underlying P2P mechanism will then use these values to control the information that is published on the network.

7.2.2 Search Configuration

Crawl Repository

In order to create the index file needed for keyword-based search as well as initial knowledge base, different sources should be crawled. The user can control which sources should be taken into account. These can be managed by going to Crawl Repository preferences. Figure 7.14 shows the preference panel presented to the user.

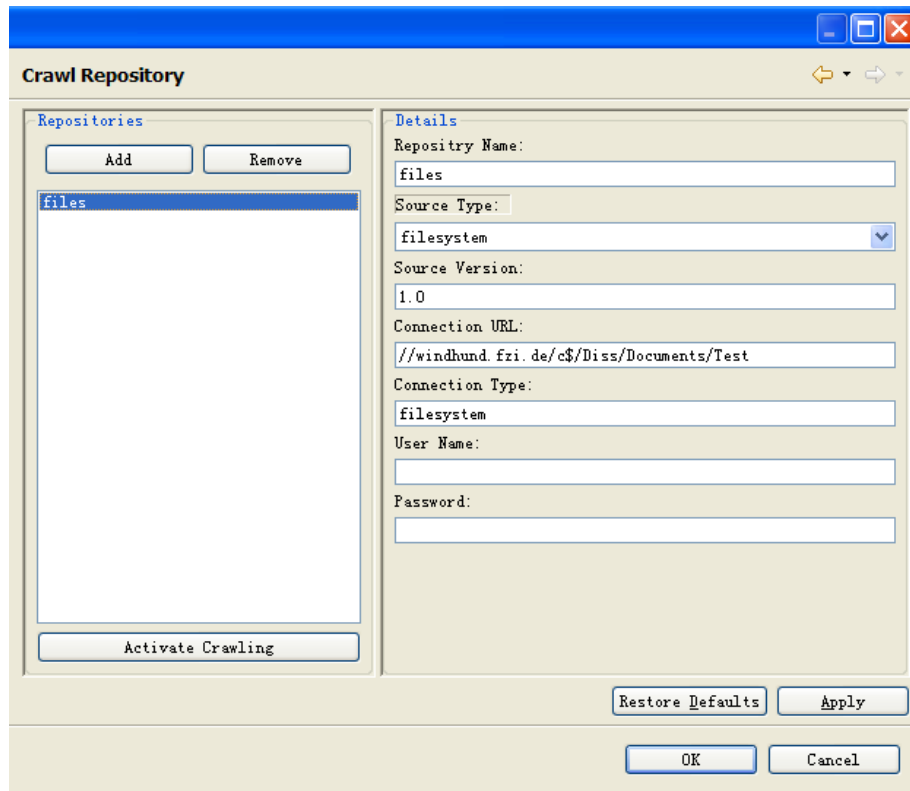


Figure 6.14: Service Crawlers Preferences

Crawling in the KnowBench system is configured by two XML based files. The `repo_config.xml` file is used for configuration of sources that could be crawled. One crawling session is specified in the `crawl_config.xml` file. Each Crawl can consist of several repositories defined in `repo_config.xml`.

The `repo_config.xml` file is used for defining repositories which should be crawled. It can contain multiple *repositoryInfo* elements. The parameters of a *repositoryInfo* element shown on the right-hand side of Figure 7.14 are:

repoName (Repository Name in Figure 7.14) - a symbolic name for the Repository. This field is mandatory

srcType (Source Type in Figure 7.14) – specifies a type of data source. This field is used for the decision which Crawler to use for this source. All available types are shown in Table 7.3. This field is mandatory.

srcVersion (Source Version in Figure 7.14) - placeholder for later versions. This field is optional.

connectURL (Connection URL in Figure 7.14) - the root url of repository to be crawled. This field is mandatory.

connectType (Connection Type in Figure 7.14) - placeholder for later versions. Actually all Crawlers decide on their own which connect type to use. This field is optional.

User (User Name in Figure 7.14) - user name. It is mandatory if authentication is required.

Pass (Password in Figure 7.14) - the password. It is mandatory if authentication is required.

The `crawl_config.xml` is used to define crawls which consist of one or several repositories specified in the `repo_config.xml`. Each Crawl is represented by a *crawlInfo* element which contains the repositories with the following sub elements:

crawlId - a unique Id for this crawl (*mandatory*)

crawlName - a symbolic name for this Crawl (*mandatory*)

updateFrequency - placeholder for later versions (*optional*)

repositoryIds – it contains one or more **repositoryId** specified in the `repo_config.xml` file.

srcType	Crawled Source
bugzilla	all bugs form a Bugzilla instance
cvschangelog	the change logs form a cvs instance
cvs	the HEAD revision of the files form a cvs instance
jira	all issues form a Jira instance
jspwiki	all wiki Pages form a JSPwiki instance
mediawiki	all wiki Pages form a Mediawiki instance
sourcesafe	all files of a SourceSafe repository
svn	the HEAD Revision of all files from an SVN Instance
filesystem	all files from a specified path
web	content of a given WebPage and all its subpages including files like pdf's etc.
imap	all mails of a given Imap/exchange mail account
outlook	all mails of a local Outlook instance

Table 6.3: Overview of the source types

To add a new source for crawling, the user should click on the Add button. This will add a new item with the repository name of “Untitled-” to the Repositories list (see Figure 7.16). The user should enter additional information and press the Apply button. As a result, both configuration files will be changed.

To remove an existing repository, the user should select it in the Repositories list and press the Remove button. The selected item will be removed from the list. In order to persist this change, the user should press the Apply button. As a result, both configuration files will be changed and the index file will be refreshed. This operation could be time-consuming.

To start crawling, a user should select one or more repositories in the list of available repositories and press the “Activate crawling” button. All selected repositories will be crawled and the index file will be created (or updated in the case that crawling was already done). Before starting the crawling, a consistency checking for the selected sources will be firstly performed. For example, the KnowBench system checks the existence of path of the filesystem, webpage URI or username and password of a JIRA instance, etc. An example of inconsistent input and the corresponding error message dialog is shown in Figure 7.15.

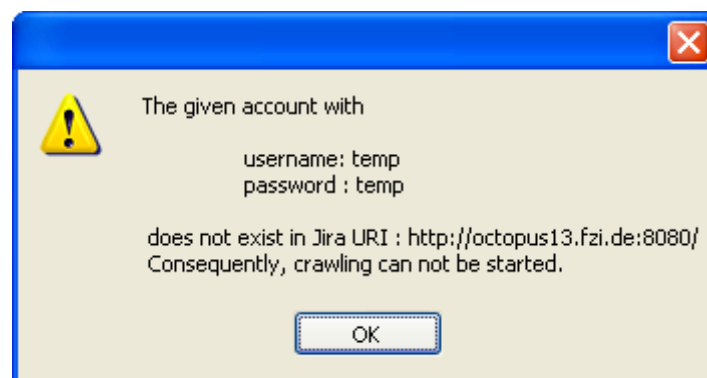


Figure 6.15: Error Message Dialog for consistency checking (Jira)

The crawling cannot be started, until all inputs are correct. In the case of a problem, an error message dialog will appear.

More information about crawling is given in section 7.3.

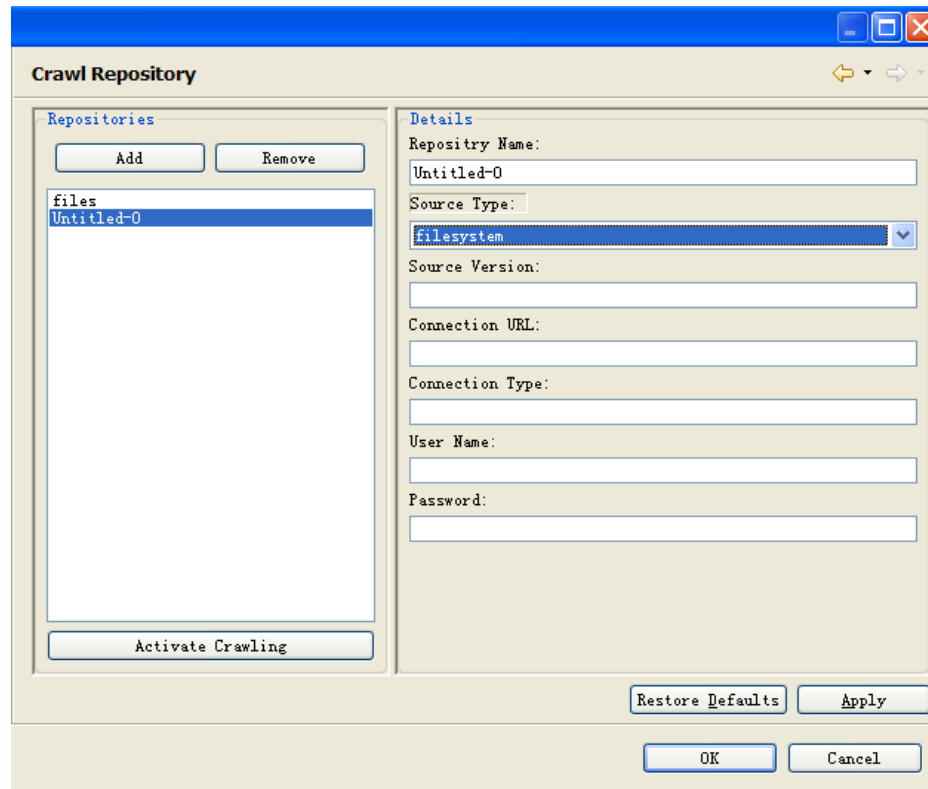


Figure 6.16: Adding a new Crawl Repository

7.3 Knowledge Acquisition

Knowledge acquisition in KnowBench is done by the search component through the crawling process. The user should click on the Activate Crawling button (see Figure 7.17). All sources identified as described in section “Crawl Repository” will be crawled. The results of this process are:

the integrated index file that will be used for the keyword search and
the initial knowledge base.

Crawling works in the background and will not block the other KnowBench functionalities. However, since it can be time-consuming (when a lot of sources or a big source is crawled), the user is always able to deactivate the crawling by clicking on the De-Activate button.

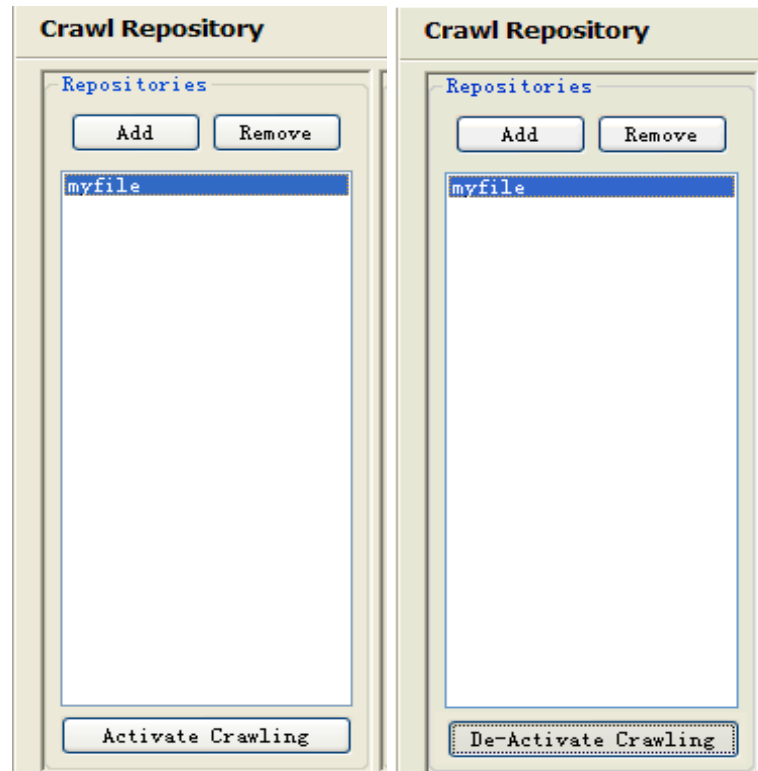


Figure 6.17: Starting and Stopping Crawler

Since crawling can be time-consuming activity, a progress of this activity will be shown for as long as the crawling is working. As shown in Figure 7.18, this information is shown in the Workbench progress area in the lower right corner of the Workbench. Crawling can be successfully finished or it can be interrupted. In both cases, a dialog will be shown to inform the user about the results of crawling. A dialog confirming successful completion of crawling is shown in Figure 7.19.



Figure 6.18: Progress during crawling

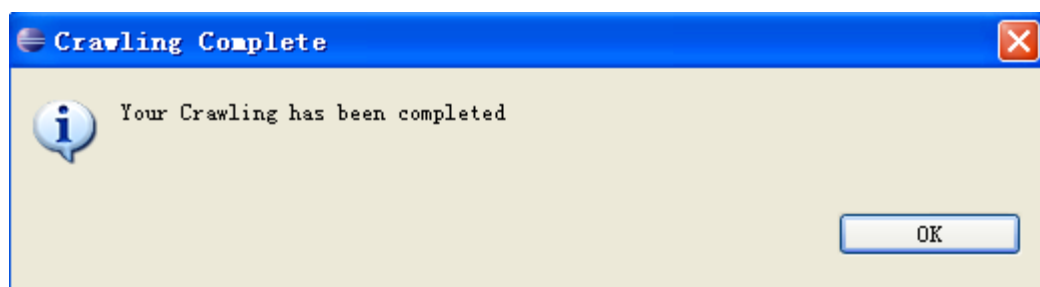


Figure 6.19: Crawling complete

The index is created in an incremental way. It means that adding new sources that should be crawled only extends the index file. The sources that are already crawled will be considered only if they are changed.

Regarding the knowledge base, the integrated data is represented semantically. This is done by (i) extracting metadata from structured content (e.g. code structure); and (ii) linking metadata - having generated metadata for various kinds of sources, it is necessary to relate them semantically (e.g. variable names, function names, log messages etc. that are likely to appear in issue/bug reports and linked to the code that is involved in producing the bug).

7.4 Knowledge Development

The KnowBench system provides a powerful user interface for developing knowledge. Several modes of putting knowledge into knowledge base are supported. They are described in the following sections.

7.4.1 Manual Knowledge Development

Knowledge Item Creation

The KnowBench system supports manual creation of knowledge items by dynamically creating dialogs whose content is generated based on an entity from the KnowBench ontologies.

To activate the manual development of knowledge, the user should select the knowledge editing perspective. This is shown in Figure 7.20. After selecting the option “Other...” in Figure 7.20, the Open Perspective dialog appears. The user has to select the “KB editing” and to press the OK button.

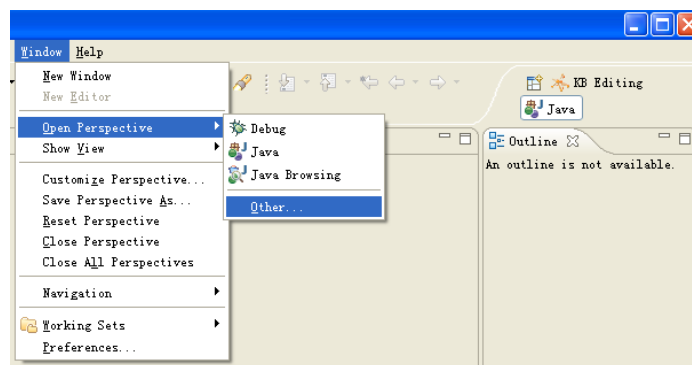


Figure 6.20: Opening knowledge editor

The knowledge editor of the KnowBench system will appear (see Figure 7.21). The left part of the knowledge editor shows the KnowBench ontologies that are used to structure the KnowBench knowledge base. The representation of the KnowBench ontologies in a form of a tree should help developers to correctly choose the most appropriate class. The user should navigate through the KnowBench ontology tree in order to find and select a class that will be instantiated, which means that the class (and its properties) will be used as a “template” for the creation of knowledge items.

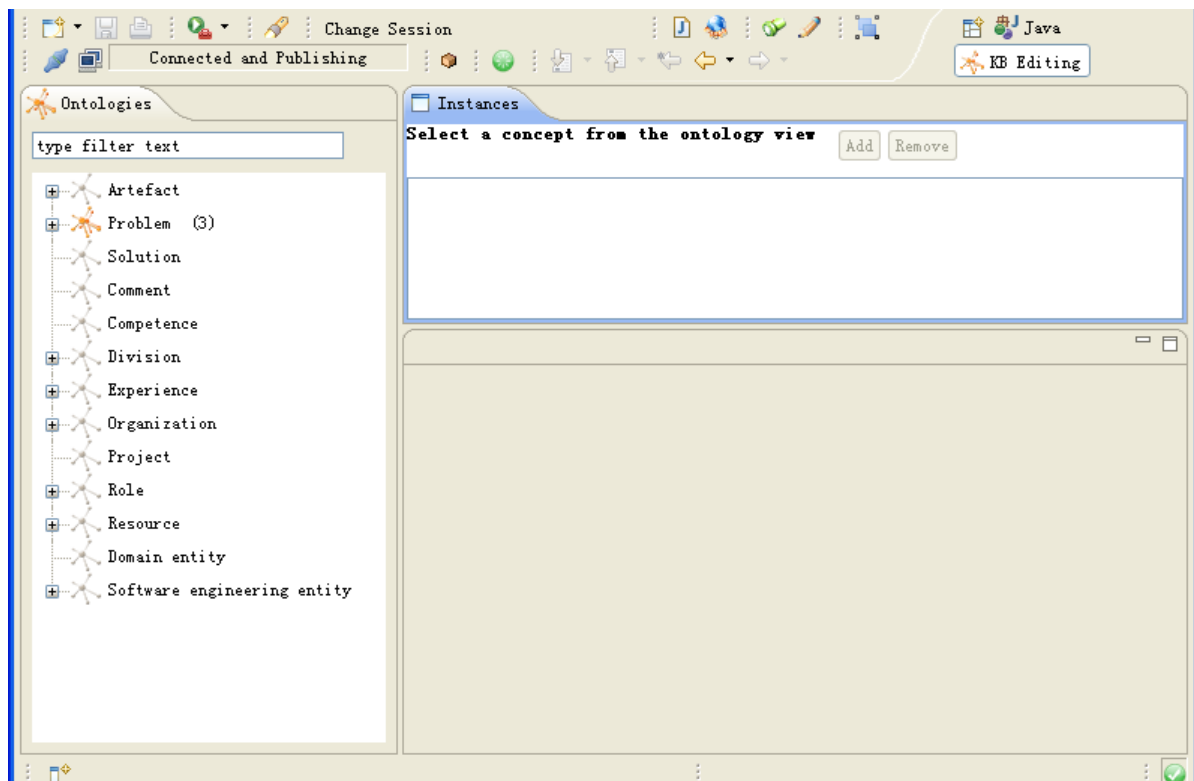


Figure 6.21: Knowledge editor

Three different icons are used for representing ontology entities in the tree. The meaning of these icons is explained in Table 7.4.

Icon	Meaning of the icon
	A class with individuals and subclasses
	A class with individuals and without subclasses
	A class without individuals

Table 6.4: Icons in the knowledge tree

The KnowBench system also provides support for searching for an ontology entity in the ontology tree. The user has to specify an input in the text field on the top of the ontology tree. The tree is automatically updated, i.e. only entities satisfying the user input will be shown in the tree. In order to speed up the process of finding the most appropriate class, filtering is supported.

Figure 7.22 shows the filtering of the ontology tree based on information provided by the user. Entities whose label or URI contains the user-provided input “is” are shown.

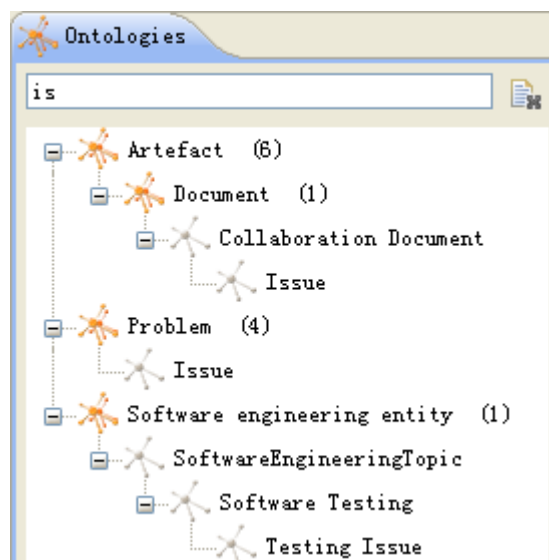


Figure 6.22: Searching in the ontology tree

The user can also clear the input by clicking the Clear button. This button is next to the text field for providing input in Figure 7.22.

When a class is selected, the UI shown on the right-hand side is updated. The upper part shows all individuals defined for the selected class. This is shown in Figure 7.23.

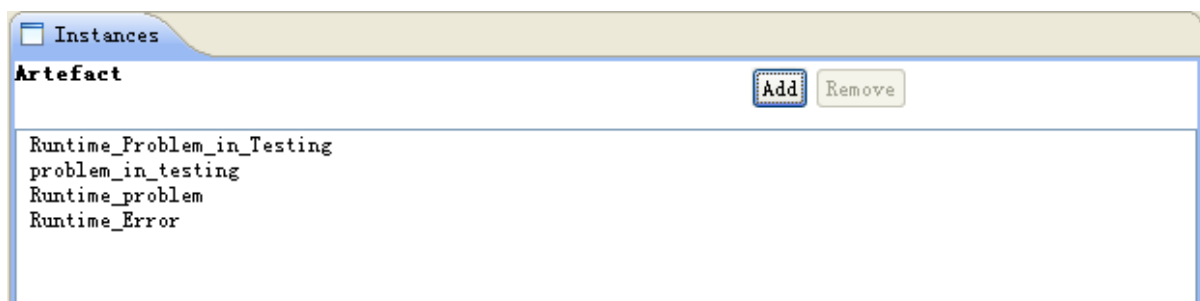


Figure 6.23: Instance part of the knowledge editor

When no individual is selected, only the Add button is enabled. In the case that an individual is selected, all buttons are enabled, which means that the removal of the individual is allowed. More information about removal is given in section 7.7.

To create a new knowledge item the user has (after selecting a class in the knowledge base tree as previously described) to press the Add button. The dynamic form on bottom will be opened as shown in Figure 7.24. The user should populate the values of the new instance. All fields in the dynamic form depend on the properties of the selected class. For the data properties, values can be added by using number widget for numeric properties, calendar for date-time properties and text field for all other properties. For object properties the values can be specified by moving items from one list to another (see Figure 7.25).

The screenshot shows a software interface for managing knowledge instances. At the top, there is a tab labeled 'Instances' and a section titled 'Artefact'. Below this, a list of related terms is shown: 'Runtime_Problem_in_Testing', 'problem_in_testing', 'Runtime_problem', and 'Runtime_Error'. To the right of this list are 'Add' and 'Remove' buttons. Below the 'Artefact' section is a dynamic form titled '*NewOntologyInstance'. This form contains several input fields and dropdown menus for creating a new instance. The fields are organized into two columns:

- Left Column:**
 - hasName:** Text input field.
 - hasOrigin:** Text input field.
 - Description:** Text input field.
 - hasValue:** Text input field.
 - Version number:** Text input field.
 - Usefulness:** Text input field with a value of '0' and up/down arrows.
 - Size:** Text input field with a value of '0' and up/down arrows.
 - Is about (Domain entity):** Text input field.
- Right Column:**
 - Create date:** Dropdown menu with '<choose date>' and a downward arrow.
 - blocker:** Text input field.
 - Title:** Text input field.
 - duration:** Text input field.
 - log:** Text input field.
 - hasTimestamp:** Dropdown menu with '<choose date>' and a downward arrow.

At the bottom left of the form, there is a 'Form' label. On the right side of the form, there are navigation icons: a back arrow, a list icon, and a forward arrow.

Figure 6.24: Manual creation of knowledge items

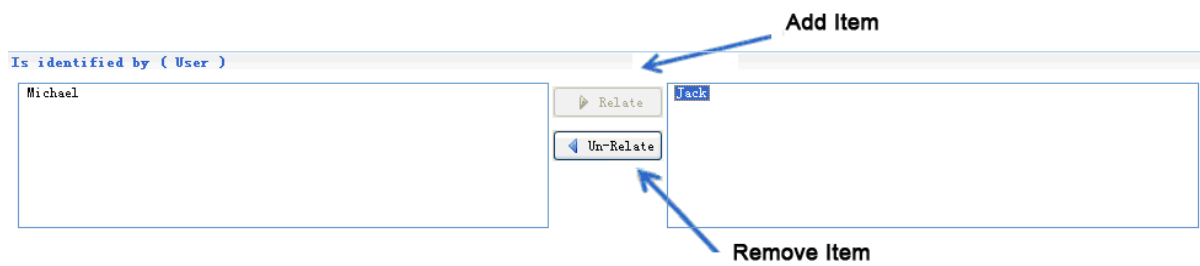


Figure 6.25: Defining values for object properties

After entering all necessary data, the user should press the Save button. The alternative way is closing of the dynamic form, when the system asks the user to confirm that the newly provided information should be stored. In both cases the dialog for identifying a new individual appears. This is shown in Figure 7.26. The user has to specify a name of a new individual which will be used later for the identification of the knowledge item and press the OK button.

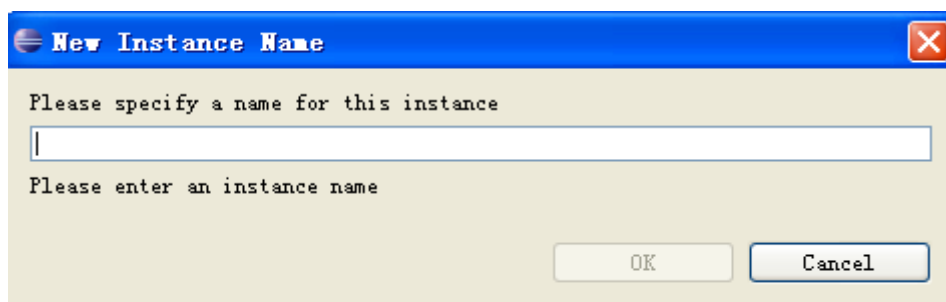


Figure 6.26: Saving a new individual

Once the values are persisted the header of the form is updated to the name of the individual and the newly added item is in the list of the existing individuals (see the upper part in Figure 7.24).

Annotation Creation

An important aspect of the KnowBench system is the ability to annotate semantically source code. Annotations can be created for any sections of the source file, as well as multiple annotations for the same section of a source file.

There are several possibilities for creating a new annotation:

the user can click on the Add Annotation button  in the toolbar

the user can use the Add Annotation option in the Annotations menu as shown in Figure 7.27; or

the user can use the context menu of the source code editor (see Figure 7.28).

In all cases a source file that will be annotated must be opened in the Eclipse development environment.

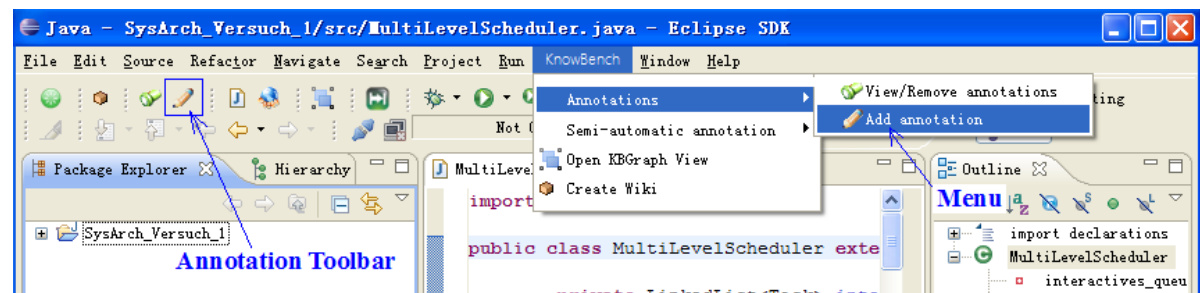


Figure 6.27: Annotation Toolbar & Menu

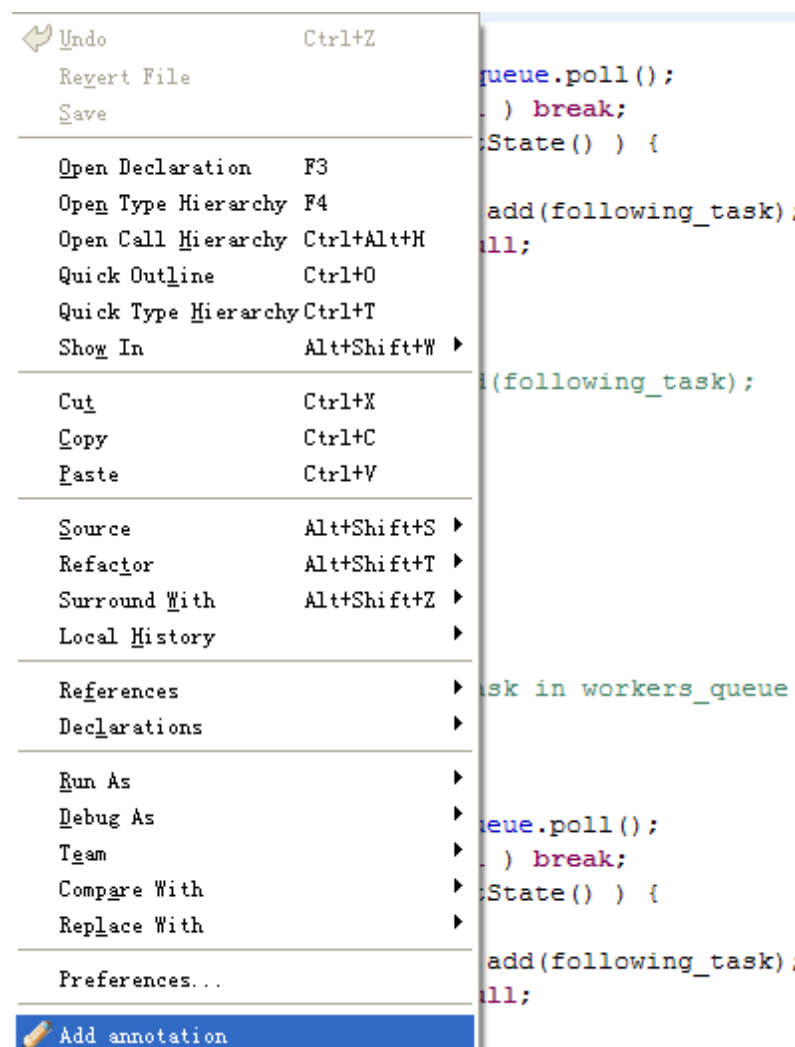


Figure 6.28: Context menu for creating annotation

The user is then prompted with a popup shown in Figure 7.29. The dialog consists of three parts:

upper part is used for selecting annotation tags in the Annotation ontology;

middle part is used for creating new annotation tags;

bottom part is used to show which code fragment (i.e. class, methods, etc.) will be annotated and which tag will be assigned to this code fragment.

To create an annotation, the user should select a tag by browsing the Annotation tags hierarchy (upper part in Figure 7.29) or define a new tag by providing a value in the Add Annotation tags part of Figure 7.29.

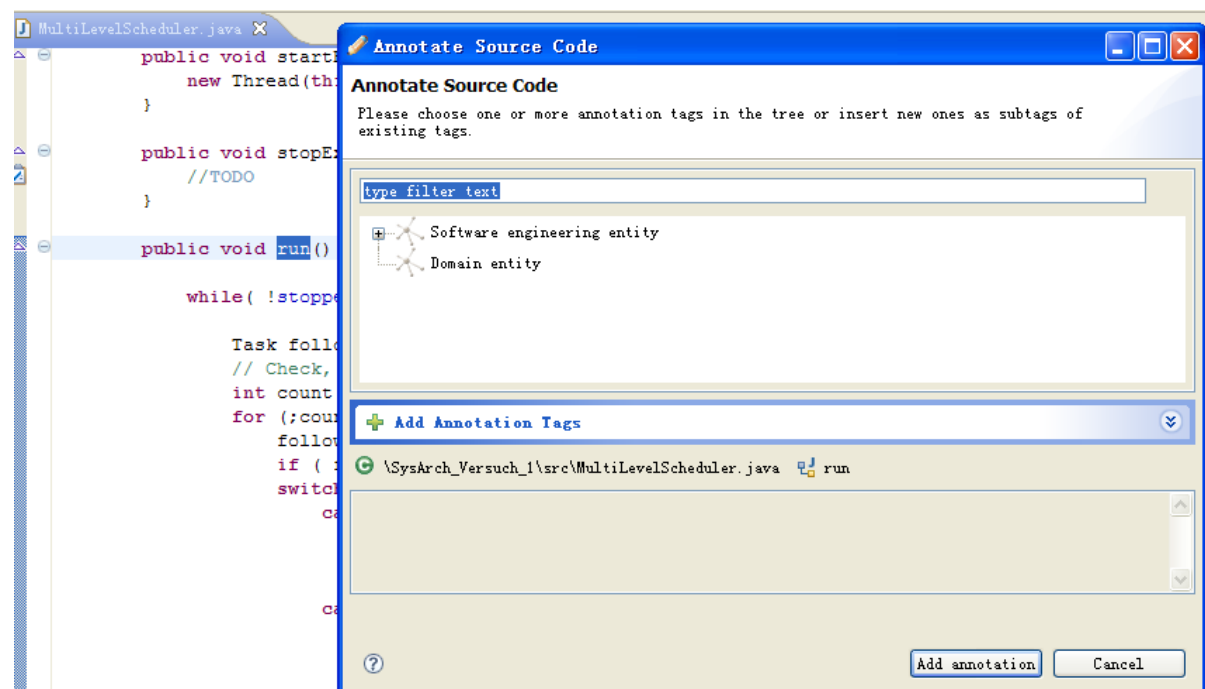


Figure 6.29: Annotation pop-up

In order to speed up the process of finding the most appropriate tag, the filtering of the annotation tags is supported. The KnowBench system is able to find all tags/classes satisfying the input provided by the user. This is shown in Figure 7.30.

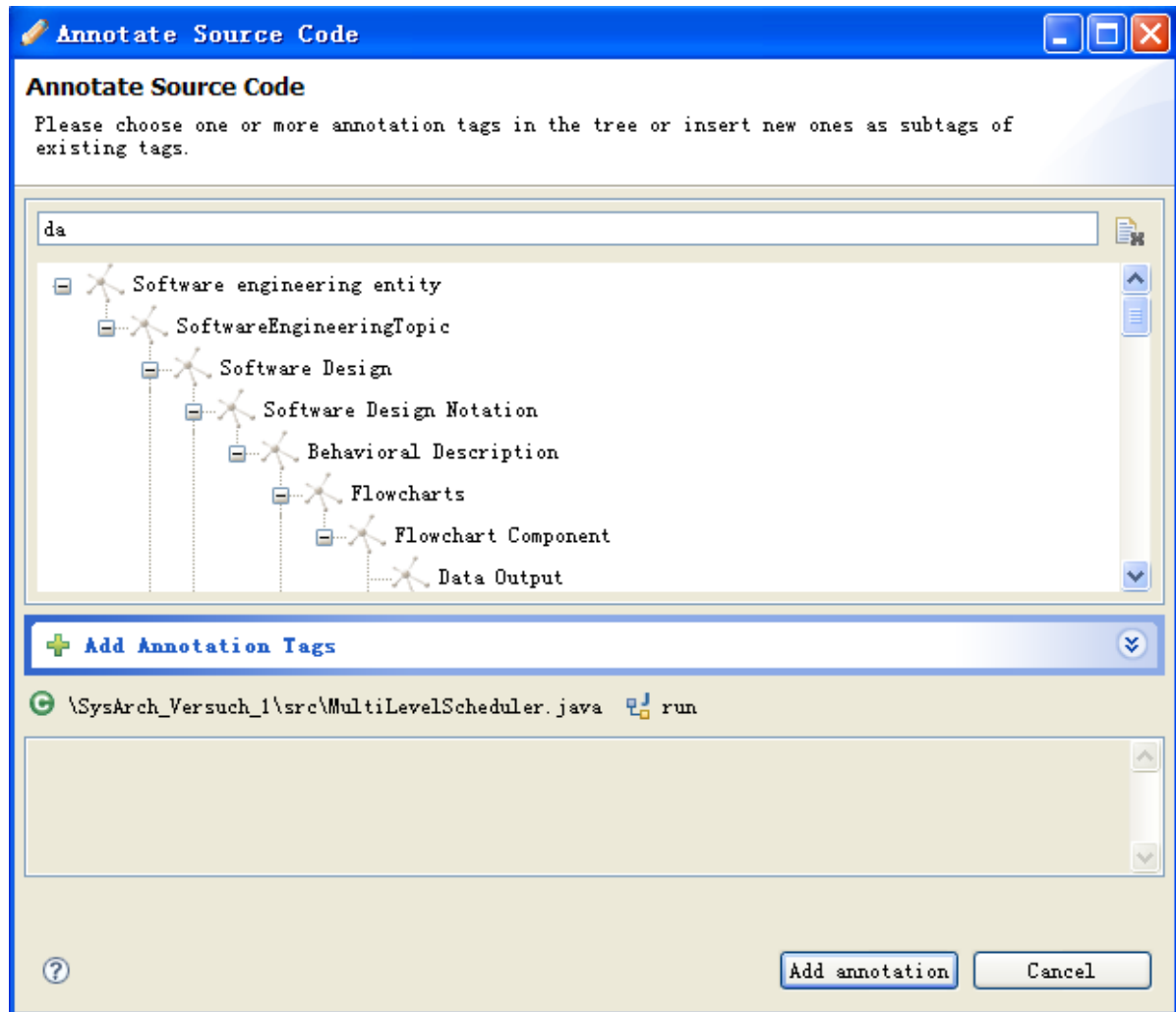


Figure 6.30: Searching an annotation tag/class

After choosing an annotation tag, the user should press Add annotation button (see bottom of Figure 7.30) in order to save the annotation. User can also cancel annotation by pressing Cancel button.

During creation of the semantic annotation of the source code, the users are able to extend the KnowBench Annotation ontology. To do that, the user has to click on the Add Annotation Tags button in Figure 7.30. The dialog will be updated as shown in Figure 7.31. The user has to select an entity in the Annotation tags tree, to provide value for a new tag and to click the button . In the case that no entity in the Annotation tags tree is selected, new tag will not be created.

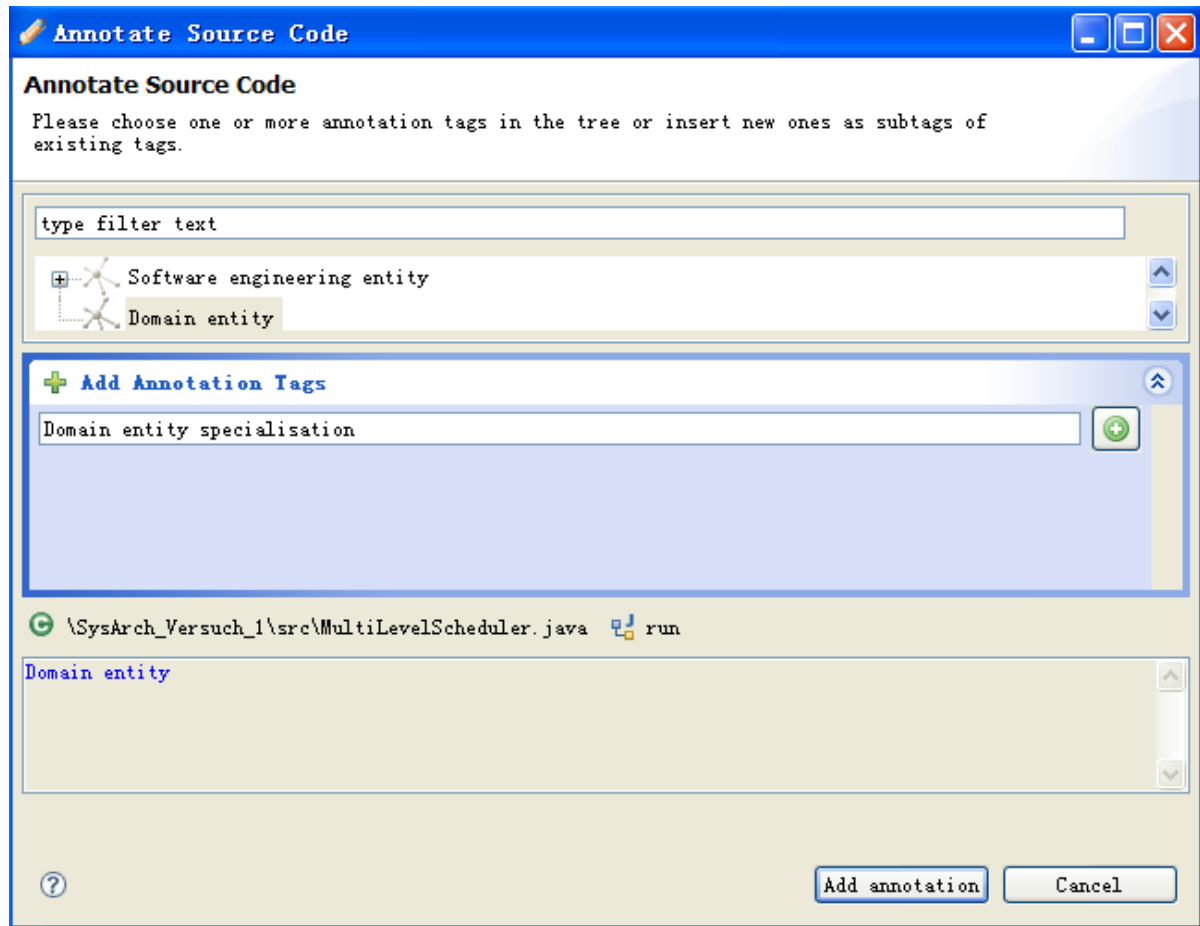


Figure 6.31: Creation of a new tag

The result is shown in Figure 7.32. The user is informed that a new class based on this tag will be created and will be added as a child of the selected entity in the Annotation tags tree. In the example shown in Figure 7.32, the new class domain entity specialisation will be added as a specialisation of the class Domain entity. The created tag can be removed by clicking the button .

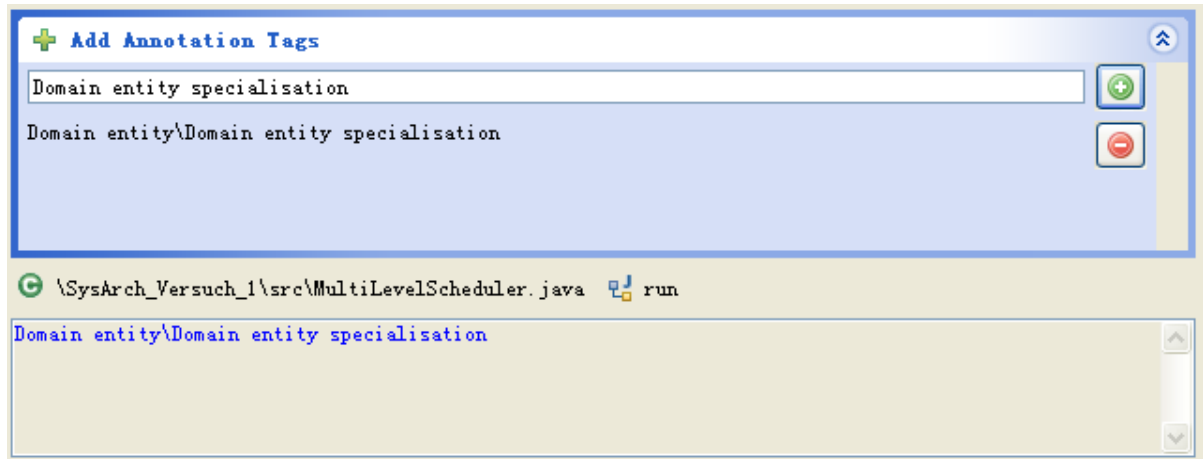


Figure 6.32: Result of tag creation

The bottom part of the annotation dialog (see Figure 7.31 or Figure 7.32) shows which source fragment will be annotated and which tags will be used for annotation. In the example shown in Figure 7.32, the run method in the class MiltiLevelScheduler.java will be annotated with the tag domain entity specialisation.

7.4.2 *Semi-automatic Knowledge Development*

Since manual development of knowledge is a very time-consuming activity, KnowBench automates this process by taking advantage of information extraction techniques to propose annotations. The user can create annotations in a semi-automatic way by using the semi-automatic annotation toolbar/menu (see Figure 7.33).

As shown in Figure 7.33, semi-automatic annotation can be created either for the currently opened source code file in the JDT editor or for a source code corpus contained in a directory. In the latter case the user should choose the directory containing the source code corpus as in Figure 7.34.

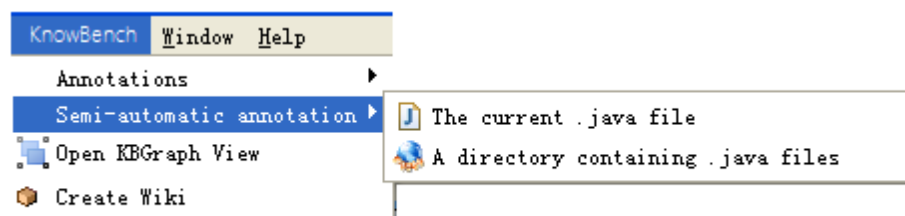


Figure 6.33: Different sources for semi-automatic annotation

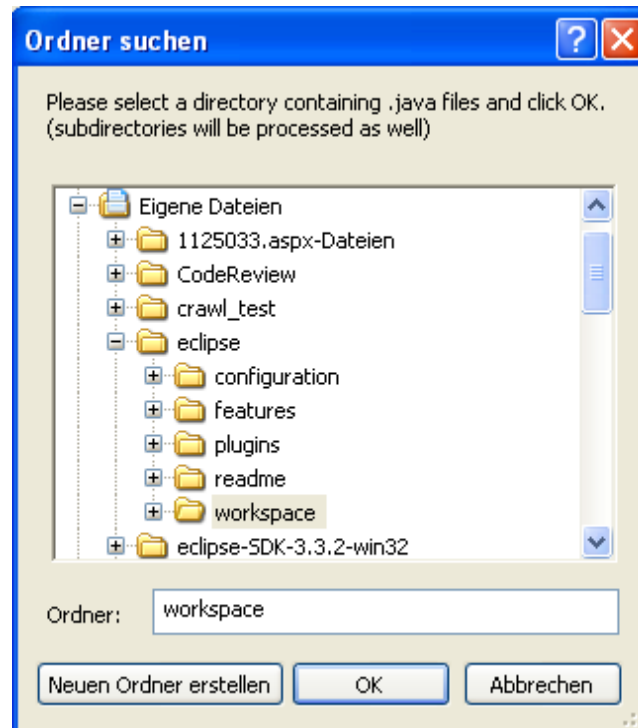


Figure 6.34: Choosing Source Code Corpus

After selecting the directory KnowBench analyzes the corpus and processes it as in Figure 7.35.

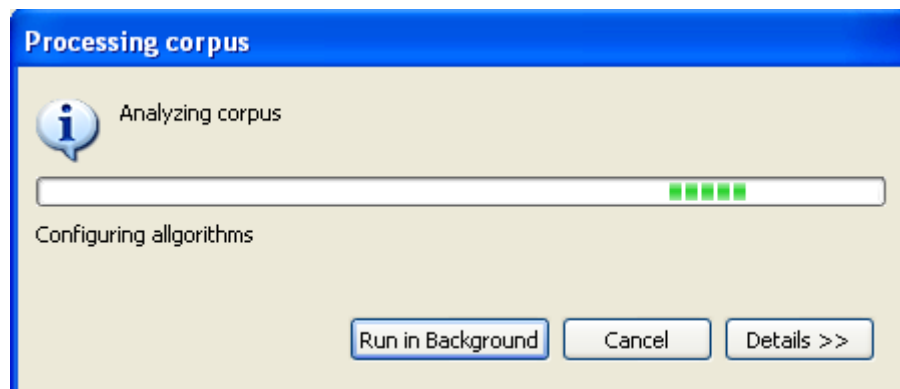


Figure 6.35: Processing Source Code Corpus

The result of the corpus processing is shown in Figure 7.36. Each proposal consists of the following parameters:

Annotation tag proposal – new proposal to be added in the Annotation ontology;

Super tag proposal – parent class for the proposed class;

Annotation type - annotation type can be either “Domain” or “Software Engineering”;

Confidence – it is the probability of a class to be precise.

The user is able to change the proposed annotations by changing their names and/or classification. The latter can be done by choosing their parent concept in the annotation ontology tree. Additionally, the user is able to filter the results dynamically by dragging the threshold slider shown on the top-right side on Figure 7.36. If the confidence of a proposed annotation is lower than the selected threshold then this annotation will not be included in the list, which is refreshed dynamically.

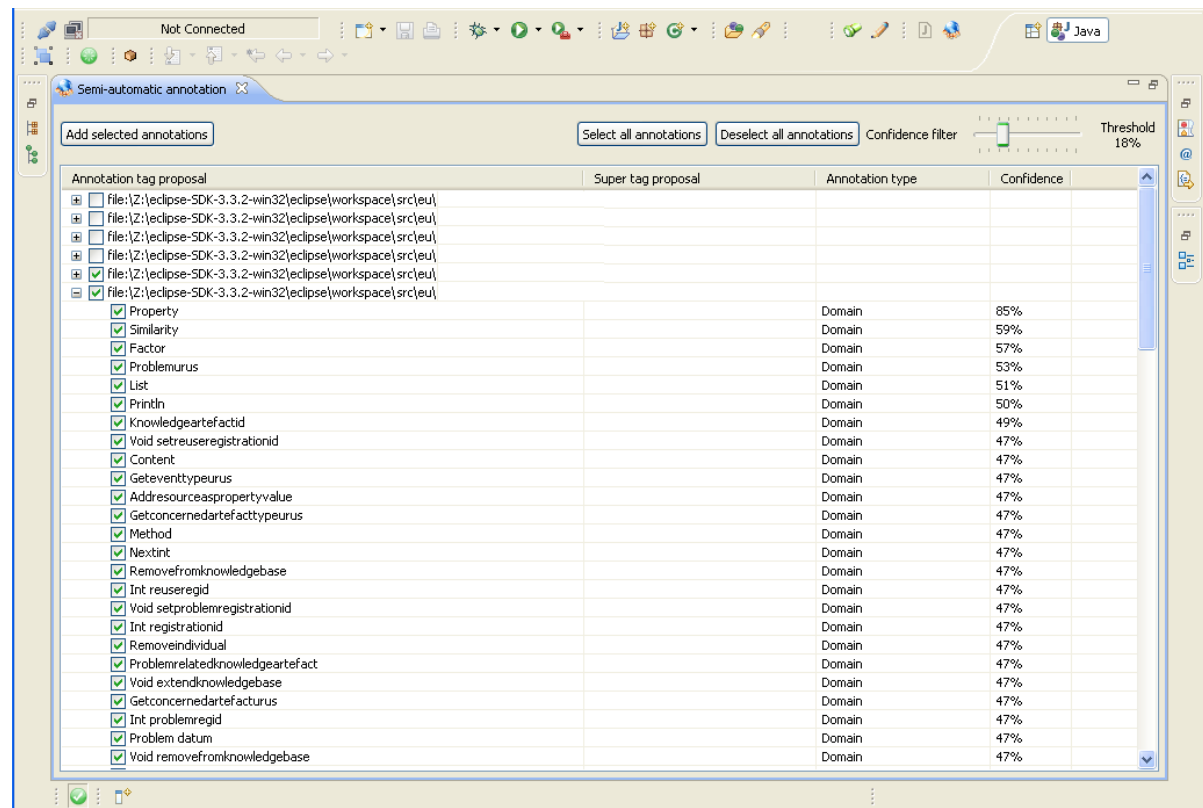


Figure 6.36: Semi-Automatic Annotation Editor

The user is able to select/deselect all proposed annotation by clicking on the Select/Deselect all annotations button. Additionally, the user can select or deselect each proposed annotation individually by clicking on the check box assigned to that proposal.

Finally, in order to store the proposed annotations the user should press the Add selected annotations button. The KnowBench system annotates the selected source code corpus as shown in Figure 7.37.

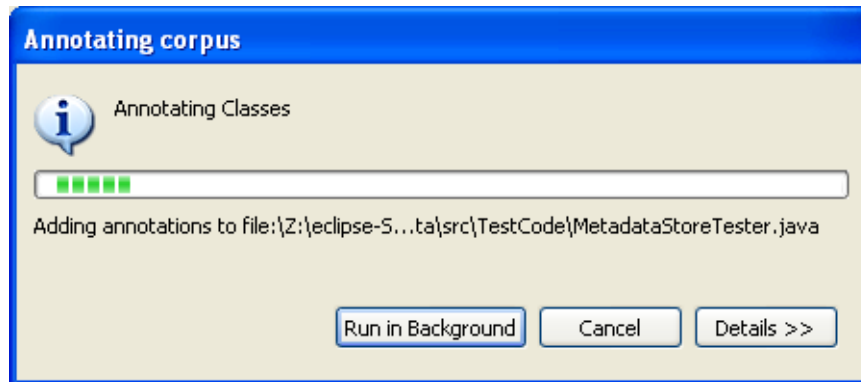


Figure 6.37: Adding Annotations to Source Code Corpus

Semi-automatic annotation requires a significant amount of memory. In the case that there is not enough memory the error dialog shown in Figure 7.38 will appear.

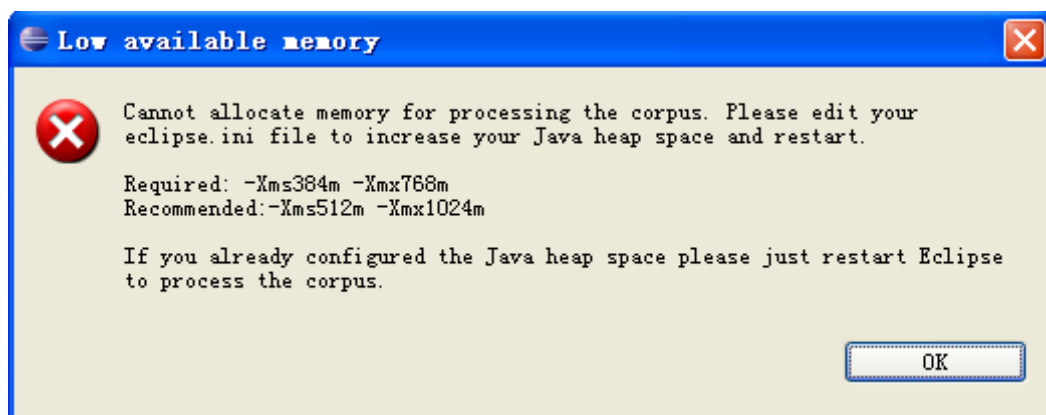


Figure 6.38: Memory requirement for semi-automatic annotation

This problem may be overcome (if physical memory is there) by either modifying the “eclipse.ini” file or launching eclipse with required parameters (e.g. xms and mxm). An example of the modified eclipse.ini file is shown below:

```
-showsplash org.eclipse.platform
--launcher.XXMaxPermSize 256M
-framework plugins\org.eclipse.osgi_3.4.2.R34x_v20080826-1230.jar
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Xms384m
-Xmx768m
```

7.4.3 Wiki-based Knowledge Development

The KnowBench system enables creating knowledge items using a wiki like system. The syntax of the semantic wiki is simple, and it is defined by the following rules:

The first section of the document has to determine the class of the instance. Subclasses are specified using the `->` operator.

The class section of the document *must* end with the class operator `->`.

The class section and the property section are separated by an empty new line.

The property values are specified using an ini style *label: value* pair.

In the case of an object property rule **(3)** still applies, and multiple values are **(,)** **comma** separated.

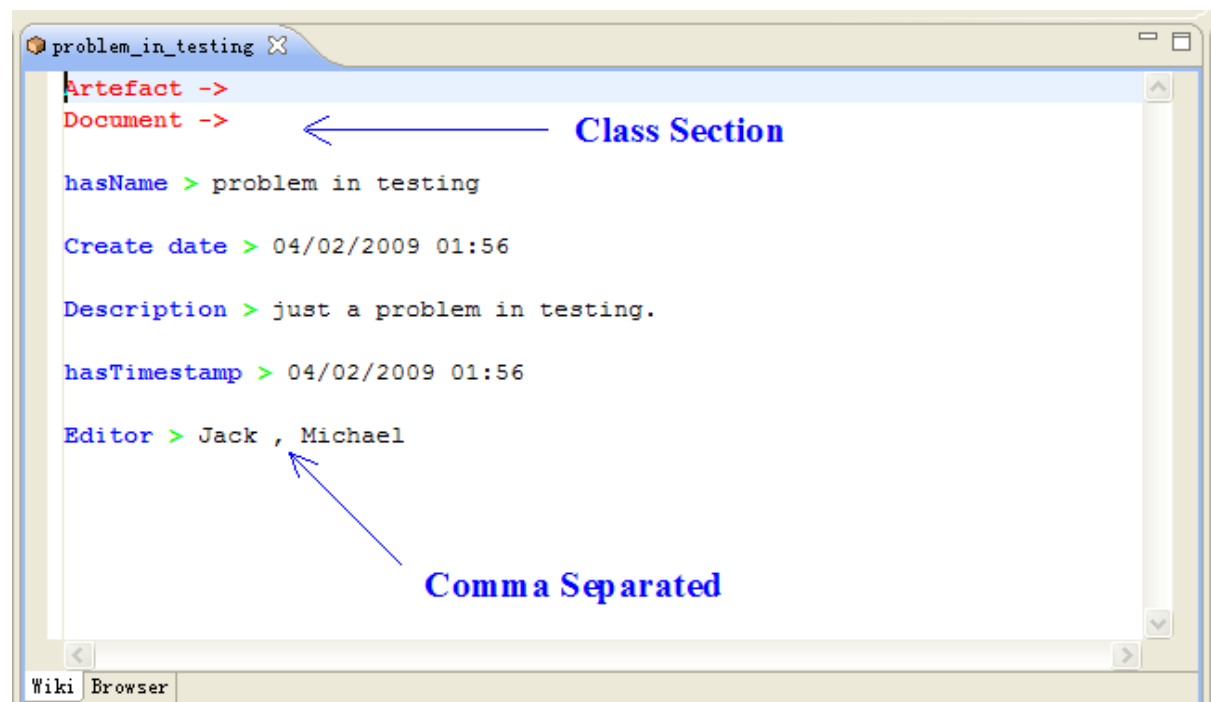


Figure 6.39: Wiki Syntax Explanation

A wiki can be created by clicking the “Create Wiki” button of the KnowBench toolbar (Figure 7.40).



Figure 6.40: Creating a new Wiki Instance

As shown in Figure 7.41, a new document is created with the name “**NewOntologyInstance**”. The newly created instance is not stored in the knowledge base automatically. The current editor is not persisted, which means that if KnowBench is closed,

then anything typed will not be saved. The user is of course notified to save the document before closing KnowBench.

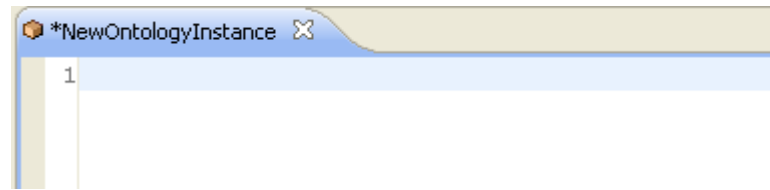


Figure 6.41: Initial Wiki Instance

To create an instance into the knowledge base, the user should use the auto complete mechanism. The invocation of the auto complete proposals occurs with the default key stroke combination of the eclipse environment, this is usually **CTRL-SPACE**. When the user is on a blank document and auto complete has been invoked, he/she will be presented with a list of all the root concepts of the KnowBench Ontologies (Figure 7.42).

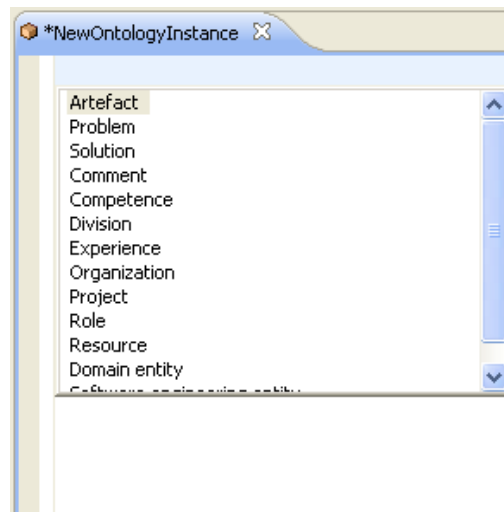


Figure 6.42: Wiki Auto-complete

In case the user has already typed a concept, and the cursor is after the class separator (->), then the auto complete proposals will contain the sub-concepts of that concept (Figure 7.43).

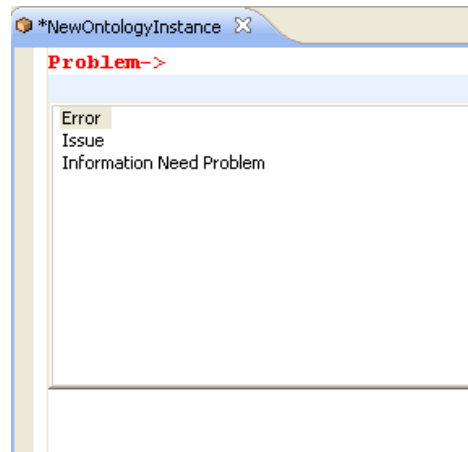


Figure 6.43: Wiki Auto-complete sub-concept

After the user has specified a correct concept for the current wiki instance and auto complete is invoked manually then the user will be presented with all the possible properties that can be specified for that concept (Figure 7.44). The proposal values suggested to the user are filtered in all cases, depending on previous existing values. For example if the property “*ID*” has already been typed, and the proposal list is invoked, then “*ID*” will not be in the values list, this makes it simpler and clearer for the user to make a selection.

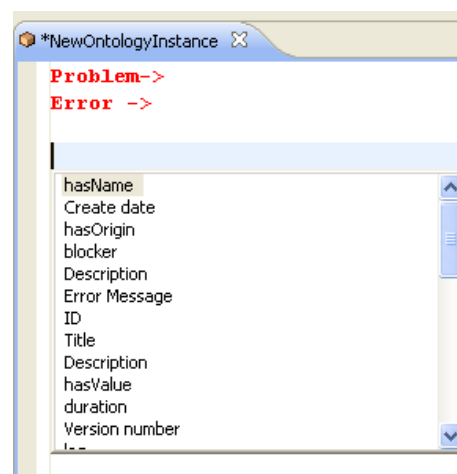


Figure 6.44: Wiki Property Auto-complete

The third and last possible auto completion proposal list is that of object properties where its values are a list of instances that already exist in the knowledge base. The wiki editor will retrieve all the instances that belong to the range class of the object property (Figure 7.45).

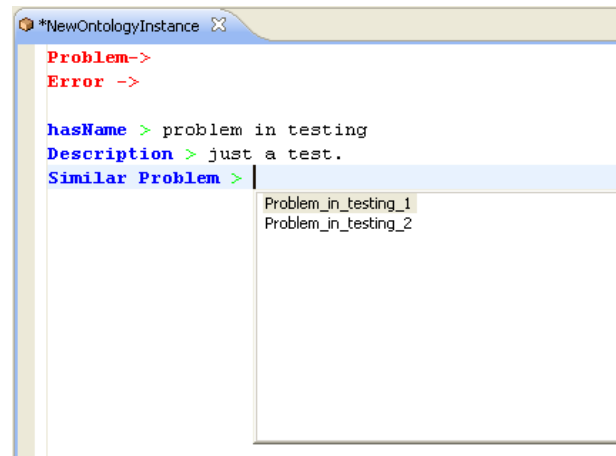


Figure 6.45: Wiki Object Property Values Auto-complete

For the data properties, the user should provide the values manually.

Syntax errors will cause the auto complete proposal retrieval to fail in all cases. Currently there is no visual representation for the user to understand invalid values or syntax errors while creating or editing the file, except for syntax colouring.

Finally, to save the wiki page, the default saving mechanism of the Eclipse environment (i.e. (the save shortcut or the File -> Save menu) should be used. As shown in Figure 7.46, the user will be asked for an “Instance Name” which will be used as identifier of the instance. Once the user provided the desired name, he/she should click the OK button.

The new individual will be added in the knowledge base. As shown in Figure 7.47, the UI will be refreshed to represent that it is an existing instance. Any errors in the input whilst adding or updating any of the information in the knowledge base using the semantic wiki will be displayed to the user.

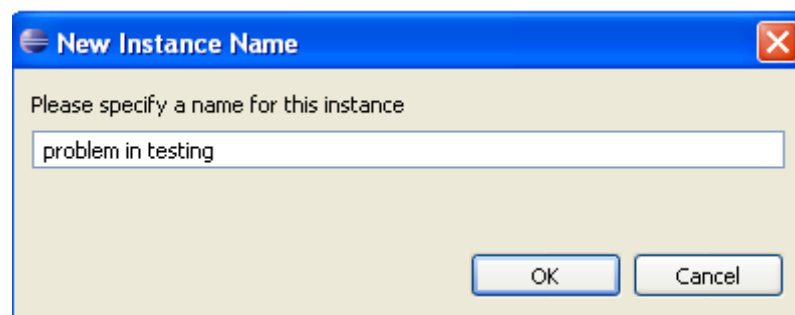


Figure 6.46: Wiki Instance Name Dialog

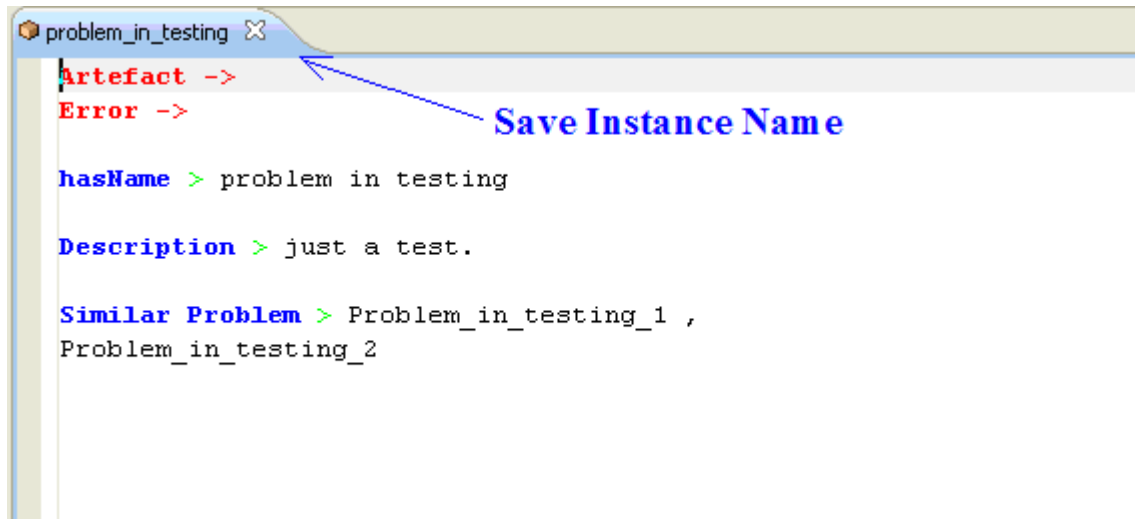


Figure 6.47: Wiki Editor Title Refreshed

7.5 Knowledge Distribution

The KnowBench system fosters knowledge sharing. It will occur within communities of practice and interests (represented as P2P network), which can help speed up the learning curve. From a technological point of view peer-to-peer solutions are particularly well-suited for knowledge sharing, because they make it possible for different participants (organizations, developers, or departments) to maintain their own knowledge structure while exchanging information.

7.5.1 Joining the P2P Network

In order to share the knowledge, the user should join one or many P2P networks. This can be done by using the Join the KnowBench network button  in the P2P join network toolbar (see Figure 7.48). In the case that the P2P network has already been established, the visual appearance of the connection button is changed in .

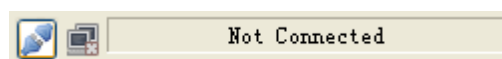


Figure 6.48: P2P Join Network Toolbar

Since the establishing of a peer-to-peer network is not a trivial activity, the KnowBench system guides the user by informing him/her what and where should be provided. This is shown in Figure 7.49.

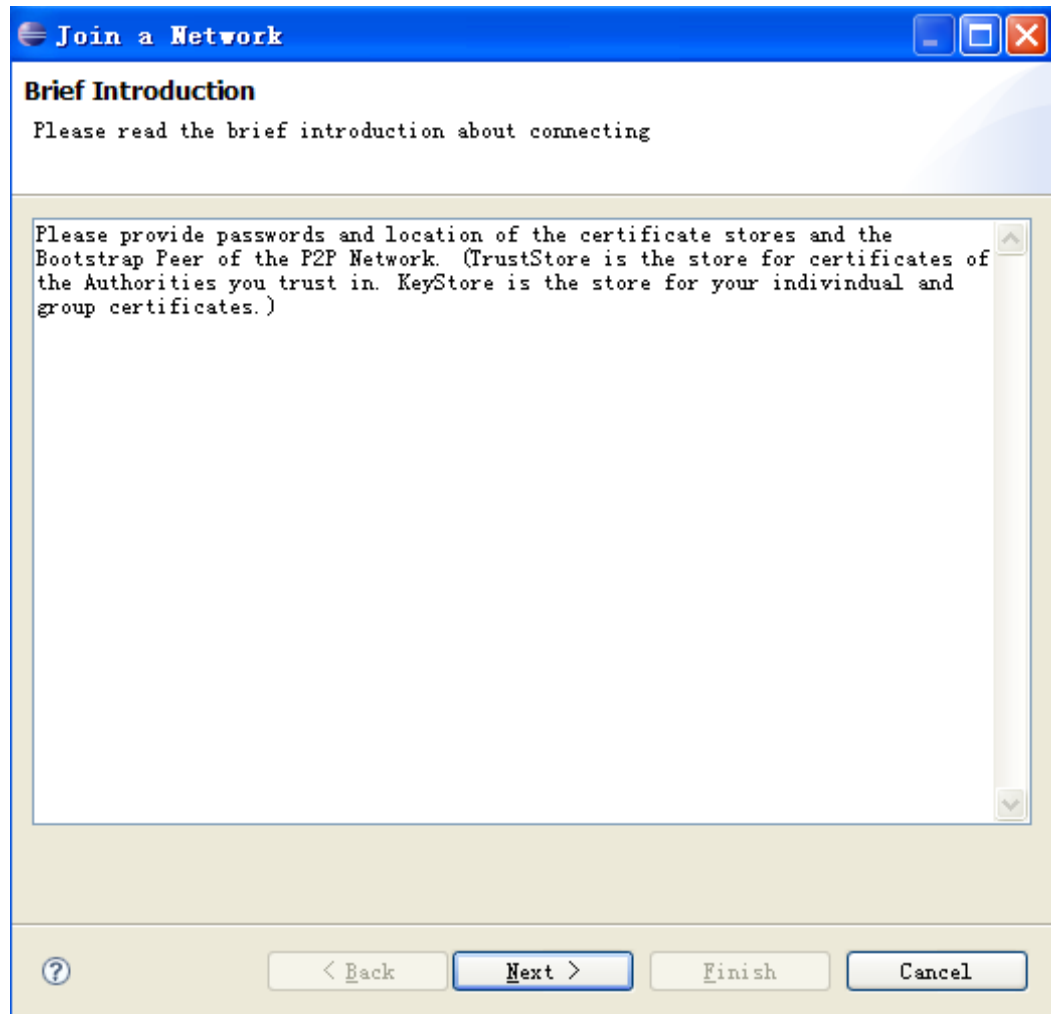


Figure 6.49: Introduction about connecting to the P2P network

After clicking the Next button in Figure 7.49, the dialog shown in Figure 7.50 will appear. The user should provide all necessary information. The following information is needed for a user to join a specific network:

KeyStore password;

TrustStore password;

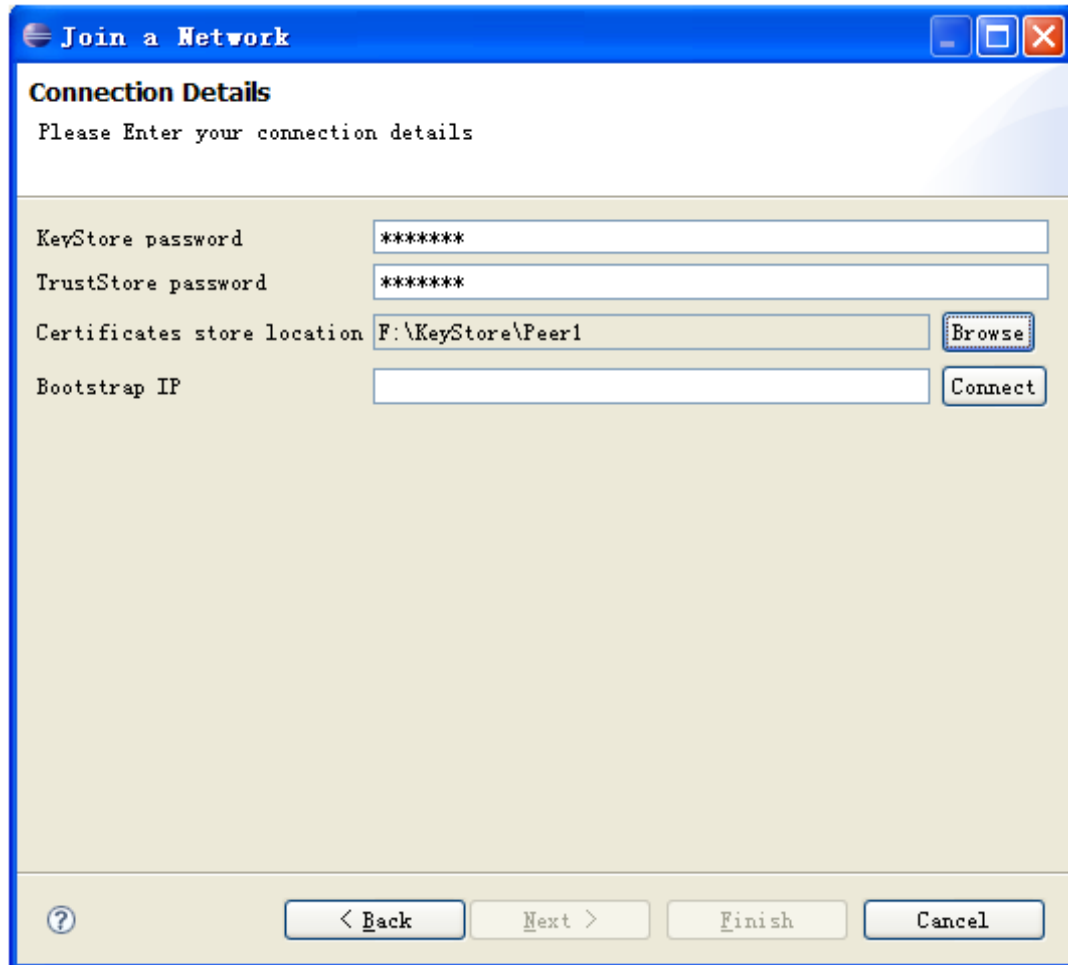
Certificate store location;

Bootstrap IP.

The user has to pass two stores; a KeyStore and TrustStore to the P2P layer. All the subjects' credentials are stored in a KeyStore. All the Certificate Authority's credentials are stored in TrustStore. These stores must have different names. Both files should be kept in a

single directory and its location should be specified in the Certificates store location in Figure 7.50.

The first node that starts the network is called "bootstrap node". For this node the user does not need to pass any text in the corresponding entry in Figure 7.50. For the second peer, the user should mention the first one's address and port number in the bootstrap field in Figure 7.50 . The ports that could be used are 1805 and 10000.



The image shows a Windows-style dialog box titled "Join a Network". It has a blue title bar with standard minimize, maximize, and close buttons. The main area is titled "Connection Details" and contains the instruction "Please Enter your connection details". There are four input fields: "KeyStore password" and "TrustStore password" both containing "*****"; "Certificates store location" containing "F:\KeyStore\Peer1" with a "Browse" button to its right; and "Bootstrap IP" which is empty with a "Connect" button to its right. At the bottom, there is a row of buttons: a help button (question mark icon), "< Back", "Next >", "Finish", and "Cancel".

Figure 6.50: P2P Connection Dialog

Once the user has entered the necessary information he/she should press the Connect button. The KnowBench system will attempt to connect the user to the network. Once connected, the user will be asked if he/she wants to publish unpublished offline data (Figure 7.51). Note that the Finish button of the wizard is disabled until the user answers to the question.

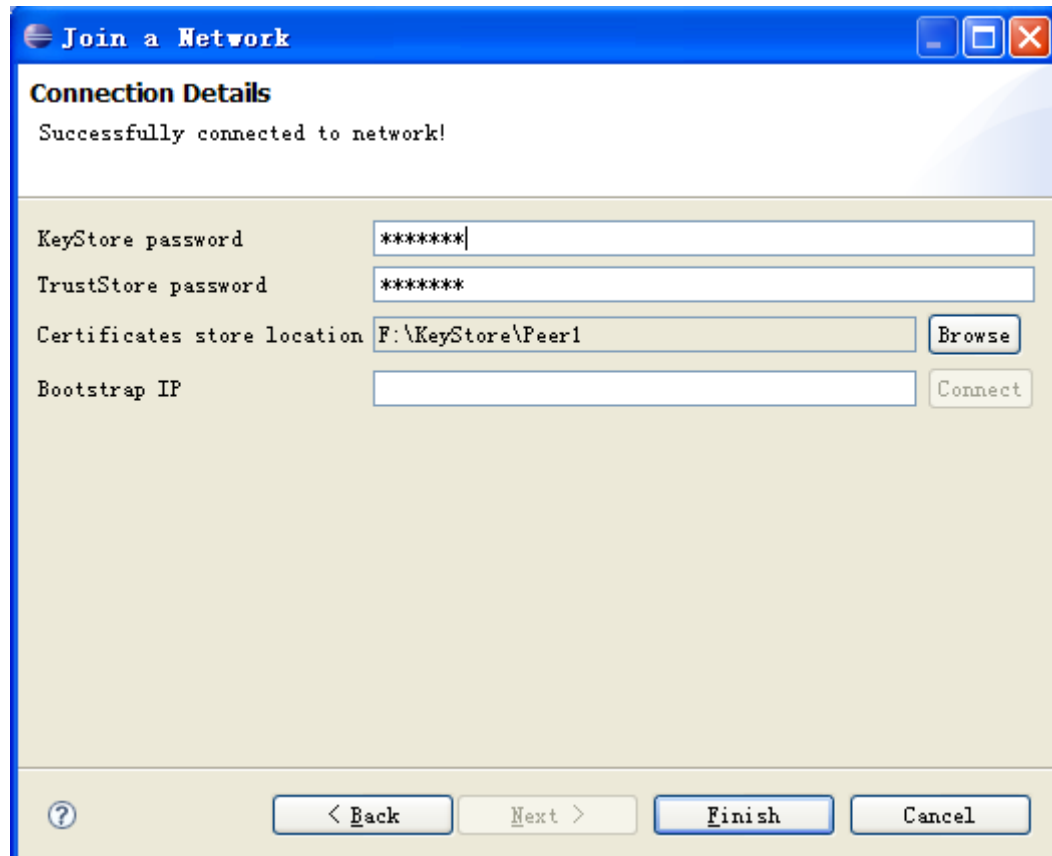


Figure 6.51: P2P Connected Dialog

7.5.2 Sharing Knowledge

Once the user connects to the network (after clicking the Connect button in Figure 7.50) the P2P toolbar is changed as shown in Figure 7.52. When the user decides to publish knowledge, the publishing starts in the background based on the policy. The creation of the policy is described in section 7.2.1. By specifying policy, the KnowBench system is able to detect new instances in the local store which meet this criterion and publishes all such instances. This is useful when a peer with lot of local data joins the network and wants to share part of the local store into the system. When the publishing is started, the P2P toolbar is changed as shown in Figure 7.53.

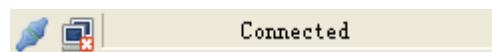


Figure 6.52: P2P Join Network Toolbar when connection is established

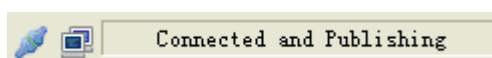


Figure 6.53: P2P Join Network Toolbar when knowledge is shared

The KnowBench knowledge is published at the instance of KnowBench ontology classes' level. When an instance is published, all the properties associated with the instance are visible in the network. When an instance is published, its object property values are also published, but not the target itself. Publishing is limited to only one level. In order to make the target also available, they must be published separately. For example, a Problem instance has a resource property that has its value an instance of a Solution class. By default, only this solution URI will be published and not the solution instance itself.

To publish data, the KnowBench system should be installed on at least 2 computers. It is also possible to run two instances of the same system on a single machine. In this case, the availability of two different working directories must be ensured.

7.6 Knowledge Usage

The KnowBench system supports knowledge usage by providing services for visualisation, searching and browsing.

7.6.1 Knowledge Base Visualization

The KnowBench system combines transparently and intuitively the metadata with the KnowBench ontologies. It enables the representation of different aspects of the underlying information and allows for easy and flexible presentations of the same information in different ways. Currently, two ways are supported: tree-based representation and graph-based representation.

Tree-based representation of knowledge base

The tree-based representation of the KnowBench knowledge base is shown in Figure 7.54. Since it is a part of the knowledge editor, it can be activated by opening the knowledge editing perspective.

The tree view gives direct visual information on the hierarchical structure of the KnowBench ontologies. Indeed, the hierarchy is expressed in a collapsible, direct manipulation tree. The class labels are used for nodes in the tree. The numbers assigned to the classes indicate the number of individuals defined for them. Additional information such as information regarding individuals or referencing relations can be seen on the right-hand side of the knowledge editor for a class selected in the tree.

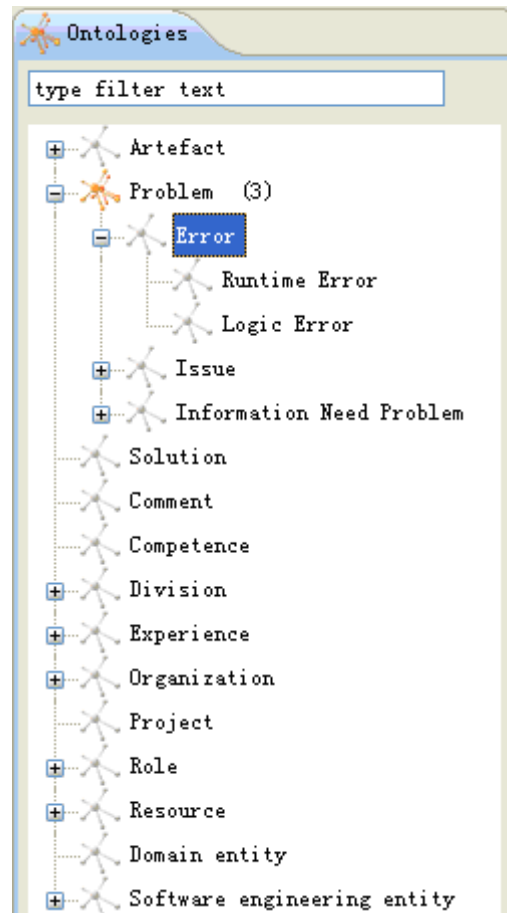


Figure 6.54: Tree-based representation of knowledge base

Graph-based representation of knowledge base

Since graphical representations are received and interpreted more efficiently, KnowBench system enables graphical representation of the KnowBench knowledge base. The graph-based representations can be invoked by clicking on the Open KB Graph button in the toolbar depicted in Figure 7.55.

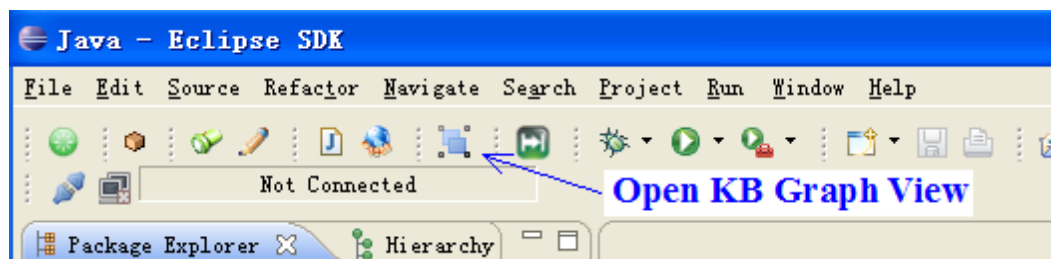


Figure 6.55: KB Graph Toolbar

As shown in Figure 7.56, the graphical representation is made up of nodes, which could be classes or instances, and edges that are made up of properties or relationships. Here the user is able to browse the knowledge, but he/she is not able to make any modification.

Visualization in the form of a graph can help the user analyze and comprehend this information better.

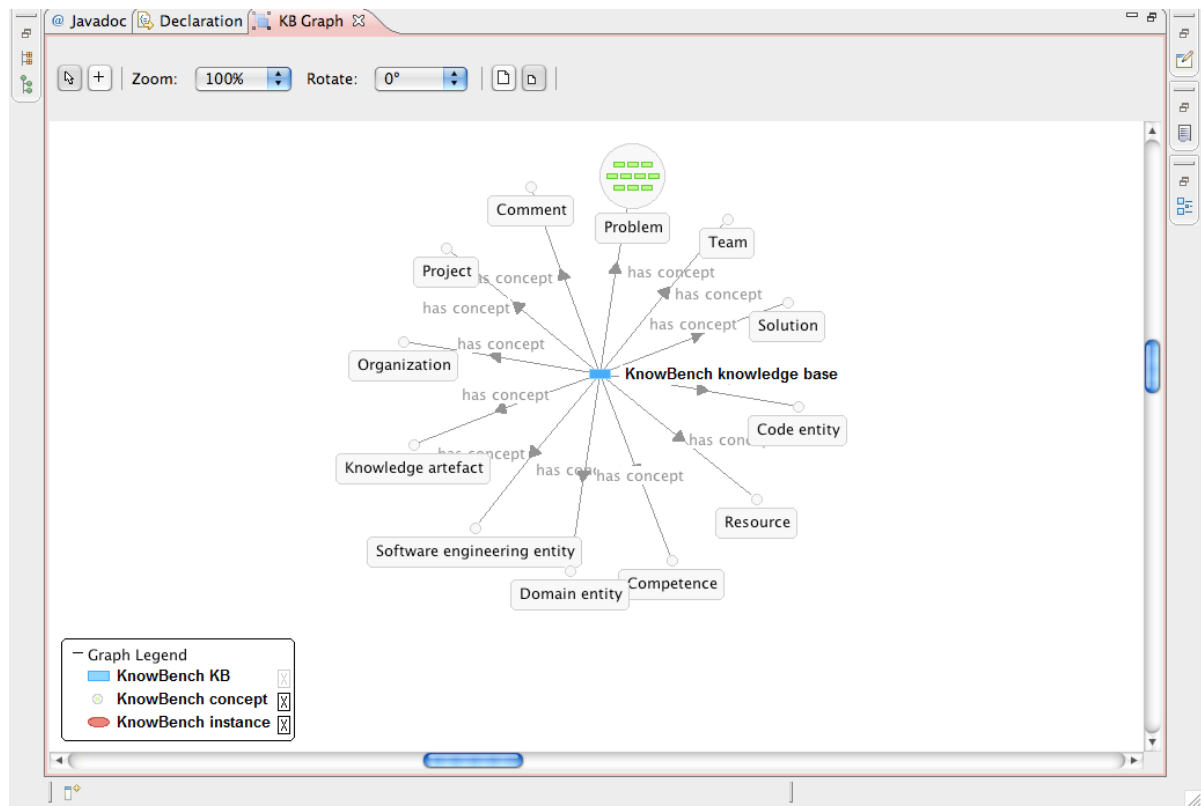


Figure 6.56: Graph-based representation of knowledge base

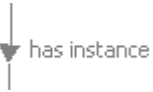
The following features are implemented:

Visualization of concepts, instances and their relationships using graph nodes (different kind of nodes for concepts or instances) and arrows respectively:

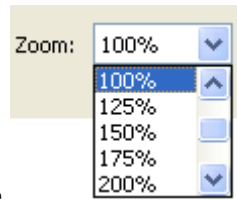
Node  represents an individual;

Node  represents a class without individuals;

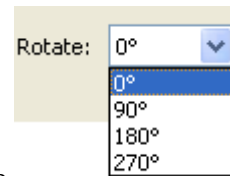
Node  represents a class with individuals;

Edge  represents a class instantiation relation;

Edge  represents an arbitrary relation between two individuals;



Zooming in/out the whole graph (see);



Rotation of the graph by 90, 180, 270 degrees (see);

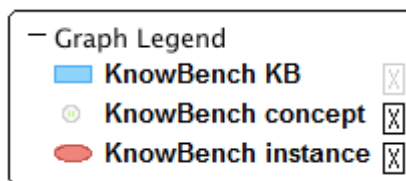


Large/small icons of the graph nodes (see);

Smooth floating animation of the graph while dragging around a node to focus on;

Auto-layout of the graph when it is moved by dragging a node;

Legend of the graph nodes (i.e. concepts and instances) (see



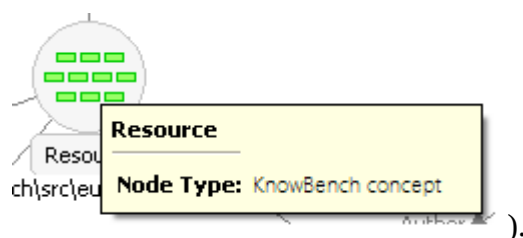
);

Filtering of all instance or concept nodes by clicking the corresponding check boxes in the legend;

Expansion/collapse of concept nodes (by double-click) will show/hide instances that have outgoing semantic links to other instances (object property instantiations);

Concept clusters indicate how much “loaded” is a concept in terms of its instances and in comparison with the overall KB load – this is indicated with a 0 to 10 scale;

Tooltips on every node indicating its name and type (concept or instance) (e.g.



).

In Figure 7.57 it is shown that each concept is represented as a node. In the case where a concept contains instances, a filled bubble is attached to it. In order to navigate the knowledge base, the user double-clicks on any of the concept nodes resulting in the node expansion revealing its instances or drag around the whole graph, as seen in Figure 7.58. All semantic links are shown between concepts and instances and their description is presented as a label in the appropriate arrow of the graph.

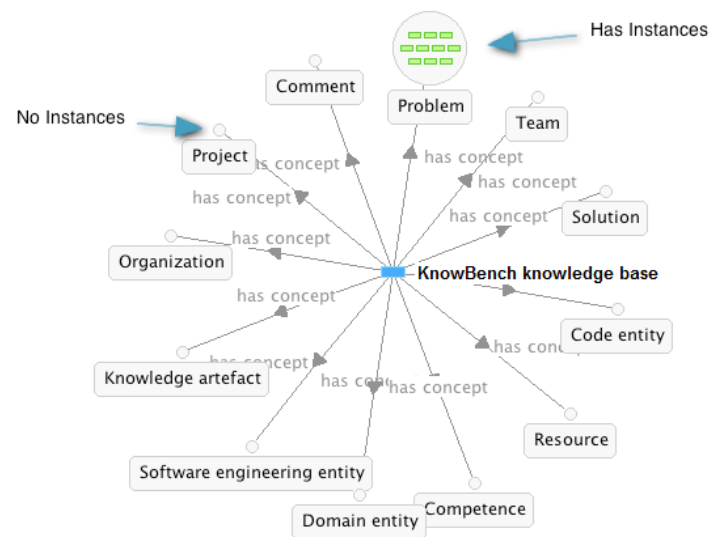


Figure 6.57: Graph Nodes

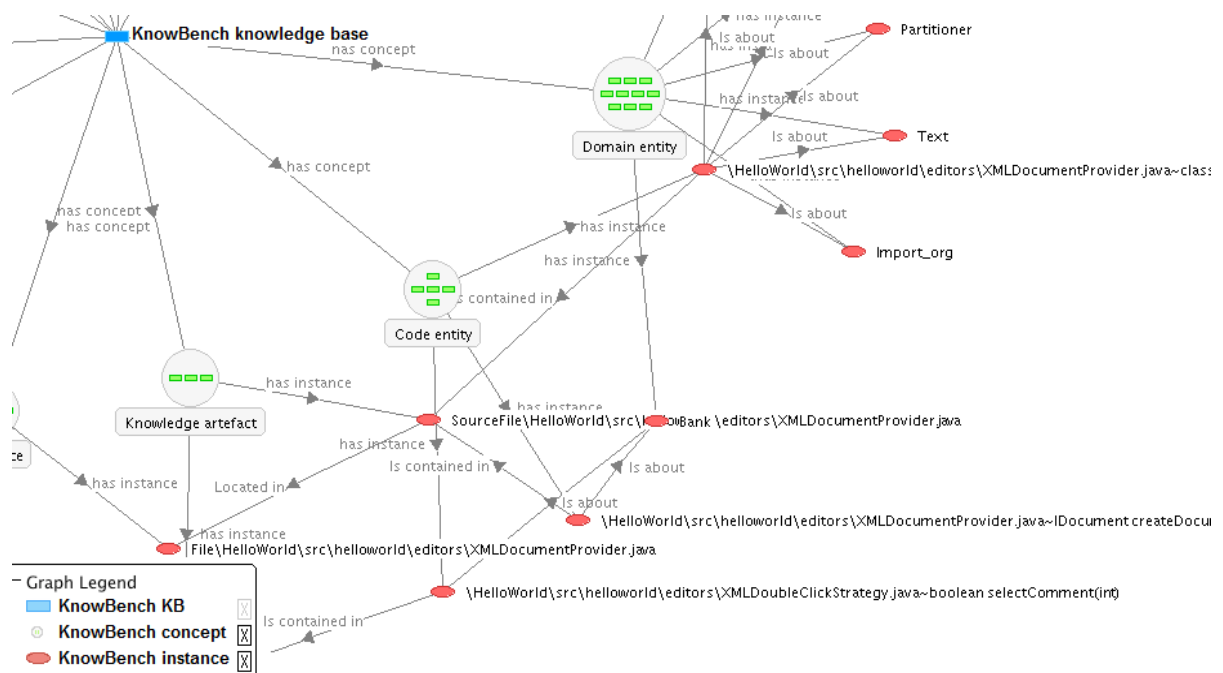


Figure 6.58: Expanded Concept Nodes

7.6.2 Search

The KnowBench system supports actively searching for knowledge both within the organization and outside. As shown in Figure 7.59, the searching functionality of the KnowBench system is integrated into the default “Search” dialog of the Eclipse environment. The “Search” dialog can be invoked using the Eclipse Search Menu, or on a default installation, by using the shortcut CTRL-H.

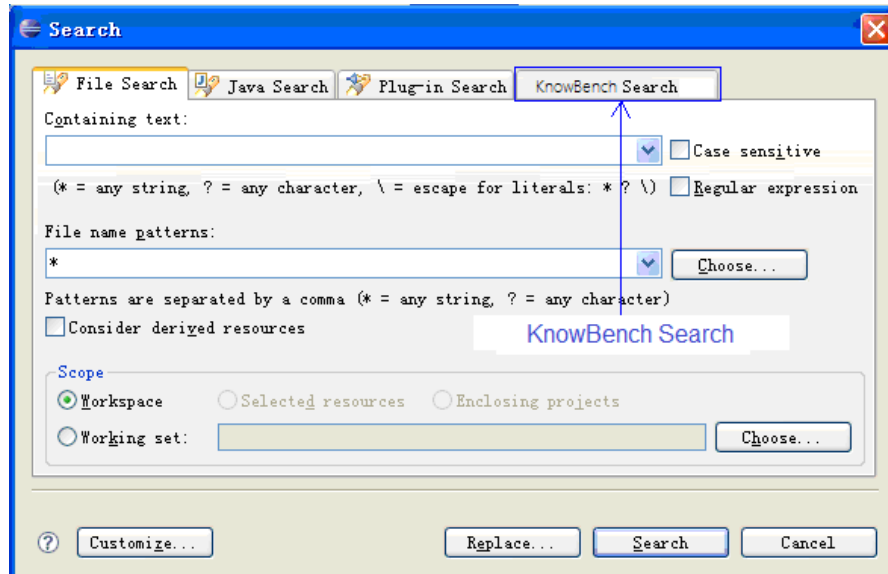


Figure 6.59: KnowBench Eclipse Search Integration

The user can search for any kind of knowledge: informal (i.e. indexes in “integrated” index storage), formal knowledge (i.e. ontology-based entities in the metadata repository), or combination of them. Consequently, three types of search are supported:

keyword-based search;

structured search;

semantic search.

Keyword Based Search

Keyword search component realizes a Lucene-based traditional paradigm of search, with an advanced functionality of indexing different sources of relevant information for software engineers, such as SVN repositories or JIRA issues.

The dialog for the keyword search is shown in Figure 7.60. The user should type one or more keywords to search for, select value “indexed based” for the Search Type and press

the Search button. Once the search is completed the results of the search are presented in the Keyword Search Results view, which is described in section 7.6.3.

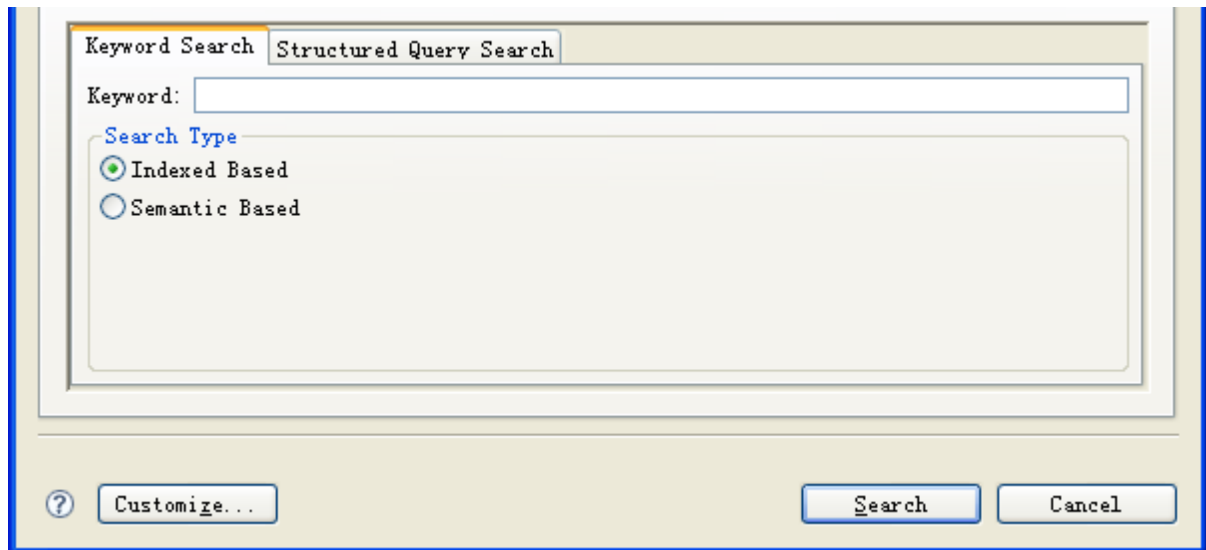


Figure 6.60: Keyword Search Dialog

The prerequisite for keyword search is the existence of the index file. In the case that the index does not exist, the user will be informed about that as shown in Figure 7.61.

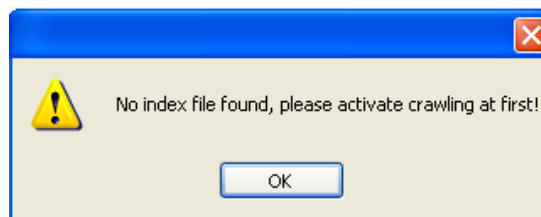


Figure 6.61: Keyword Search Problem

Structured Search

While keyword-based search introduced in the previous section represents the standard model for search interfaces, structured search allows more precise queries which in turn yield more precise results. Structured search is the search defined through ontology entities. Figure 7.62 shows the different sections and options available to the user when expressing a structured query. As can be seen, users do not need to have any background knowledge on the ontology as they can directly define through the UI the specific classes and the properties that they are looking for. The classes and the properties are shown in the Knowledge Base (i.e. left-hand side) and Properties parts (i.e. right-hand side) of the Figure 7.62 respectively.

The creation of ontology-based queries based on the user's specifications is done automatically.

The user can choose whether he or she wants to search in a local repository or in a corresponding KnowBench P2P network. For the P2P search, the user has to check the "Search in P2P networks" button and select one of available P2P networks (see P2P scope in Figure 7.62). He/she can also specify the value for the timeout parameter which is used for the search over the P2P network. The end-user has the possibility to control this value by using the "Search Timeout" spinner. If the P2P search takes longer, the end-user can change the value of this parameter to guarantee that the search can be completely performed.

In the case that there is no P2P connection, the right-bottom part of Figure 7.62 will not be shown.

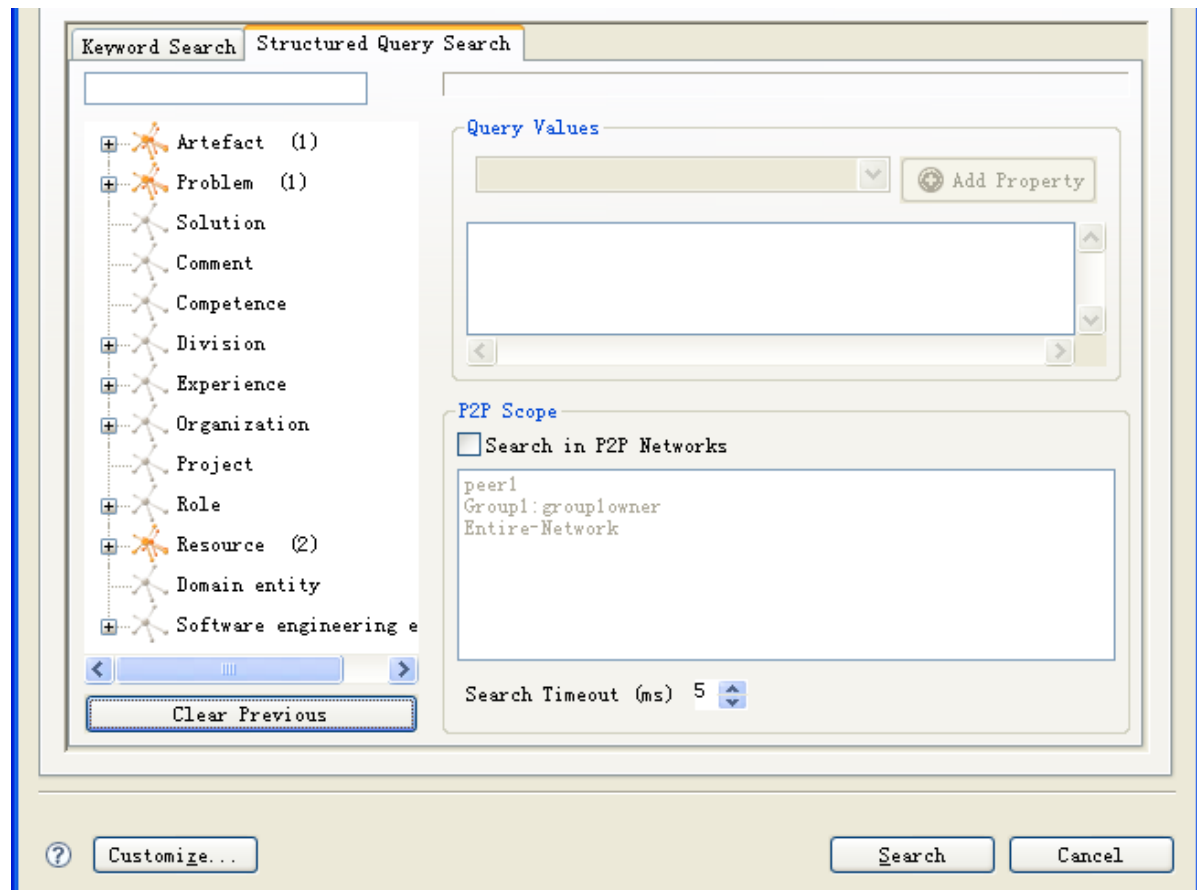


Figure 6.62: Structured Query Search Dialog

To specify the query, the user should find the most appropriate class in the knowledge tree. Once the class is selected, the user should add properties to the structured query by selecting them from the property combo box and pressing the Add Property button.

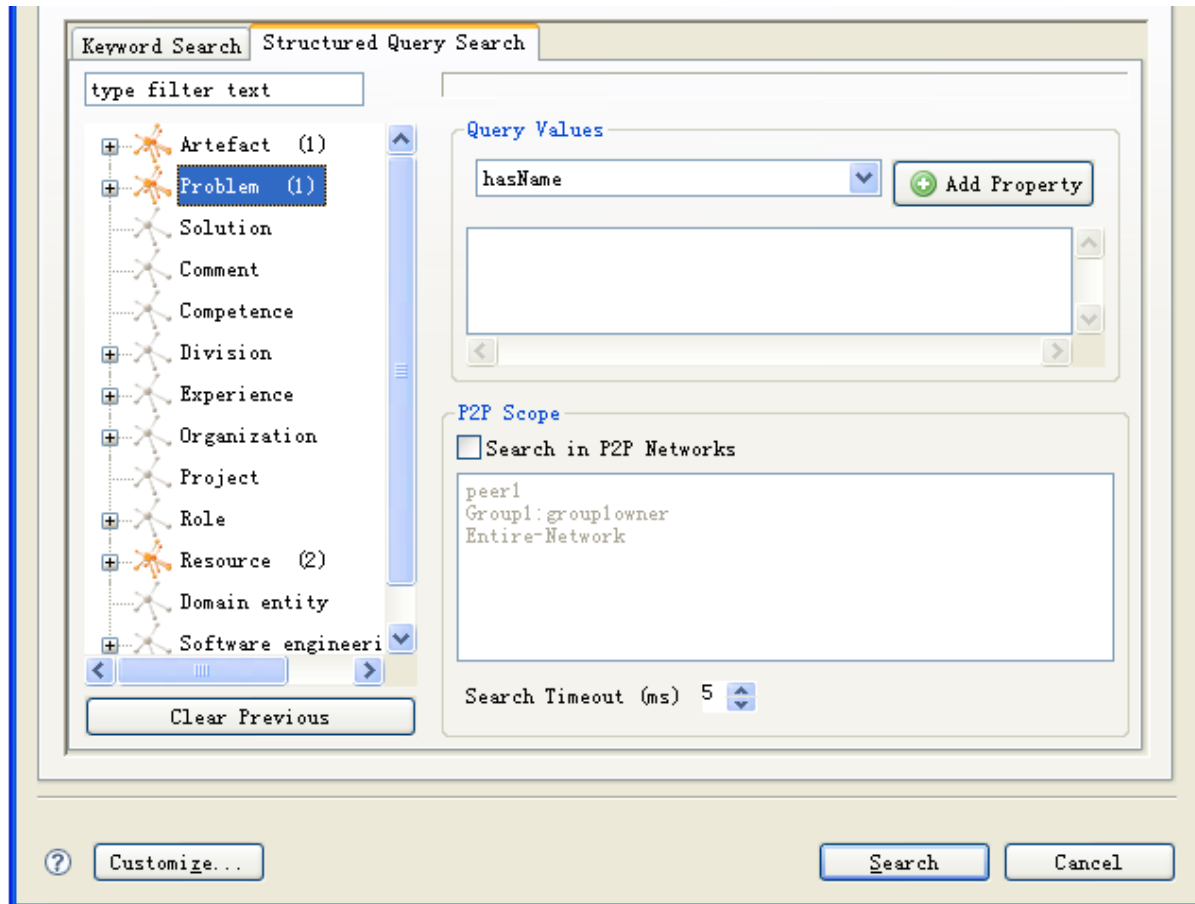


Figure 6.63: Structured Query Search Dialog – Adding Properties to the Query

The added property will be displayed in the Query Values list (see right-upper part of Figure 7.64). Depending on the type of information required for the added property, different widgets will be presented. For object properties the user should select one of the existing individual from the corresponding combo box. For the data properties values can be added by using number widget for numeric properties and text field for all other properties (see “hasName” in Figure 7.64).

To delete already added property, the user should click the  button.

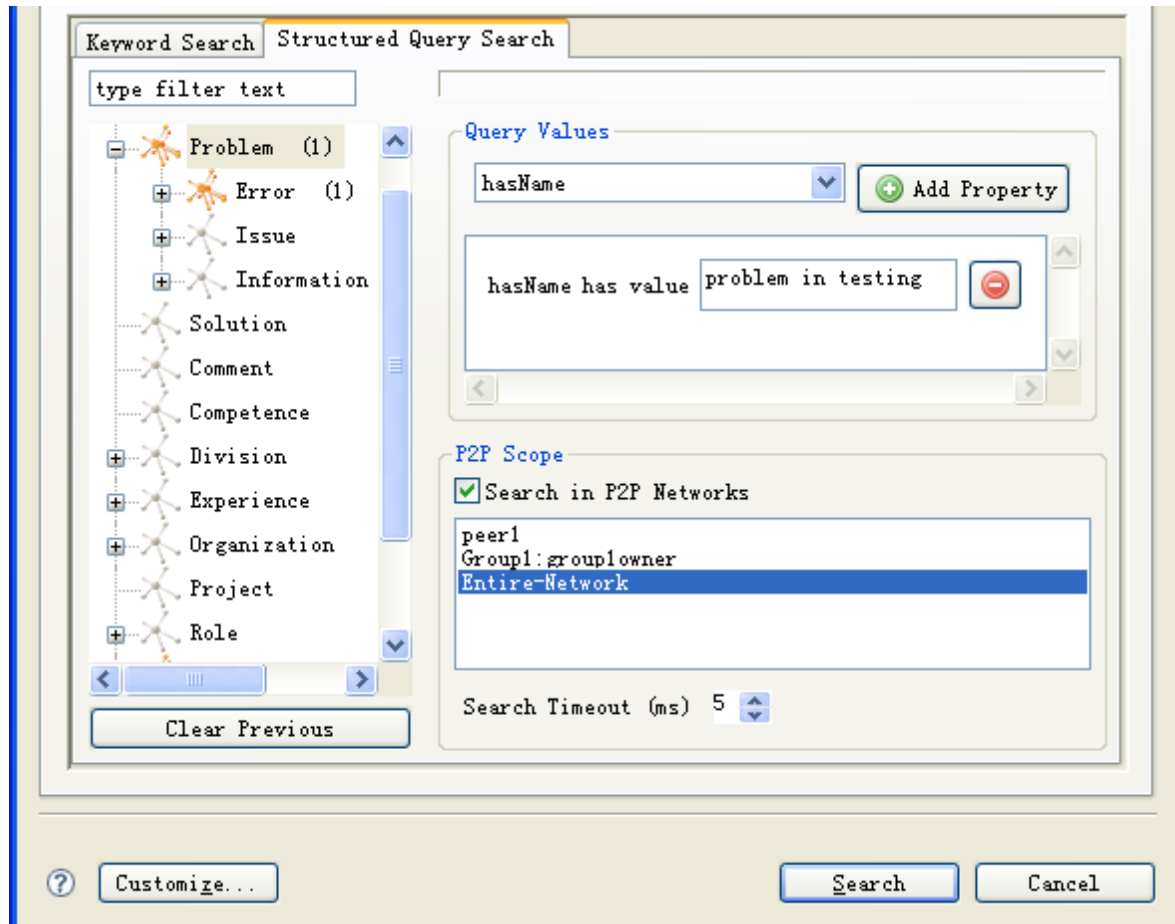


Figure 6.64: Structured Query Search Dialog – Property Values

Finally, the user should press the Search button. Once the search is completed the results of the search are presented in the Structured Search Results view, which is described in section 7.6.3.

Semantic Search

Semantic search is the search which automatically translates keyword queries into formal logic queries so that developers can use familiar keywords to perform structured search without having to navigate in advance through the ontology tree.

The same dialog as for the keyword search is used (see Figure 7.60). The user should type one or more keywords to search for, select value “semantic based” for the Search Type and press the Search button. The KnowBench system computes all possible query interpretations in the background. Once the search is completed the results of the search are presented in the Semantic Search Results view, which is described in section 7.6.3.

7.6.3 Views

In the KnowBench system developers are not only contributors of knowledge but also consumers. It is therefore crucial to provide them with means for interacting with this knowledge. The KnowBench system allows navigation through the search results, existing annotations, as well as through the Wiki. In the following, these views in more detail are described.

Search results view

Keyword search results

The results of keyword search explained in section 7.6.2 are shown in Figure 7.65. The user can click on the link to browse to the selected search item in the semantic wiki. Additionally, by right-clicking on the link he/she can browse via the semantic wiki (in the browser or edit mode) or the knowledge base editor (see section 7.6.1). This is shown in Figure 7.66.

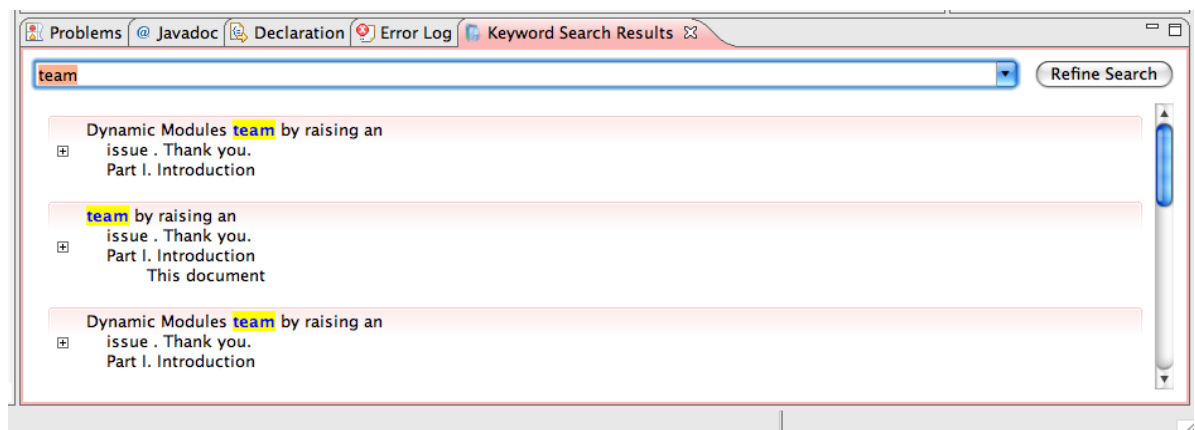


Figure 6.65: KnowBench Keyword Search Results View

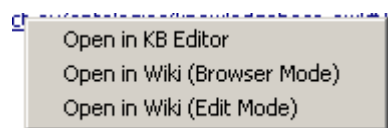


Figure 6.66: Relationships between Search Results view and other KnowBench options

The user is also able to refine his/her search by selecting one of the proposed terms in the combo box shown in the upper part of Figure 7.67 and by pressing the Refine Search button.

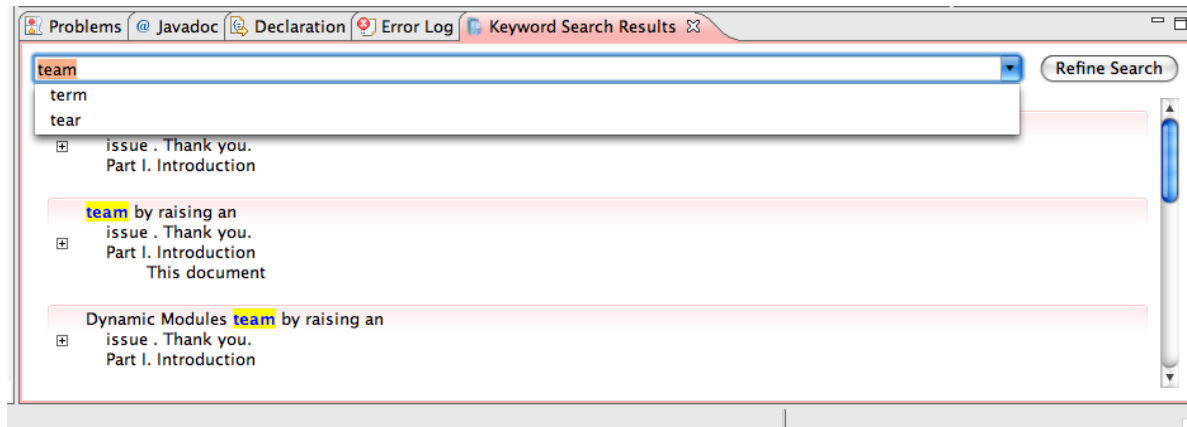


Figure 6.67: Keyword Search Results Refinement

The results of the query refinement are shown in Figure 7.68.

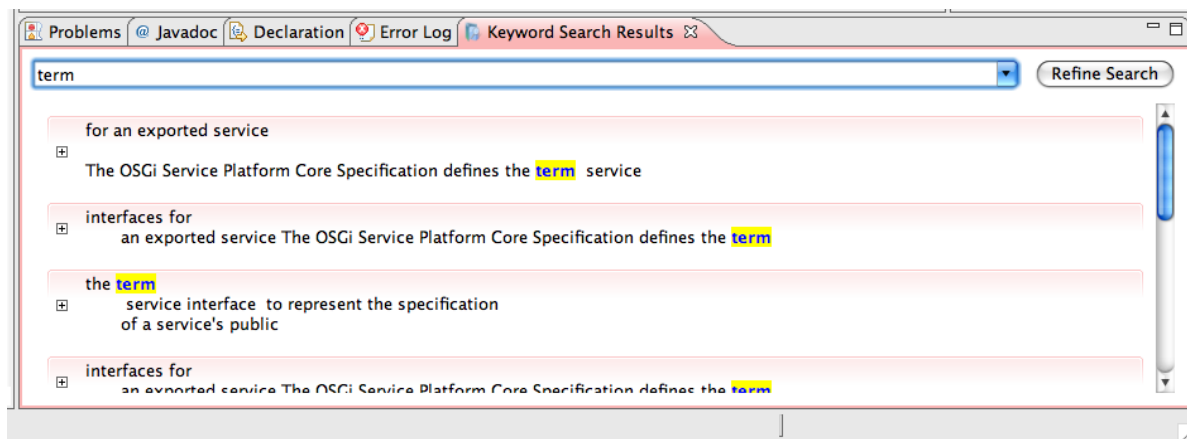


Figure 6.68: Keyword Refined Search Results

Structured search results

The results of the P2P structured search explained in section 7.6.2 are shown in Figure 7.69. As in the case of the keyword search results, the user can click on the link to browse to the selected search item in the knowledge base editor.

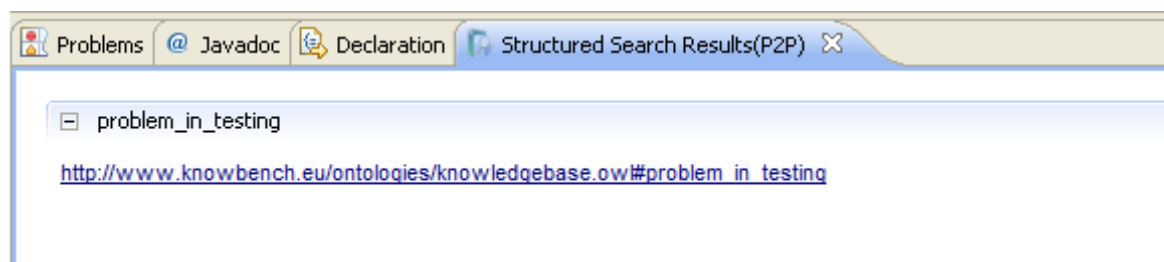


Figure 6.69: Structured Query Search Results Dialog – P2P Search Results depicting the solution found for the query shown in Figure 7.64

The P2P search has actually 2 phases. Figure 7.69 shows only the 1st phase, retrieving a list of individuals. The second phase (retrieving the attributes) is shown in Figure 7.70. The user can click on the links in the P2P results shown in Figure 7.69 and a window shown in the upper part of Figure 7.70 appears.

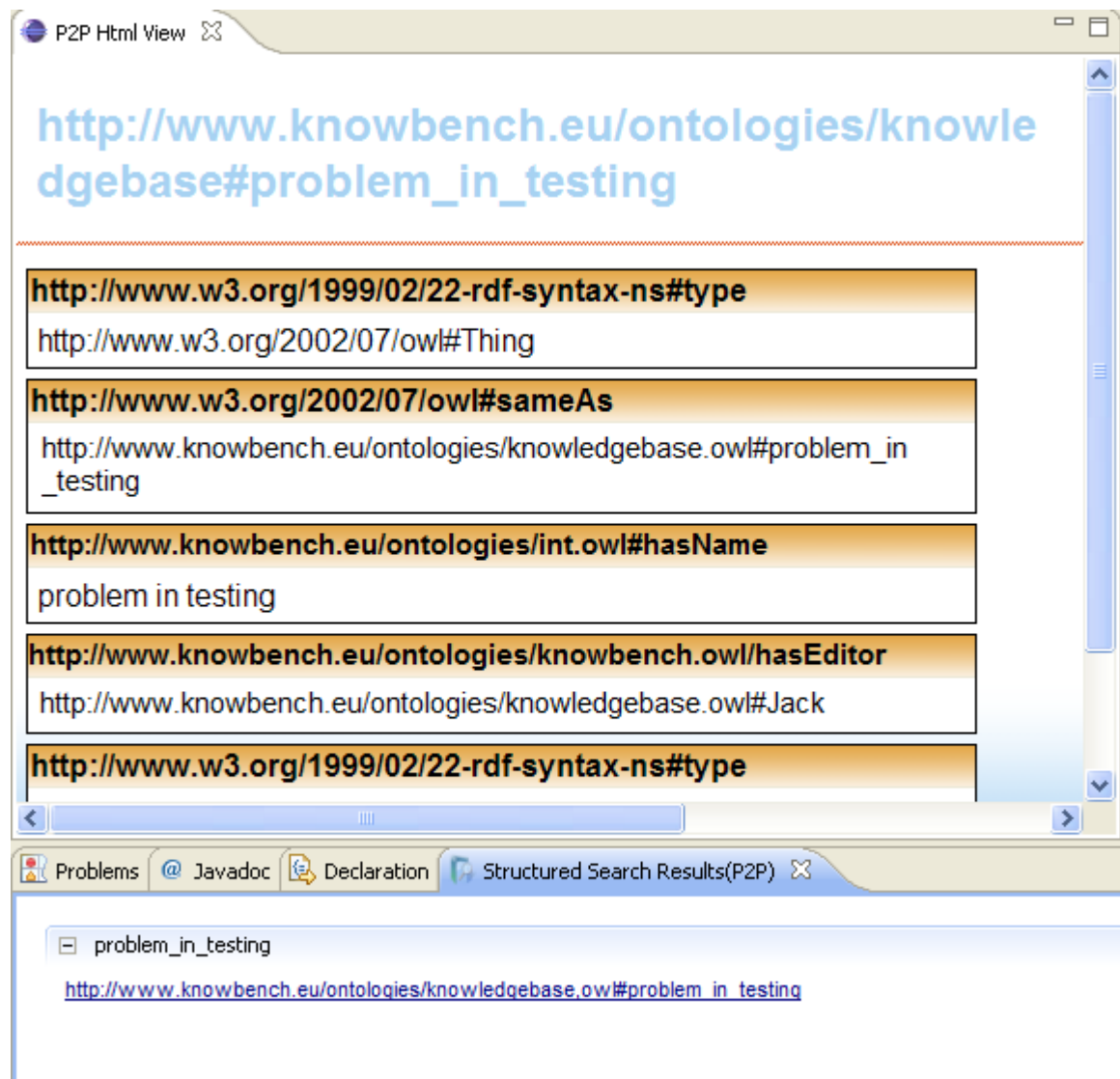


Figure 6.70: Retrieving attributes for the P2P search results

Semantic search results

The results of semantic search explained in section 7.6.2 are shown in Figure 7.71.

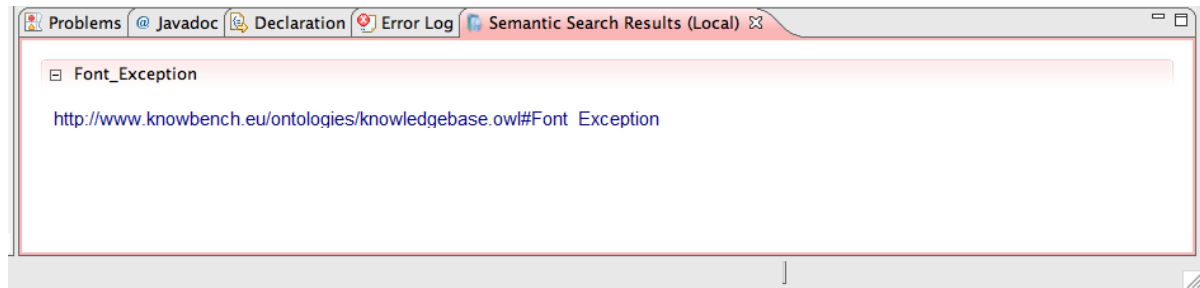


Figure 6.71: Semantic Query Search Results Dialog

Annotation view

Creation of annotation is described in section 7.4.2. The existing annotation can be seen by using either the toolbar or the menu (Figure 7.72). The prerequisite for viewing annotation is that the respective source code file is opened. The file that is annotated is shown on the left-hand side in Figure 7.73. It is then highlighted where annotations have been defined.

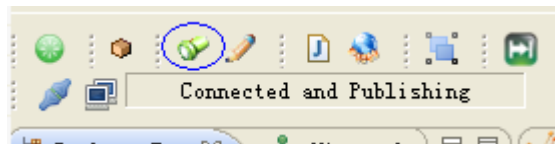


Figure 6.72: View Annotation Toolbar Button

The annotation view itself is shown at the bottom part of Figure 7.73. For each existing annotation, the following information is shown in the view:

Code entity type – the ontology class that is instantiated;

Code entity name – the part of the code that is annotated;

Domain annotations – list of entities from the KnowBench Domain Ontology used for the annotation;

Software Engineering annotations – list of entities from the KnowBench Software Engineering Ontology used for the annotation.

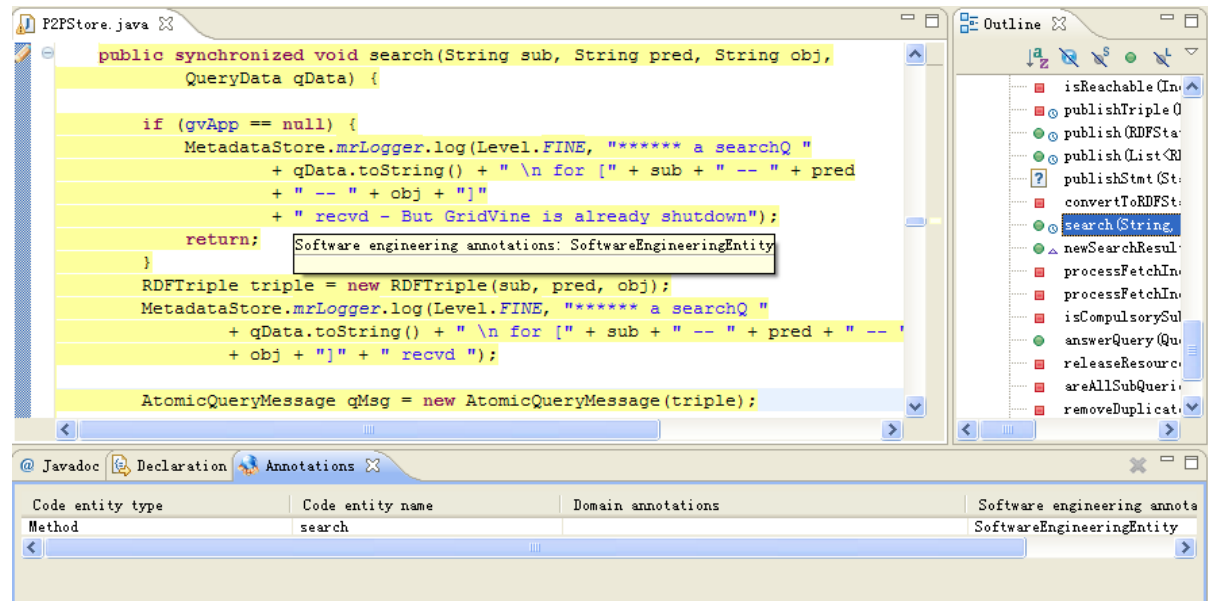


Figure 6.73: Annotation Views

Wiki browser

The wiki browser shown in Figure 7.74 is an alternative way to navigate through the knowledge base besides the knowledge base editor or graph browser (see 7.6.1). It can be opened in two ways:

by selecting Open in Wiki from the context menu e.g. of the search results view (see Figure 7.66) or

by following a link to an instance from the wiki browser.

It displays the properties of an instance in an HTML fashion. Only properties that are instantiated are shown. In the case of the object properties, the user can click on the values. As a result, another wiki browser will be opened with the contents of the instance clicked.

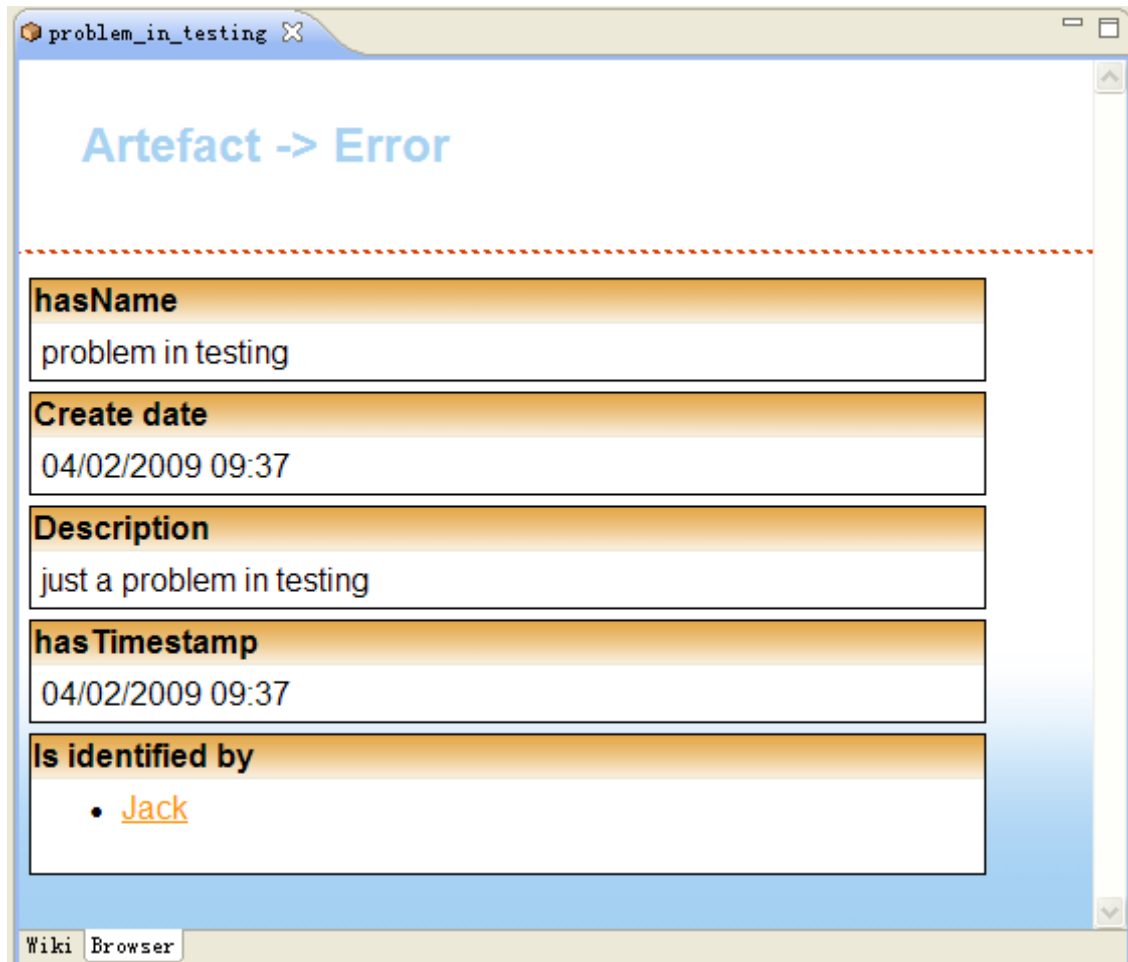


Figure 6.74: Wiki Browser

7.7 Knowledge Preservation

Creation of knowledge in the KnowBench system is described in section 7.4. In this section the way knowledge can be modified and removed is described.

Management of Knowledge items

The KnowBench system facilitates the modification and deletion of knowledge items through the knowledge editor as shown in Figure 7.75. In order to modify or to delete an instance, the user should first select a class in the knowledge tree (see left-hand side in Figure 7.75). Then he/she should select an instance from the “Instances” list (see upper-left part of Figure 7.75). This will enable modification of selected individual and its deletion.

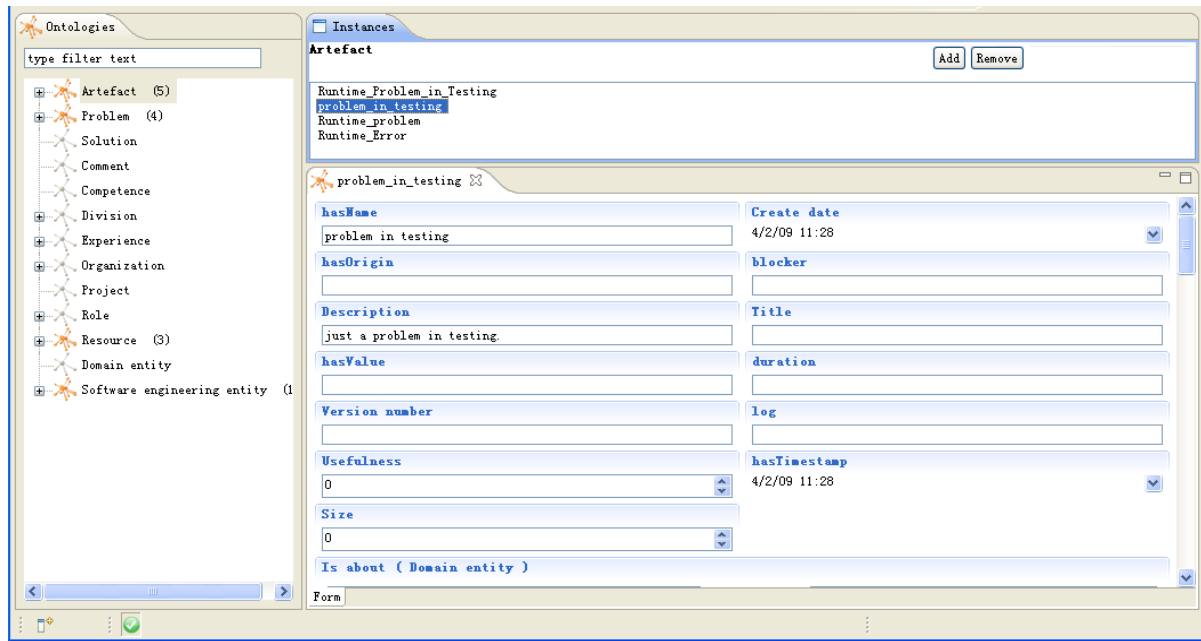


Figure 6.75: Modification of Individuals

To modify an individual, the user has to double-click on the selected individual on the right-upper side in Figure 7.75. A form shown in the right-bottom side is dynamically generated for the properties of the selected instance and pre-populated with the current values of that instance.

The user is able to change the values of all properties. The name of the individual (i.e. its URI) cannot be changed, since it is used as an identifier for knowledge items. A change in the values of properties is indicating by the symbol “*” in the header of this individual. This is shown in Figure 7.76.

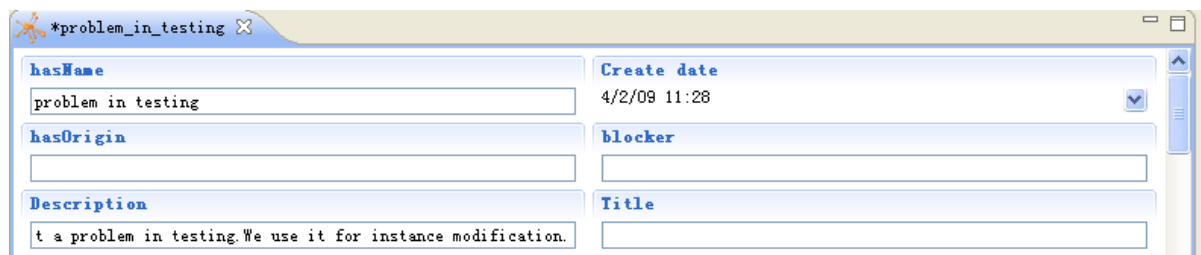


Figure 6.76: Modified Individual

To persist the modifications made to the instance, the user must press the “Save” button or close the form of the individual. The user will also be asked to confirm the modification as shown in Figure 7.77. Once the change is stored, the symbol “*” in the header of the individual disappears.

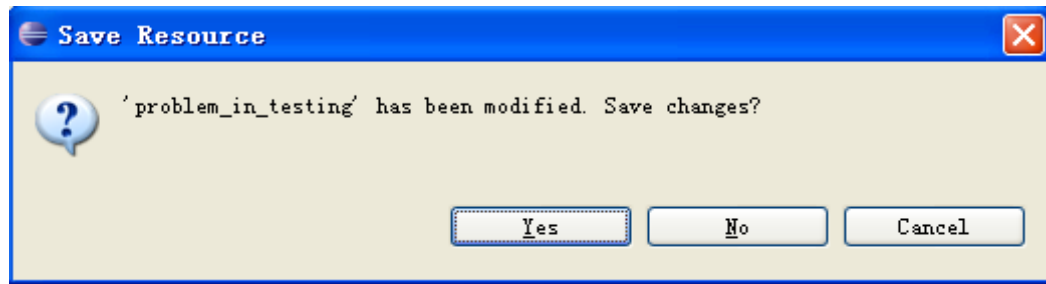


Figure 6.77: Confirmation of Modification

To remove individual(s), the user should select them in the list of individuals on the top-right side in Figure 7.78 and press the Remove button. In order for removal to be performed, the user has to confirm it. This is shown in Figure 7.78. After clicking the OK button, the selected individual(s) will be removed from the knowledge base and the number assigned to the class of these individuals will be decreased.

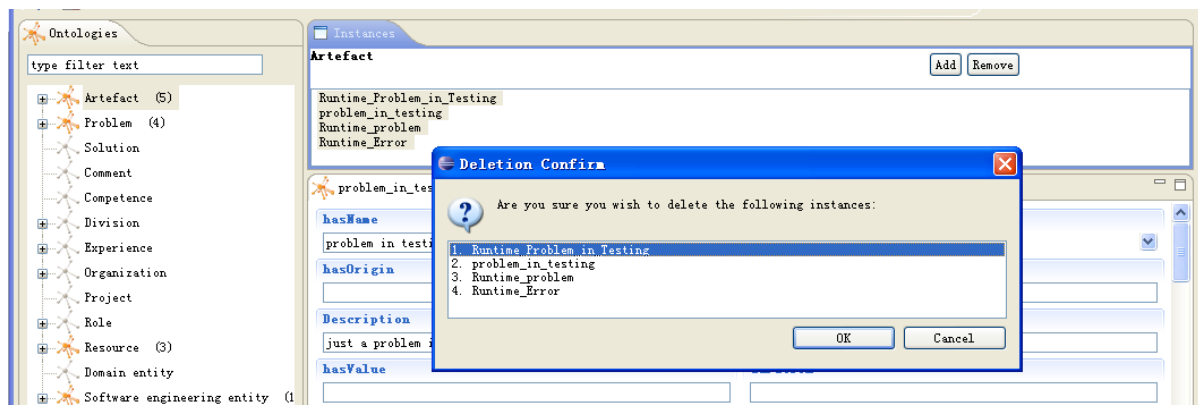


Figure 6.78: Removal of Individuals

Management of Annotations

Annotations can be only created or removed. To remove an annotation the user can use either the Eclipse toolbar/menu or the context menu of the source code editor.

Management of Wiki

The Wiki part of the KnowBench system provides access to the knowledge base and gives an alternative way to browse, insert, modify or remove instances of the knowledge base. As shown in Figure 7.79, the information of an instance can be completely changed by editing the values represented in the wiki editor. If the user changes the concept of the instance, then the user will be requested to save this information as a new instance.

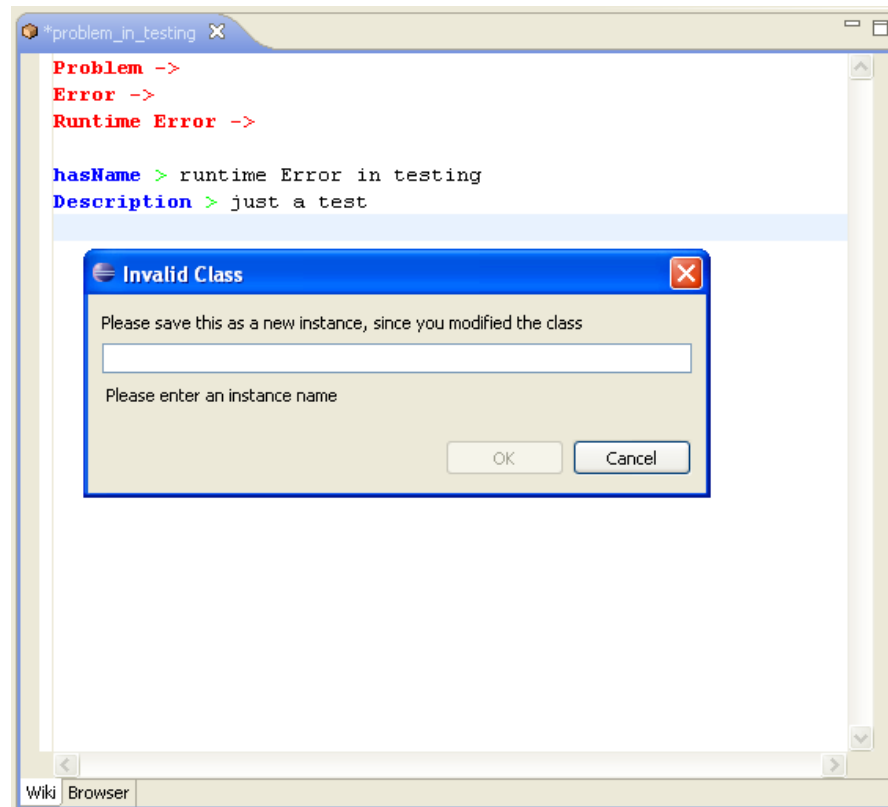


Figure 6.79: Editing a Wiki Page

8 EVALUATION OF KNOWBENCH

Evaluation targets the "systematic investigation of the worth or merit of an object" (Frechtling, 2002). Worth or merit relates to a specific purpose of an evaluation. Such purposes are to collect information that helps to improve an object, to provide a data basis for assessment and decision or to derive new insights about the usefulness and behavior of the object under certain conditions.

There are various approaches to carry out evaluation, mostly differing in their methods for data collection and analysis and in their stage of the object lifecycle. Typically, two kinds of evaluation are distinguished: formative and summative evaluation. Formative evaluation is on-going and iterative and yield direct feedback for improving the object, while summative evaluation assesses "the quality and impact of a fully implemented project" (Santos, Albuquerque, & Meira, 2008).

Thus, regarding the goal of investigation, formative evaluation strives to collect information for project improvement during development, while summative evaluation is thought to derive insights of the efficacy (its ability to do what it was designed to do) and its influencing factors, after the final product has been created.

This chapter reports on all pertinent aspects of the KnowBench summative evaluation. It starts with the details of the evaluation framework. The GQM method is shortly described and also the main areas of evaluation i.e. GQM Goals. Then, the method of analysis applied on the collected feedback for the calculation of the evaluation goals is described. Finally, the collected feedback from four pilot organizations (INTRASOFT, LIPSZ, THALES, TXT) and the interpretation of the observed trends in users' responses are presented.

8.1 The GQM Method

In summative evaluation, the efficacy of the final product is usually derived by comparison to previously stated goals. Thus, in this chapter such goals and the further process of how to measure and analyze the actual outcome of measurements are described.

For the purpose of defining the structure of the summative evaluation for KnowBench, the Goal-Question-Metric (GQM) method (Basili, Caldiera, & H. D. Rombach, 1994) was

used, which provides a process framework for evaluating software systems. GQM was developed by Victor Basili of the University of Maryland, College Park and the Software Engineering Laboratory at the NASA Goddard Space Flight Center. It defines a measurement model on three levels (Figure 8.1):

- Conceptual level (goal): A goal is defined for an object for a variety of reasons, with respect to various models of quality, from various points of view and relative to a particular environment.
- Operational level (question): A set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.
- Quantitative level (metric): A set of metrics, based on the models, is associated with every question in order to answer it in a measurable way.

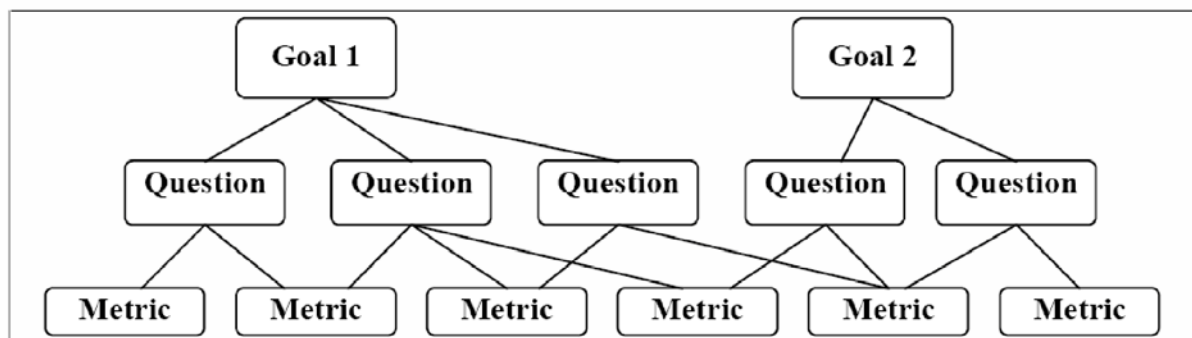


Figure 7.1: Schematic overview of the GQM approach (Basili et al., 1994)

The GQM process consists of four phases (Figure 8.2):

- The project plan
- Definition of goals, questions and metrics
- Data collection
- Data analysis under consideration of goals and questions.

A set of questions and metrics have been developed to assess the goals of KnowBench. A questionnaire was then used for eliciting respondents' feedback. Data collection took place by deploying four pilots of the KnowBench system, one at each industrial partner's site.

The industrial partners for deploying the pilots were:

- a Brussels based company specializing in the field of Information and Communication Technology (ICT) services (Intrasoft International S.A. – 4 developers),
- a leading hungarian association dealing with open source software at corporate level (Linux Industrial Association – 4 developers),
- an italian company which operates in the Information Technology market, focusing on business applications (TXT e-Solutions – 4 developers) and
- the corporate research laboratory of the Thales group – a global electronics company serving Aerospace, Defence, and Information Technology markets worldwide (Thales Research & Technology – 4 developers)

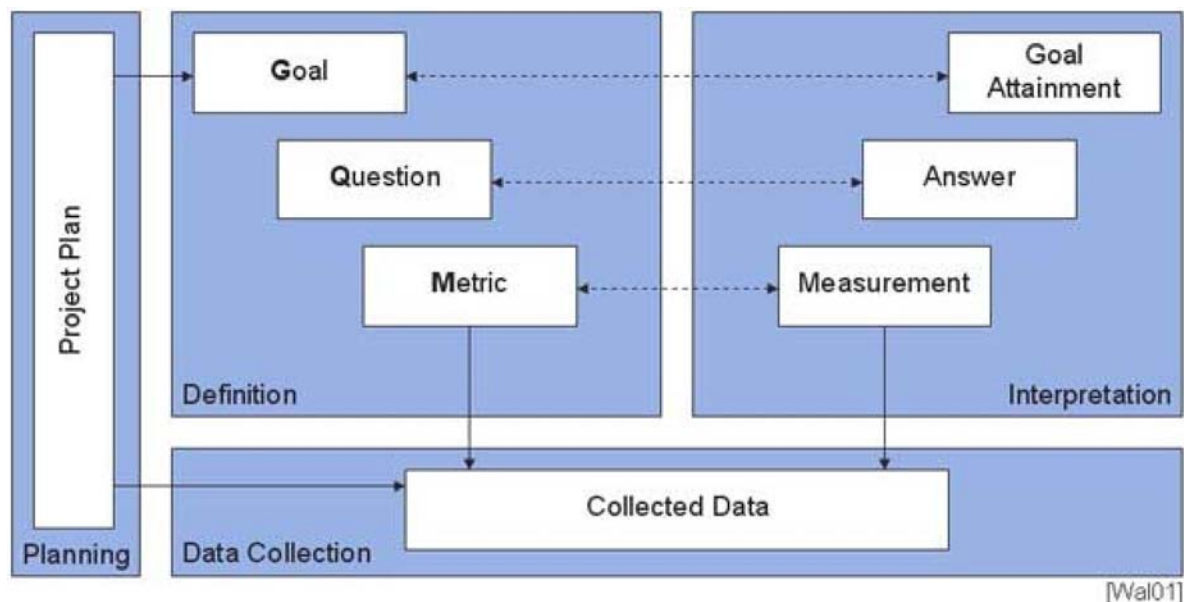


Figure 7.2: Phases of the GQM process (Basili et al., 1994)

In the interpretation phase, results from the data collection were compared to the defined metrics. Based on this, the overall level of efficacy of the KnowBench system was determined, by looking at the attainment of the project goals.

8.2 Areas of Evaluation (Goals)

The KnowBench pilots comprised a complex endeavour of IT intervention. In this respect, the summative evaluation aimed at capturing feedback for both systemic and organizational aspects. Practically this translates to the following two areas of evaluation:

- the **knowledge empowerments in the software development practice** of the organization, in order to assess the benefits of using the KnowBench system for the support of pertinent knowledge processes i.e. sharing, etc., and
- the **KnowBench system itself**, since its successful application depends on its endorsement by the members of the organization.

All the areas that the trial members provided feedback for are briefly described below.

8.2.1 Basic Perceptions of Knowledge Management

Knowledge Management might be around for a long time but it is always surprising to discover how many people might not have heard of it or know what it is used for. This is very common in professional fields where KM has not yet become a standard practice. Software Engineering is one of them and hopefully KnowBench would pave the way for similar initiatives. The investigation of people's knowledge about KM and their perception of KnowBench as a KM system was based on the following questions:

- What is your perception of Knowledge Management?
- Do you have experience with Knowledge Management systems (used before)?
- What were your expectations from the KnowBench knowledge management system?
- Would you agree with the entitlement of KnowBench as a knowledge management system?
- Do you find the KnowBench knowledge management functionalities sufficient for the needs of your job/practice? If not, please explain.

8.2.2 General Perceptions of KnowBench's Efficiency and Effectiveness

KnowBench's capacity to fulfil its KM promise comprised a major challenge and hopefully the lifecycle questions have contributed to this direction. Measurement, however, of KnowBench's impact on work practices and human behaviours would require a longitudinal and in-depth inquiry (that is by the way partially also true for the measurement of the lifecycle aspects). Eliciting meaningful insights about the impact of KnowBench in people's jobs through a set of carefully designed qualitative questions that address basic adoptability and users' satisfaction issues is important. Such issues refer to:

- Easiness of use
- Improvements in work
- Integration ability

8.2.3 The System

Evaluation goals for the KnowBench system itself are primarily formulated based on software quality attributes. A typical classification for this is the FURPS approach⁵⁹, considering functional as well as non-functional requirements (usability, reliability, performance, supportability, etc.). The system measures are described below:

- Usability: The capability of the software product to be understood, learned, used and be attractive to the user, when used under specified conditions.
- Reliability: The capability of the software product to maintain a specified level of performance when used under specified conditions.
- Scalability and Availability: The capability of the software product to be modified/extended but also transferred from one environment to another.
- Security: The capability of the software to safeguard users' knowledge when used in a shared environment (KnowBench case).
- Deployment: The capability of the system to be set- and unset-up easily.

8.3 Method of Analysis

The GQM method is based on the idea of goal attainment. This is determined through the summative account of the collected replies for the set of chosen questions. This section describes the metrics adopted for determining the positive and the negative contributions towards the attainment of the goals as well as the method for aggregating results within each goal context.

⁵⁹ Functionality, Usability, Reliability, Performance and Supportability model

The questionnaire adopted for the elicitation of the respondents' feedback was implemented on the basis of three types of questions:

- a) Questions of 1..5 likert scale (negative to positive orientation),
- b) Questions of "Yes" or "No", and
- c) Questions of open end.

The first two types of questions were subject of the quantitative analysis, while the latter group was used to formulate the ideas for the improvement of the KnowBench system. For the Likert scale questions, answers rated with 1 and 2 are considered as negative while those rated with 3-5 as positive contributing to the attainment of a goal.

The questionnaire is divided in sections reflecting the various areas of evaluation as described above. Questions in each section contribute to the measurement of KnowBench's capacity to fulfill the corresponding theme/goal i.e. support knowledge identification, etc. Therefore the analysis and the subsequent interpretations are provided per section/evaluation area.

The measurement of the goal is derived after the aggregation of the average of all positively-and negatively-contributing answers in each section.

Due to the different ranges of type a) and b) questions, the final aggregation is done using the weight of each type of answer in the total measurement. The weight is determined by the number of questions of the same type in a section.

For example: If there are 22 type a) questions and 2 type b) questions then the final (goal) calculation looks like this:

Goal Measurement formula:

$(\text{SUM}(Z32:Z33)*22+AB27*2)/24$ where

Z32:Z33 is the range of the average of all negatively- or positively-contributing answers, and

AB27 is the average of all "Yes" or "No" answers respectively

Analysis of the data included the following processing (Figure 8.3):

1. Grouping of feedback data according to evaluation areas;
2. Integrity check of feedback data i.e. content purification, scaling alignment, etc.;

3. For each question, the total number of answers received for each point of the likert scale is calculated (how many respondents gave 1, 2, 3, 4 or 5 answer – similarly for the “Yes” and “No” answers);
4. The percentage of the “total number of answers received in each likert scale” in the total number of answers is calculated for each question;
5. The “total number of answers received in each likert scale” and the “total number of Yes or No answers” for all questions in each section is averaged;
6. Goal measurement is calculated using the above formula.

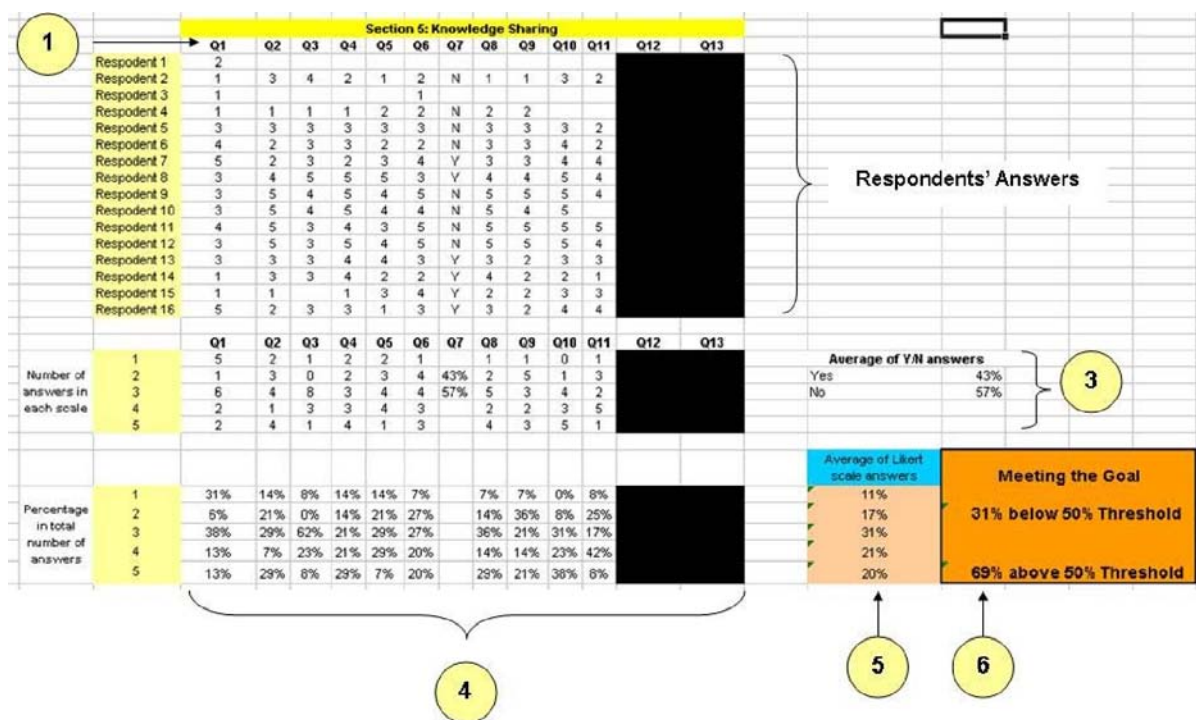


Figure 7.3: Example of Data Analysis

On the basis of the performed analysis, interpretations were formulated, which enclose attributions for the result of the goal measurement but also for questions that required further contemplation.

8.4 Basic Perceptions of Knowledge Management

Investigation about the level of understanding of the trial members about KM revealed a variety of viewpoints, all of them underlined by the acknowledgement of KnowBench as a potentially useful system for KM. All respondents demonstrated adequate awareness of Knowledge Management and in several cases they had experience working with relevant

tools i.e. Wikis, mediawikis, semantic search engines. Moreover, they seem to have received enough information about the KnowBench system prior to the trial. Their expectations, therefore, were to a great extent adapted to the KnowBench offerings. Sharing of knowledge among team members and creation of a common knowledge base featured as prominent areas of support by the KnowBench system.

On the practical side of the respondents' job, KnowBench made the expected impact as more than 70% found the KnowBench knowledge management functionalities sufficient for the needs of their job/practice. Apparently, the respondents' well informed state about KM combined with the probably increased expectations for KnowBench system have negatively influenced their perception on this matter.

8.5 KnowBench support for Knowledge Management (lifecycle) Functions

8.5.1 Knowledge Identification in KnowBench

Knowledge Identification - Questions													
Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
93%	57%	94%	93%	92%	53%	80%	7%	0%	69%	100	92%	-	-
60 %													

Table 7.1: Percentage of answers above threshold

The users' acceptance of knowledge management systems heavily depends on how knowledge they are dealing with has been identified. Within the KnowBench system, the KnowBench ontologies model the critical knowledge relevant for software development process. Thus, the quality of the knowledge in KnowBench is assured by the KnowBench ontologies that have been iteratively developed by taking into account the real needs of end users. This is visible in the results of evaluation of knowledge identification aspects of the KnowBench (Figure 8.4). 69% of answers related to the knowledge identification are above

the threshold (set to 50%). KnowBench ontologies were stable during trial for the 69% of respondents (Q10). The respondents appreciated a lot the possibilities to use hierarchies either for describing the content (Q3 - 94% of respondents) or for navigation (Q4 - 93% of respondents).

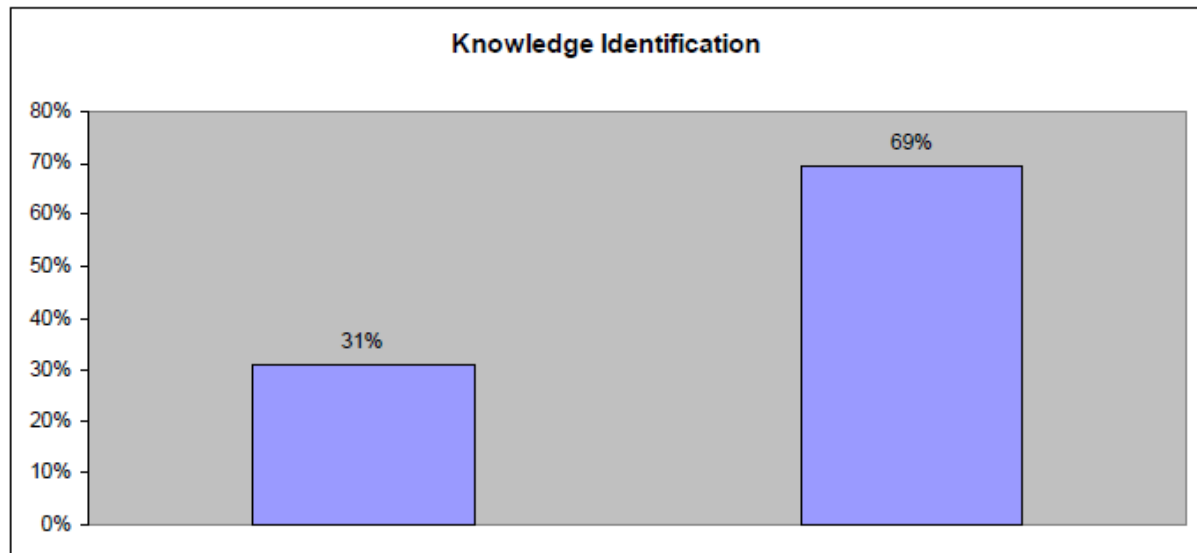


Figure 7.4: Goal measurement for Knowledge identification

However, even through the 93% (Q1) of respondents agreed that the level of detail of knowledge created in KnowBench was adequate and no classes/properties are missing, the 57% (Q2) of them found that a part of the information/knowledge that could be of use to them was not captured in KnowBench. It means that the meta-model for capturing knowledge (i.e. KnowBench ontologies) is rich enough for modelling all relevant aspects, but its instantiation (i.e. the knowledge base) did not contain all relevant knowledge. Consequently, quality of knowledge was good, but not the quantity.

The low quantity of KnowBench knowledge base could be a consequence of (i) the fact that knowledge creation is a time-consuming activity and/or (ii) reluctance of respondents to provide it. Whereas the first problem can be alleviated by extending the duration of trial, the resolution of the second problem requires creating awareness of the impact of captured knowledge e.g. by explaining the goal of the knowledge management in

⁶⁰ Q13 and Q14 are excluded from the goal measurement calculation as they enclose interpretive capacity only. In subsequent sections, all cells in black should be regarded as not contributing to the measurement of the goal.

general, the KnowBench system, the role of ontologies, etc. Indeed, the 53% (Q6) of respondents found that background knowledge of ontologies is required in order to effectively use KnowBench system. Some respondents suggested using something easier to understand than ontologies.

Additionally, it seems that respondents were not familiar with the terminology used in questionnaires, since some of their answers were contradictory. For example, the respondents who indicated that the KnowBench ontologies were stable during trials (Q10) also created many tags (Q13) which confirms that the relation of the newly created tags with the KnowBench ontologies was not understood. Some of respondents suggested that personal tags to be added by developers would help, although this functionality already exists in the KnowBench system. The conclusion would be that besides user guide, help and demos that give assistance to people using the KnowBench system, some teaching/exercises will be well appreciated.

Finally, there are 31% of the answers that did not reach threshold. For example, even though the respondents were satisfied with the richness of the class hierarchies (Q5), they did not use/instantiate all classes (Q8). Similarly, the proportion of properties instantiated during creating individuals was low (Q9). A more careful analysis of two “problematic” questions (Q8 and Q9) indicates these questions should be specified more precisely. For example, the usage/instantiation of a class means that all its parents are also considered. Besides generality of questions, the other reason for low marks could be that the respondents were not aware of the importance of having more knowledge about the domain. The usage of the KnowBench system in the long run can improve the satisfaction with the knowledge identification as more knowledge will be created in a more smoothly way.

8.5.2 Knowledge Acquisition in KnowBench

8.5.2.1 Knowledge acquisition of existing knowledge

Knowledge acquisition of existing knowledge - Questions

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
88%	100%	83%	92%	85%	77%	0%	92%	77%	33%	85%	64%	-	-

Table 7.2: Percentage of answers above threshold

According to the respondents, the KnowBench system achieves a good score for its support for acquisition of existing knowledge (Figure 8.5). Almost all respondents found that the crawling process is understandable and easy to follow (Q1, Q3). The respondents encountered no problems in understanding the purpose and results of the crawling process (Q4).

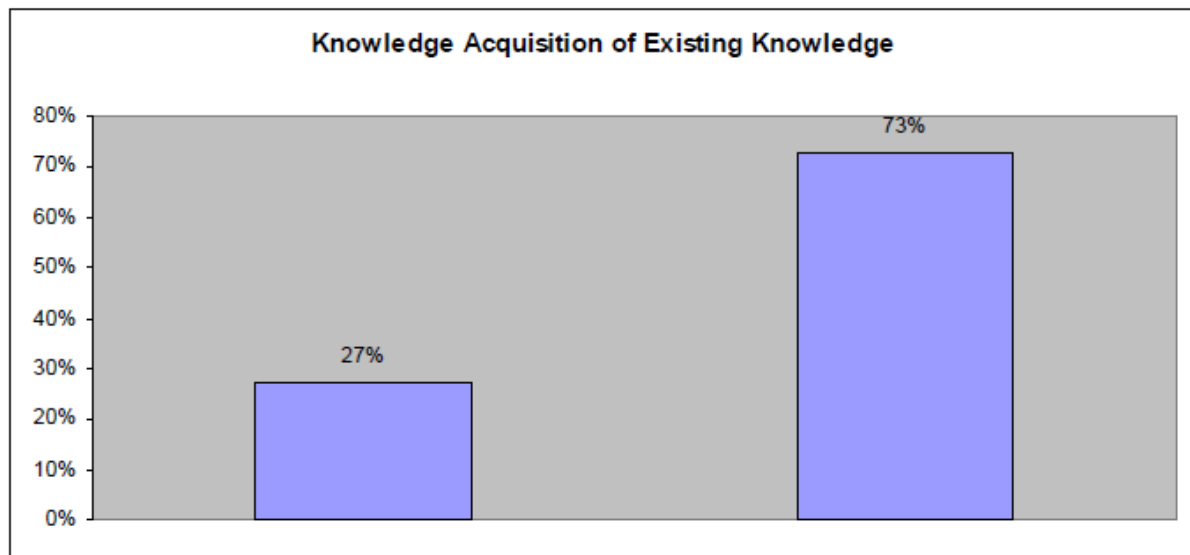


Figure 7.5: Goal measurement for Knowledge Acquisition of existing knowledge

However, there is still some potential for improvement. Even though it was easy for 92% (Q8) of respondents to identify which information should be provided in order to crawl a repository, 33% (Q10) of the respondents were not satisfied with the support for consistency checking of the provided input. As data validation/verification does not work always correctly, the recommendation would be to extend the KnowBench system with comprehensive validation checks as well as to prepare usage examples for defining the external sources of information.

Regarding the supported types of knowledge sources, 85% (Q5) of the respondents were satisfied with the support; only 23% (Q6) found that additional types of knowledge sources relevant for coding should be supported. However, there is no respondent who used either all available type of knowledge sources or suggested that the crawler has missing any.

In general, the respondents encountered some minor problems during the trial use of the system. 23% (Q9) of the respondents mentioned that the system presented occasional failures of not sending notification when the crawling is completed, which could be

attributed to the development nature of the KnowBench system. The recommendation here is to perform detailed testing based on respondents' feedback and to add more visual feedback.

As far as the system's response time, although 64% (Q12) of the respondents found it quite fast, some further optimization of the system would be useful.

8.5.3 Knowledge Development in KnowBench

Knowledge development - Questions													
Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
-	86%	77%	93%	79%	67%	92%	69%	83%	67%	57%	75%	46%	77%
Knowledge development cont'd – Questions													
Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24	Q25	Q26	Q27	
86%	86%	79%	100%	64%	92%	92%	82%	42%	46%	92%	-	-	

Table 7.3: Percentage of answers above threshold

All respondents found the knowledge development in KnowBench clear and easy to follow and agree that the KnowBench system provides support at the adequate level (Figure 8.6).

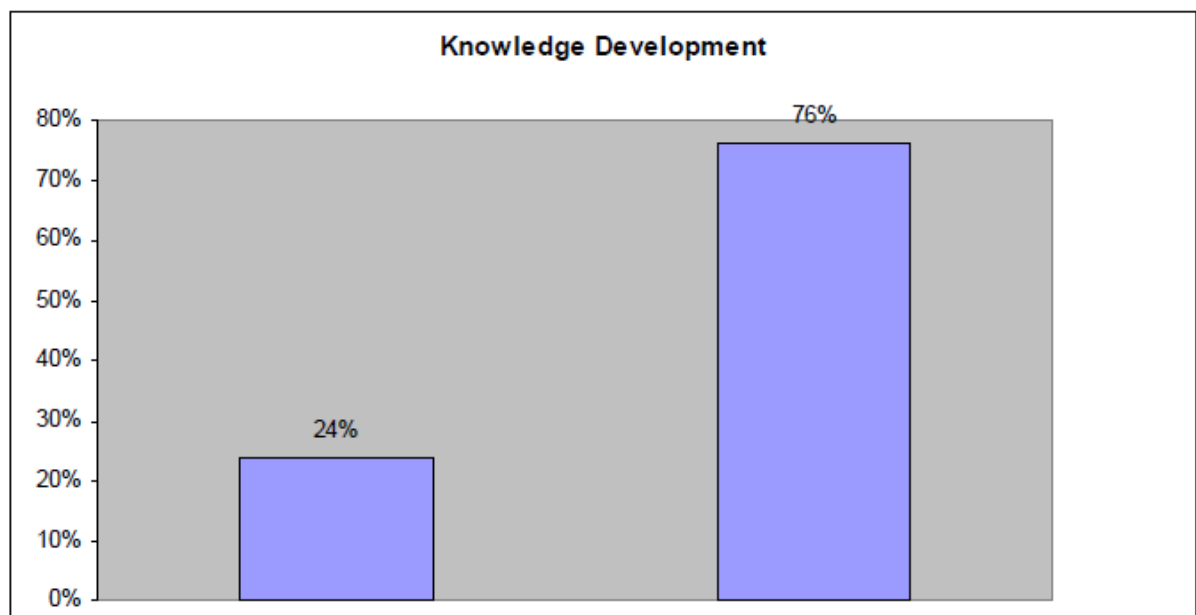


Figure 7.6: Goal measurement for Knowledge Development

The meaning of knowledge items is understandable for 86% (Q2) of the respondents. The KnowBench knowledge editor as well as the wiki-editor meets expectations of respondents. Creation of knowledge items was understandable and did not disturb developers during daily work. Even though the respondents agreed that the wiki-editor provides very good support for knowledge development (Q6), the usage of knowledge editor (Q5) is slightly more appreciated by the respondents. The main reason is that wiki syntax is not intuitive and a little bit more effort and time is needed as a user has to select all properties that should be instantiated instead of just finding them in a friendly and easy-to-use manner as it is the case with knowledge editor. The recommendation would be to provide more guidance in creating knowledge items using wiki-based editor, which is also indicated by the 46% (Q13) of respondents.

Regarding annotations, their meaning and purpose were clear for 86% (Q15 and Q16) of the respondents. 79% (Q17) of the respondents found the granularity level of source code that can be annotated sufficient. However, some respondents explicitly suggested more fine-grained granularity e.g. few lines in a method. This will be taken into account in future work as creating and maintaining unique identifier for a part of a code is a research challenge.

Respondents agreed that manual creation of annotations required more effort in comparison with automatic annotations. This is expected as they had to analyse the source code and to assign a tag to it. However, as 92% (Q25) of the respondents found that the KnowBench system provides friendly and easy-to-use forms for creating annotations and only 36% (Q19) of them considered the manual annotation as effortful activity, it seems that longer usage of the KnowBench system will alleviate this problem. The recommendation would be to prepare some demos showing not only how source code should be annotated but most importantly why it should be done. The demos should also include explanation of the selection/creation of tags.

The fact that only 42% (Q23) of proposed annotations were chosen, does not mean that the proposed annotations were not good. In contrary, 82% (Q22) of the respondents were satisfied with the suggestions for semi-automatic creation of annotations. As the KnowBench system enables developers to set up the threshold for the number of automatic annotations shown to them, it might be the case that low values were used for threshold and consequently too many possible annotations were shown. The other interpretation confirmed by the total number of created annotations would be that the respondents wanted to annotate some

domain specific aspects and not to take into account all results of NLP analysis of the source code.

According to the answers given by the trial users to the questions related to the knowledge development, the ontology management support would be appreciated. However, this is out of the scope of the KnowBench project as there are a lot of open source tools for creating/maintaining ontologies and knowledge bases.

On the other side, import and export of knowledge bases seems to be important and should work properly to correctly work with the KnowBench system and reuse existing knowledge. Additionally, the KnowBench system should be extended to be able to rename/delete described annotation concepts. Deletion of annotations should be improved especially if created by semiautomatic annotation. Automatically created annotation should be deleted one by one.

On the usability level, the KnowBench system should provide more information during editing a knowledge item by means of online documentation of properties, e.g. by tool-tips.

8.5.4 Knowledge Sharing in KnowBench

Knowledge sharing - Questions

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13
63%	64%	92%	71%	64%	67%	43%	79%	57%	92%	67%	-	-

Table 7.4: Percentage of answers above threshold

Knowledge sharing in KnowBench meets the expectation of 69% of respondents (Figure 8.7).

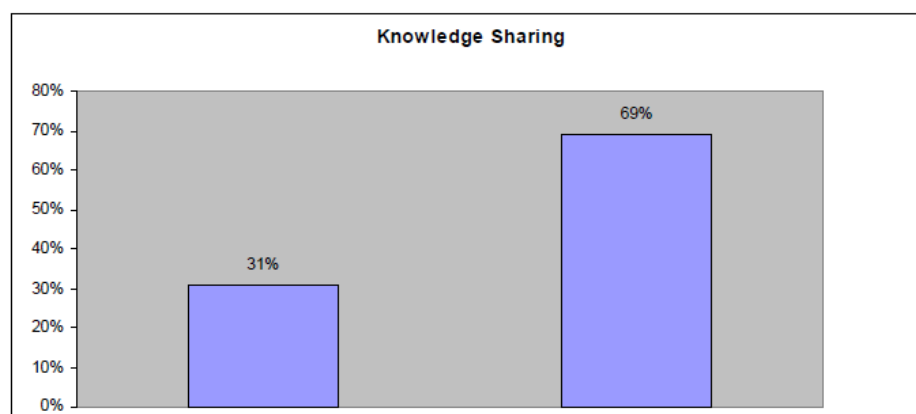


Figure 7.7: Goal measurement for Knowledge Sharing

Even though all aspects of knowledge sharing in KnowBench are above the threshold, only the level of details to be specified in order to share knowledge (Q3) and the usefulness of shared information (Q10) received high marks (greater than 90%). Other aspects received an average score. For example, the meaning of policy rules for knowledge sharing is understandable for the 71% (Q4) of respondents and 64% (Q5) of them found that creation of new policy rule for sharing is not easy.

A lot of suggestions for improvement were proposed:

As far as the P2P network is concerned:

- The P2P network should be faster, it requires a lots of machine and network resources and sometime computer had to be restarted;
- Response time should be highly increased as some times search over P2P network comes with no results even though knowledge exists;
- Access to P2P knowledge is slow and work on optimisation of system response time should be done;
- Security levels should be removed, as among developers the security is not an issue in sharing knowledge information;

As far as the policy sharing is concerned:

- Sharing policies should be more flexible, more generic/simple to be applied;
- The time required for defining properly policy sharing rules should be minimized;
- Automatic publishing (publishing of all annotations and instances) should be supported;

On the usability level , quite a lot of improvements should be made:

- Users should be able to categorize knowledge that he/she wants to access;
- In the knowledge browser, a possibility should be provided to easily share individuals (e.g. by means of a checkbox);
- It would be useful to have personal created folder to create/divide the knowledge to be shared. This would give developers more control on what they share;
- It should be possible to browse knowledge items entered by other persons.

All suggestions for the improvement of the functionality related to the P2P network are out of scope of the KnowBench system, as the P2P support is built on the work of the P-Grid, which is a research prototype.

8.5.5 Knowledge Usage in KnowBench

8.5.5.1 Search

Knowledge search - Questions

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
54%	82%	62%	82%	9%	91%	73%	36%	40%	46%

Table 7.5: Percentage of answers above threshold

The search functionality received an average score (Figure 8.8). The respondents used this functionality very often and had totally different opinions about it. All kinds of answers were given to all questions related to search functionality.

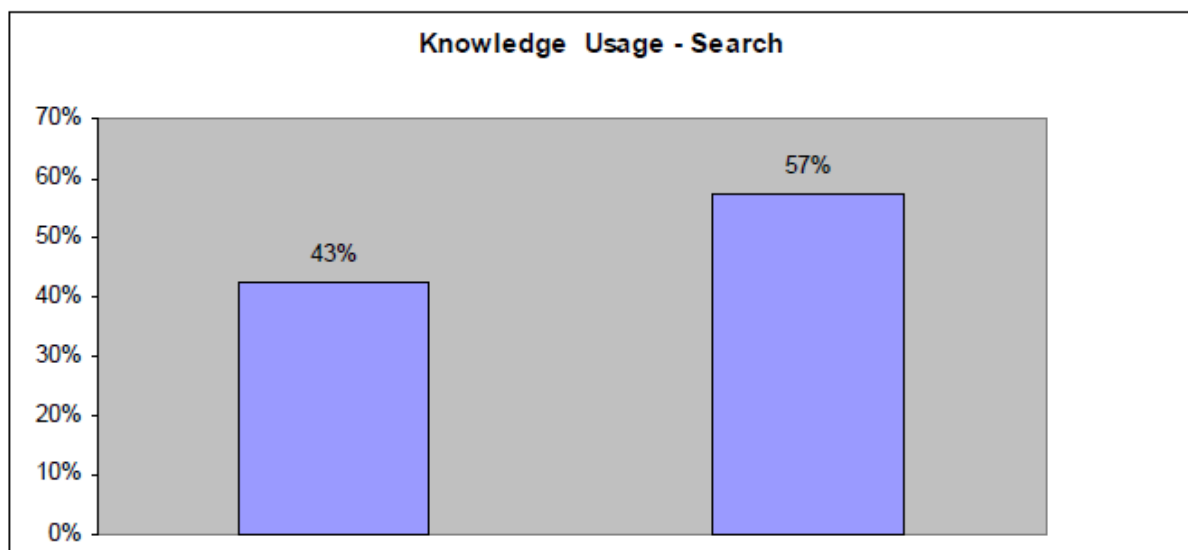


Figure 7.8: Goal measurement for Knowledge Usage - Search

The respondents were satisfied with the quantity and quality of search results. As far as quantity of search results is concerned, 91% (Q6) of the respondents found the number of results optimal. For 73% of the respondents the list of result did not contain any irrelevant result. As regards the quality of search results, 62% (Q3) of the respondents confirmed that the search results satisfy their information needs more than average.

Although 82% (Q4) of them found useful the possibility to search through heterogeneous knowledge sources, only 54% (Q1) of the respondents found that the KnowBench system provides adequate “search” mechanisms. The main problem was that only 9% of the respondents were able to find what they searched for. This could be explained as follows:

Query conceptualisation: It is known from the information retrieval that a user starts his/her search with a short query and tries to exploit the repository in several subsequent refinement steps. As the KnowBench system does not provide refinement support for all types of search, only the 40% is satisfied with the refinement of search. Refinement of keyword-search results is based on the collocation search terms. For the local structured search, refinement is based on focusing on entities in the class hierarchy that could be specialized/relaxed. Adding refinement support on the top of the P2P functionality is a challenging task and is a topic for a future work.

Limitation of the sources to be searched in: Due to short trial period and the fact that knowledge development is a time-consuming activity, the sources to be searched did not contain enough knowledge. Some of respondents explicitly wrote that they spent more time with security issues (that are out of the scope of the KnowBench project, but needed for P2P) than with the knowledge part.

P2P problems: In the presence of network failures, the P2P search is not able to find shared knowledge. Due to limitation of the underlying P2P component, a user is not aware of that. Some self correcting mechanisms of the P2P component were turned off, as in the case when there is no network problem, the same entity will be displayed several times to the user, which was not acceptable for the users.

The search in KnowBench should be faster as only 36% (Q8) of the respondents consider the time needed to find results acceptable. The respondents did not complain about keyword search or structural search. However, they found that the P2P search should be faster as it blocked the machine for a while probably due to high traffic on the network. Thus, the recommendation would be to improve the P2P response time.

Ranking of query results met expectations of 46% (Q10) of the respondents. The current version of the KnowBench system provides ranking support for the keyword-based search and semantic search. Ranking of keyword-search results is based on the frequency of

appearance of a search term. For semantic search, distance between corresponding ontology entities is considered: the shortest path, the more relevant search result. There is not ranking support either for local or for P2P structural search as all knowledge/information is equally important. This can be extended in the future by combining the frequency of usage (either local or distributed) of search results and profile of users who used these results.

According to the answers given by the trial users to the search-related questions of the questionnaire for the summative evaluation, some additional work needs to be done in order to strengthen the search functionality:

Similar to the sharing policies, structured search needs to be simplified and optimised in order to enable a more flexible search. It should be able to search shared data more easily than "property equals string exactly". E.g. using wildcards or similarity based search would be nice.

8.5.6 Knowledge Preservation in KnowBench

Knowledge preservation – Questions					
Q1	Q2	Q3	Q4	Q5	Q6
54%	73%	75%	58%	88%	55%

Table 7.6: Percentage of answers above threshold

The lifecycle of the knowledge items i.e. creation, update, deletion in the KnowBench system seems to be supported well (Figure 8.9). It meets the expectation of 67% of the respondents. Modification of knowledge is not a time consuming function for 75% (Q3) of the respondents and can be done very easily by 73% (Q2) of the developers. Even though the same GUI is used for knowledge modification and knowledge deletion, the respondents found that knowledge deletion can be done faster than knowledge modification but the easiness is decreased.

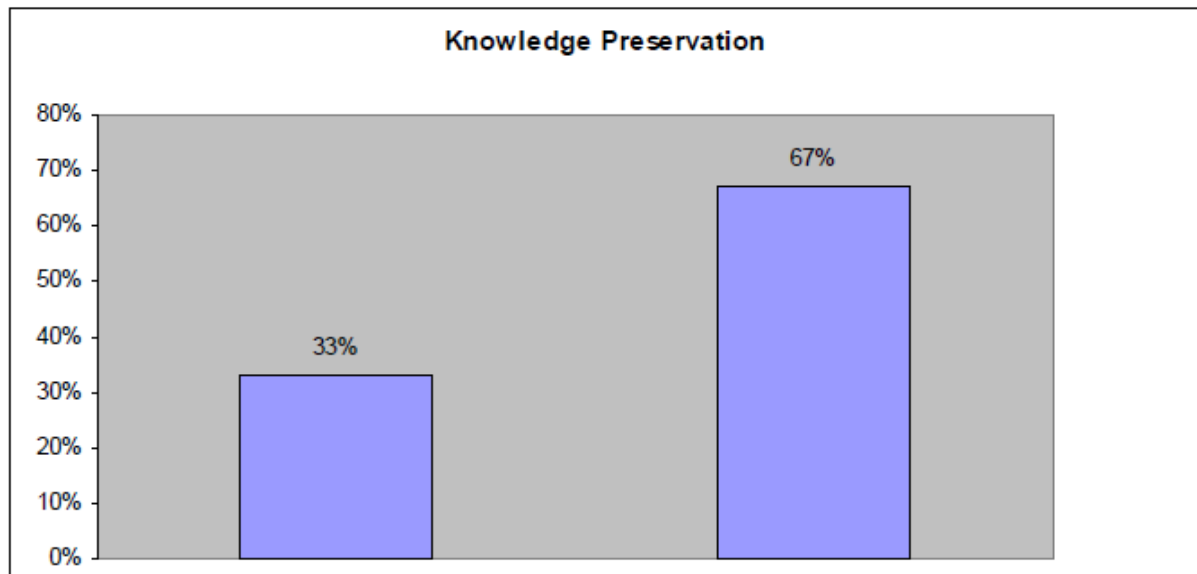


Figure 7.9: Goal measurement for Knowledge Preservation

On the other hand, the usually suspicious function of deletion behaves consistently for only 55% of respondents. However, all respondents who gave the negative answer regarding ensuring consistency in KnowBench also commented that the delete operation is not provided by the system. As this function is a part of the KnowBench system since its 1st release, it seems that the respondents did not find it and consequently gave negative score.

The KnowBench system distinguishes between local (private) knowledge, which is not available to others, and public (distributed) knowledge which is available to others via the P2P network. 54% (Q1) of the respondents found that the knowledge in the P2P network is not always up-to-date. Once published knowledge cannot be revoked / updated then “source knowledge” is deleted/modified. The respondents also noticed delays in sharing of knowledge probably due to high traffic on the network. Sometimes there is some time needed for having P2P knowledge from the other peer published. Also to be sure that the team knowledge is up to date it should be ensured that all the peers are publishing all their knowledge in a timely manner. Thus, some additional work needs to be done in order to fix the P2P problems and strengthen the accuracy of the system.

8.6 System Performance

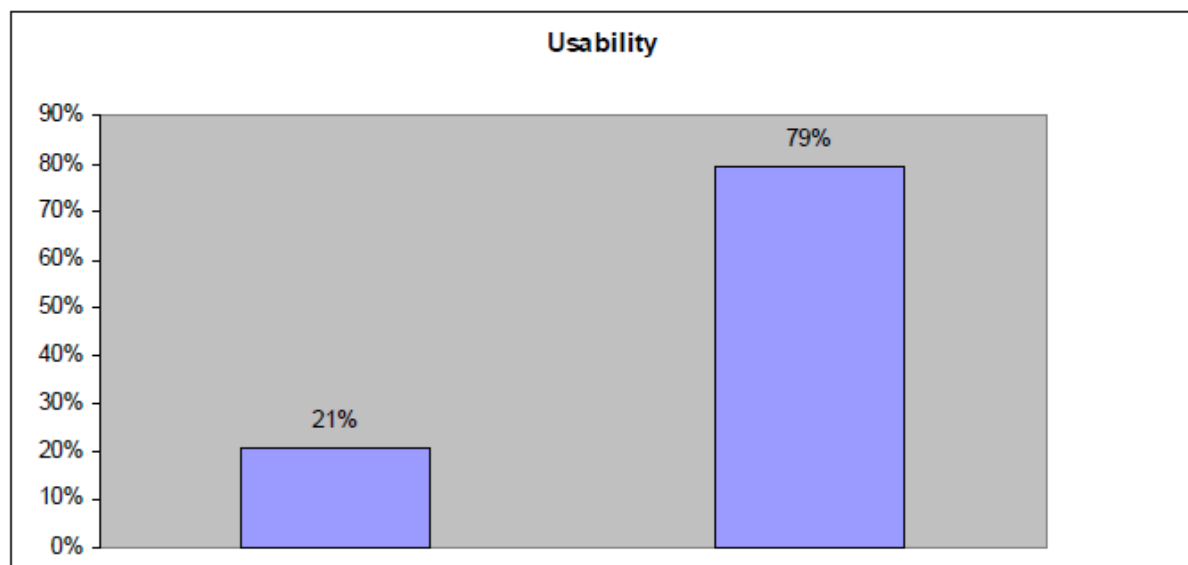
8.6.1 Usability

Usability – Questions

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
81%	67%	80%	93%	69%	87%	85%	73%

Table 7.7: Percentage of answers above threshold

Overall, the usability of the KnowBench system has been well appreciated by the 79% of respondents (Figure 8.10). Almost all respondents said that the KnowBench system provides clear error messages explaining the developers what the problem is and what actions to take to recover from the error.

**Figure 7.10: Goal measurement for Usability**

More than 80% of respondents agree that the user know where he is at all times, how he got there, and how to get back to the point from which he started. They also found that the KnowBench system is simple and that GUI behaves as the users expect. The navigation through the system's functions was found to be very good. Additionally, the form used for knowledge development is sufficiently clear and allows the developers to select data from predefined lists in order to create linkages with existing knowledge items.

The appearance of the system pleased 73% (Q8) of the respondents, but this was somehow expected due the prototype nature of the system.

Although the overall interaction with the KnowBench system is quite comprehensive, some improvements are recommended, since 31% (Q5) of respondents occasionally perceived the interactions as awkward/redundant.

Even though the KnowBench system achieves a good score for its usability, some additional work should be done so that it becomes a user-friendly and intuitive system aiming to improve the productivity of the software developers. Indeed, the 67% (Q2) of respondents found that easiness and intuitiveness of the use of the KnowBench GUI could be further improved.

8.6.2 Reliability

Reliability - Questions

Q1
67%

Table 7.8: Percentage of answers above threshold

The KnowBench system meets the expectations of the users as regards its reliability. More specifically, only 33% of respondents encountered minor problems during the trial use of the KnowBench system (Figure 8.11).

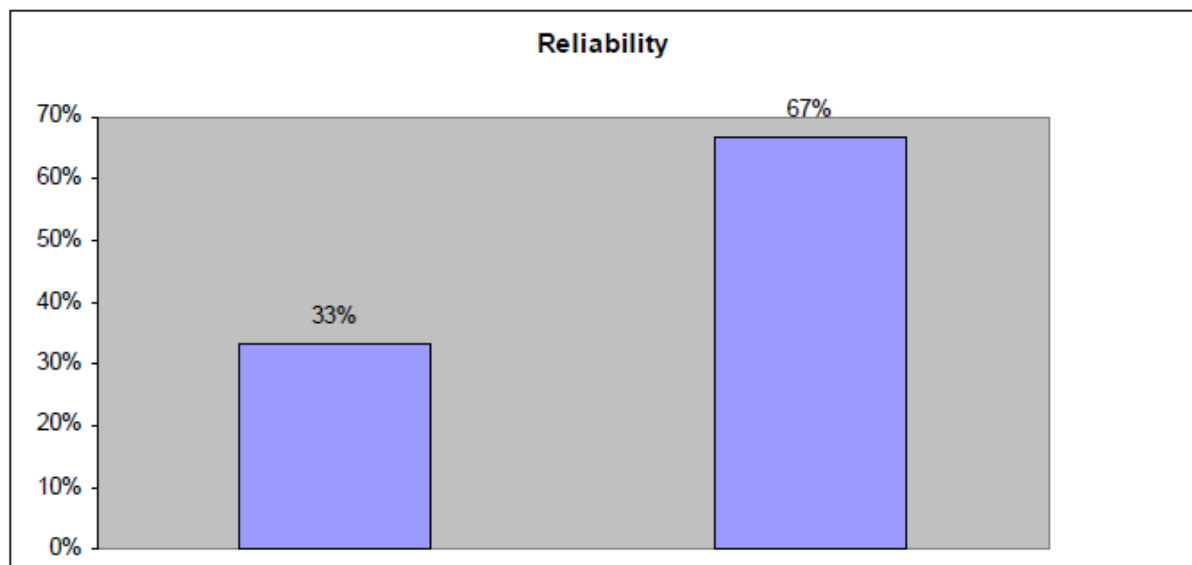


Figure 7.11: Goal measurement for Reliability

8.6.3 Scalability & Availability

Scalability and availability – Questions

Q1	Q2	Q3	Q4	Q5
92%	40%	29%	87%	67%

Table 7.9: Percentage of answers above threshold

The scalability & availability of the KnowBench system meets the expectation of 65% of the respondents (Figure 8.12). The respondents agree that the KnowBench system behaves normally when the size of knowledge base increased.

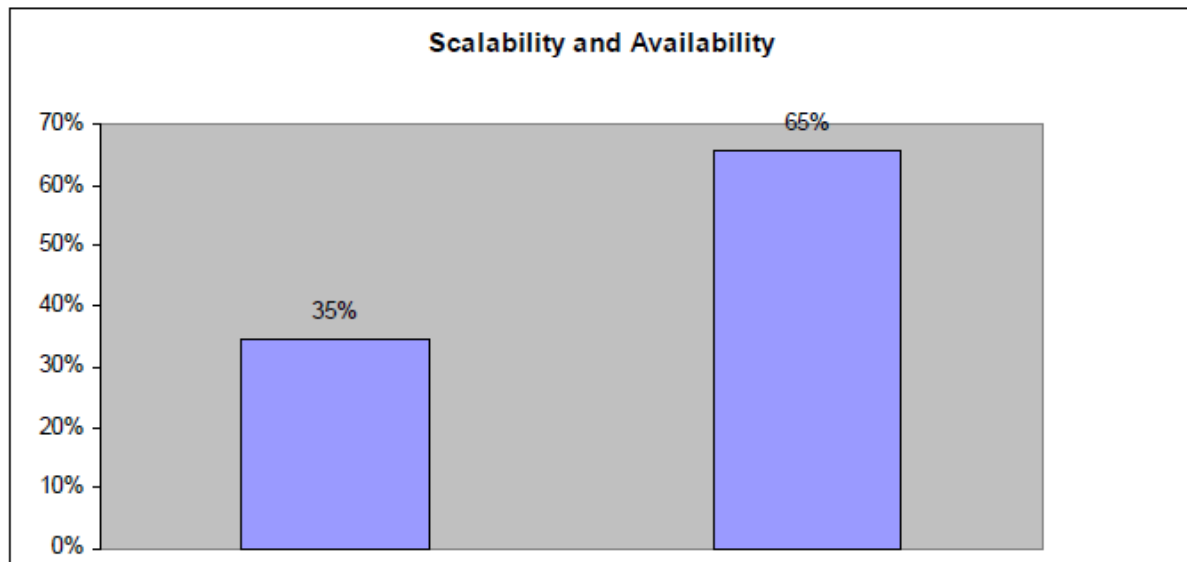


Figure 7.12: Goal measurement for Scalability and Availability

As far as the system's response time, 40% (Q2) of the respondents found it quite slow; especially the P2P network should be faster. Thus, some improvements in the system's response time are needed in order for the KnowBench system to become more agile. 29% (Q3) of the respondents answered that the KnowBench system requires a lot of memory. The respondents also mentioned the need for further optimisation of the system.

On the other hand, the KnowBench system does not occupy a significant size of hard disk space. It also does not introduce an increase raise in network traffic. Thus, no suggestion for improvement is needed.

8.6.4 Security

Security – Questions

Q1	Q2
88%	80%

Table 7.10: Percentage of answers above threshold

The respondents agree that the KnowBench system provides high level of security (Figure 8.13) through:

- restricted access to shared information to users with appropriate rights;
- provision of adequate protection of shared knowledge (e.g. encryption)

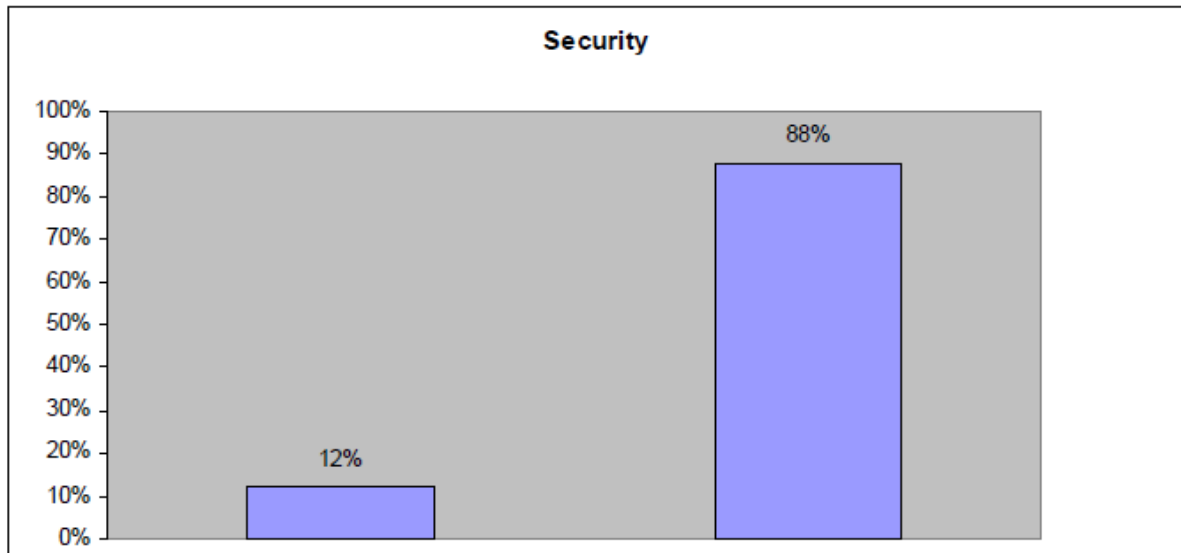


Figure 7.13: Goal measurement for Security

8.6.5 Deployment

Deployment – Questions

Q1	Q2	Q3
81%	86%	73%

Table 7.11: Percentage of answers above threshold

The deployment of the KnowBench system meets the expectation of 80% of the respondents (Figure 8.14). The respondents found that the installation procedure is easy to follow. They also agreed on the friendliness and ease of use of the uninstall procedure. Repeat setup operation in the case that an error occurred could be further simplified in order to satisfy more than 73% of respondents.

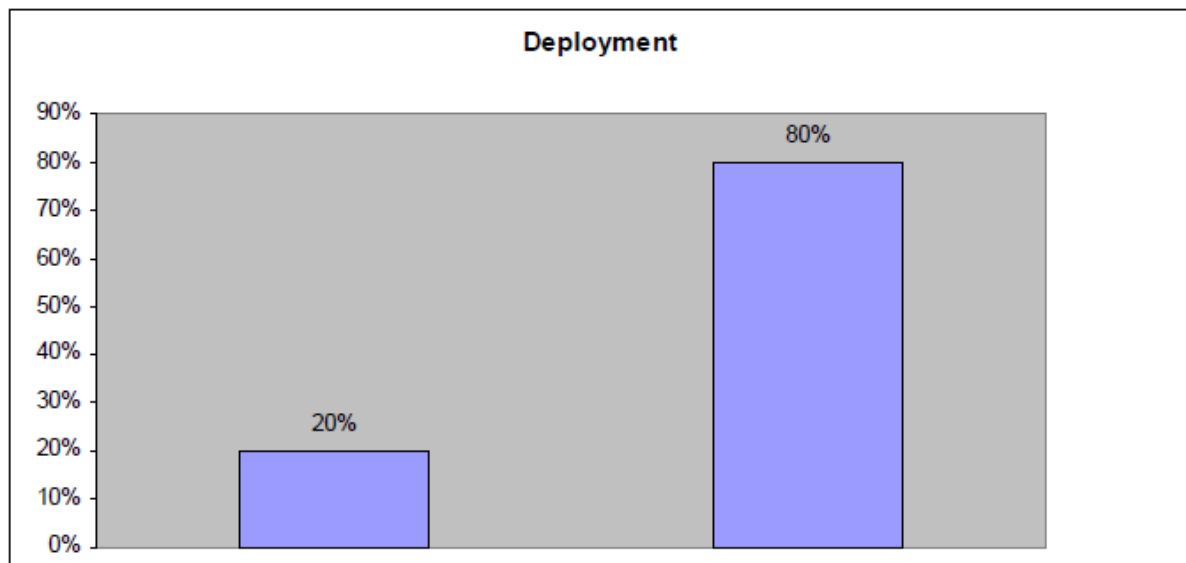


Figure 7.14: Goal measurement for Deployment

8.7 Evaluation Results Synopsis

KnowBench has been well appreciated by the 62% of respondents. All respondents found that its functionality seem to integrate well with Eclipse without overloading developers with information or encumbering them with time-consuming functions. 77% of respondents agreed that KnowBench is capable of improving their overall working experience. They found that the KnowBench concept is good. However at its current prototypical implementation phase, its potential could not be fully exploited. Figure 8.15 summarizes the percentages of positive and negative responses of the developers concerning the knowledge management lifecycle support. Likewise Figure 8.16 summarizes the responses concerning the various aspects of the FURPS evaluation.

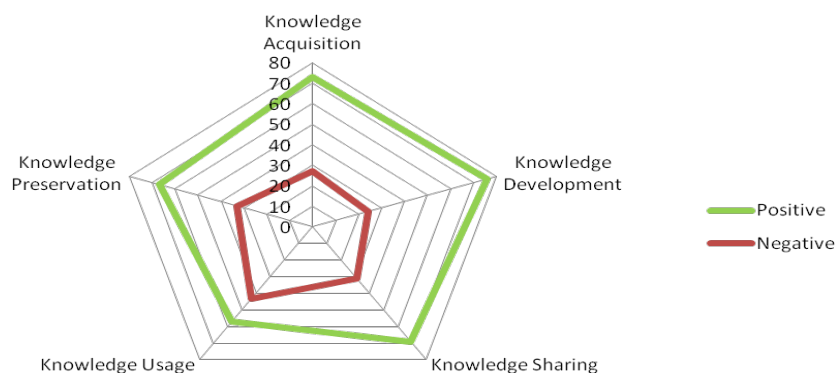


Figure 7.15: Evaluation results synopsis – knowledge management lifecycle support

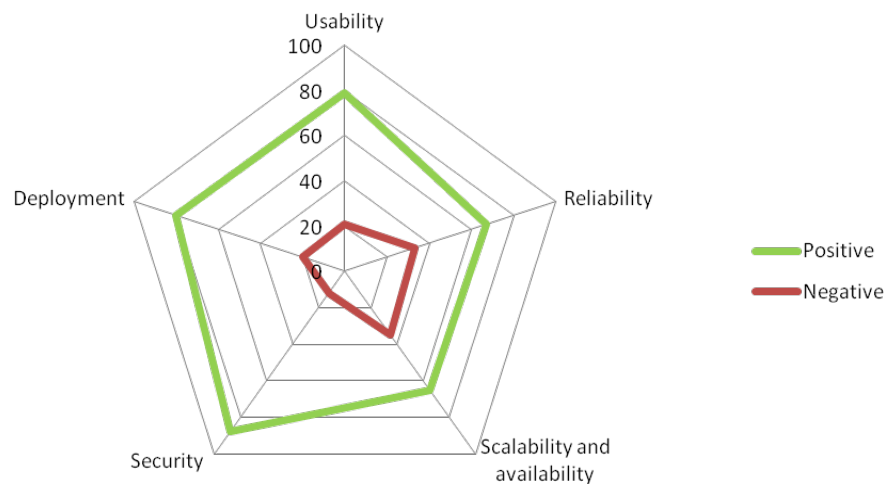


Figure 7.16: Evaluation results synopsis – FURPS

9 CONCLUSIONS AND FUTURE WORK

This doctoral thesis focuses on the domain of semantic-based knowledge management in software development and proposes an approach that is based on the knowledge management lifecycle which is synthesized by the following building blocks: identification, acquisition, development, distribution, preservation, and use of knowledge. The main research goal of the thesis is the development and application of an ***innovative approach for managing knowledge in software development***, as well as the design, development and evaluation of ***a knowledge management system that aids software developers and is powered by social semantic desktop technologies***.

The thesis proposes an approach and a method to apply semantic-based knowledge management in software development as well as a respective system and its evaluation in real case scenarios during software development in software companies. The evaluation is subjective, in the sense that it is performed by software developers (end-users).

At the heart of the proposed framework lies a holistic knowledge management approach for software development based on all the building blocks of the knowledge management lifecycle.

The main research objectives of the proposed approach and system are directly connected to the challenges and limitations of the current state of practice while developing software and the way knowledge is treated and managed.

The thesis addresses the need for knowledge management in software development in distributed environments in order to foster knowledge sharing, supporting in this way advanced capabilities in the distributed engineering and management of software systems.

Contributions

The thesis contributes in current research by providing an approach which strives to resolve problems during ***managing knowledge in software development*** (supporting developers in the ***whole knowledge management lifecycle***) by exploiting ***social semantic desktop*** technologies.

Furthermore, the approach provides:

- ***explicit and tacit knowledge capturing*** - automatic interlinking is supported in order to fasten tacit knowledge capturing.
- ***individual as well as collective knowledge articulation*** - this is achieved by using semantic annotations (in manual and semi-automatic ways), as well as by allowing developers to articulate knowledge inside various metadata-based editing environments.
- ***multifaceted knowledge representation based on semantic visualization of data*** - the thesis' approach suggests multifaceted knowledge representation in order to facilitate several views of the same data, thus achieving knowledge usage in an easier fashion.

Another contribution of the thesis is the design and development of a semantic-based knowledge management system (KnowBench – presented in the following chapters 5, 6 and 7). KnowBench is developed as a proof of concept of the thesis' approach. It is constituted by the following parts:

- Manual semantic annotation
- Semi-automatic semantic annotation
- Knowledge base editing
- P2P services
- Semantic search
- Software Development semantic Wiki (DevWiki)
- Knowledge base graph-based browsing

With the KnowBench system integrated into an IDE, it is easier for developers to create new knowledge. In this way, the repository is not closed. It is always evolving.

The KnowBench system, is actively integrated into the work process. This fact enables capturing information/knowledge during the software development process without developers' extra effort.

9.1 Limitations and Possible Improvements

The thesis' approach and the respective KnowBench system support only the software development process instead of the whole software engineering lifecycle (e.g. requirements engineering and design of software as well as testing of software systems are not supported).

As far as the KnowBench system is concerned the following issues constitute its limitations.

One issue that is not trivial to solve is that the knowledge in the P2P network is not always up-to-date. Once published, knowledge cannot be revoked /updated whenever "source knowledge" is deleted/ modified.

The semantic annotation of source code provides granularity which is restricted by the underlying Eclipse platform itself as the IJavaElement interface is exploited to map between source code fragments and metadata. This limits the selected source code fragments to be annotated in the nearest Java elements that surround the source code fragment at hand (e.g. package/class/method/variable etc.).

The evaluation of KnowBench assisted in determining what improvements may further optimize it. Evaluators' responses provided valuable feedback. Some examples of improvements that would be nice to have are: (a) some parts of the ontologies could be more detailed and more structured, (b) the wiki language could be easier to understand and learn, (c) it would be nice to keep a history of the changes of each knowledge item to see who changed what, (d) visualisation of all knowledge items shared by a specific user (the owner or a colleague), (e) the P2P search could be faster, (f) include personal tags to some of the knowledge items, (g) faster response time with respect to searching/visualisation of shared knowledge items, (h) improvement of the install and uninstall procedure and (i) KnowBench is limited to work with Java 6; some evaluators expressed the need to work with Java 5 as well.

Furthermore, the semi-automatic semantic annotation mechanism could be extended in order to derive more precise annotations by introducing JAPE grammar rules of the GATE system that will be customized for source code.

Finally, KnowBench does not provide an internal ontology editor to alter the underlying ontologies. An external tool has to be used for that purpose.

9.2 Further Research

Some issues that could be of interest for further research are outlined below.

A possible extension could be to introduce a context system. Such a system could observe a software developer's behaviour and proactively determine situations in which he needs knowledge support. Experience could be treated as a context in which a problem is resolved, i.e. the cycle "develop-problem-source-develop-..." that leads to resolving a situation in which a developer requires knowledge support (e.g. a malfunctioning code). This context system could refine captured interactions and sensed events to semantically useful context information. For example, from the interaction events the context system could describe what the developer has been doing in a given period of time (e.g., localizing a bug, refactoring code, implementing authorisation mechanism).

Another interesting future extension could be to introduce semantic recommendations that may enable proactive knowledge delivery, depending on the actual working and personal context of the user. A semantic recommendation system could support resolving granular problems, such as instantiating an object, handling a specific error. The semantic recommendation may potentially make use of the information monitored by the context system mentioned above, e.g. components used in the current class. One recommendation situation occurs when a developer repeats the same error several times. By analyzing common errors developers made in the past, developers who are creating the same artefacts and getting the same error, can be warned how this error in previous situations has been solved. Another situation for proactive support is the time a developer is spending on writing a program block. It is possible that a developer gets stuck during coding and that hints can be useful. Contextual information, e.g. working context and user's preferences, is used to determine implicit knowledge needs. For example, if a user opened a web site while programming, the proactive support takes the content of that document into account.

REFERENCES

- Abadi, D. J., Marcus, A., Madden, S. R., & Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd international conference on Very large data bases* (pp. 411–422). Vienna, Austria.
- Aberer, K., Datta, A., Hauswirth, M., & Schmidt, R. (2005). Indexing data-oriented overlay networks. In *Proceedings of the 31st international conference on Very large data bases* (pp. 685–696). Trondheim.
- Ackerman, M. S., & Malone, T. W. (1990). Answer Garden: A tool for growing organizational memory. In *Proceedings of the ACM SIGOIS and IEEE CS TC-OA conference on Office information systems* (pp. 31–39).
- Ademar Aguiar, Gabriel David, & Manuel Padilha. (2003). XSDoc: an Extensible Wiki-based Infrastructure for Framework Documentation. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.103.479>
- Ag, H. (2008). IS/6 Administrator's Guide Hyperwave IS/6 Release 3 Hyperwave Guides IS/6 Administrator's Guide.
- Aguiar, A., & David, G. (2005). WikiWiki weaving heterogeneous software artifacts. In *Proceedings of the 2005 international symposium on Wikis - WikiSym '05* (pp. 67–74). Presented at the the 2005 international symposium, San Diego, California. doi:10.1145/1104973.1104980
- Amir, M. (2001). Code web: data mining library reuse patterns. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE '01)* (pp. 827–828). Washington, DC, USA: IEEE Computer Society.

- Ankolekar, A., Sycara, K., Herbsleb, J., Kraut, R., & Welty, C. (2006). Supporting online problem-solving communities with the semantic web. In *Proceedings of the 15th international conference on World Wide Web* (pp. 575–584).
- Antoniol, G., Canfora, G., Casazza, G., & De Lucia, A. (2000). Information retrieval models for recovering traceability links between code and documentation. In *Proceedings of the International Conference on Software Maintenance* (pp. 40–49). San Jose, CA.
- Arent, J., & Norbjerg, J. (2000). Software process improvement as organizational knowledge creation: a multiple case analysis. In *Proceedings of the Hawaii International Conference on System Sciences* (p. 105). Maui, USA.
- Arent, J., Norbjerg, J., & Pedersen, M. H. (2000). Creating organizational knowledge in software process improvement. In *Proceedings of the 2nd Workshop on Learning Software Organizations* (pp. 81–92). Oulu, Finland.
- Athanasiadis, I. N., Villa, F., & Rizzoli, A. E. (2007). Enabling knowledge-based software engineering through semantic-object-relational mappings. In *Proceedings of the 3rd International Workshop on Semantic Web Enabled Software Engineering*. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.2747>
- Auer, S., Dietzold, S., & Riechert, T. (2006). OntoWiki - A Tool for Social, Semantic Collaboration. In *The 5th International Semantic Web Conference (ISWC 2006)* (Vol. 4273, pp. 736-749). Springer. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.6346>
- Aumueller, D., & Auer, S. (2005). Towards a Semantic Wiki Experience - Desktop Integration and Interactivity in WikSAR. Retrieved November 29, 2010, from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.7454>

- Bachmann, F., & Merson, P. (1998, October 27). Experience Using the Web-Based Tool Wiki for Architecture Documentation. Retrieved from <http://handle.dtic.mil/100.2/ADA446186>
- Barros, M. O., Werner, C. M., & Travassos, G. H. (2004). Supporting risks in software project management. *Journal of Systems and Software*, 70(1-2), 21–35.
- Basili, V., Costa, P., Lindvall, M., Mendonca, M., Seaman, C., Tesoriero, R., & Zelkowitz, M. (2002). An experience management system for a software engineering research organization. In *Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard* (pp. 29–35).
- Basili, V. R., Caldiera, G., & Rombach, D. H. (1994). The Experience Factory. *Encyclopedia of Software Engineering*, John Wiley & Sons, Inc, 469–476.
- Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). The goal question metric approach. *Encyclopedia of software engineering*, John Wiley & Sons, Inc., 1, 528–532.
- Basili, V. R., Lindvall, M., & Costa, P. (2001). Implementing the Experience Factory concepts as a set of Experience Bases. In *13th International Conference on Software Engineering & Knowledge Engineering*, Knowledge Systems Institute (pp. 102–109).
- Basili, V. R., & Rombach, H. D. (2002). Support for comprehensive reuse. *IEEE Software Engineering Journal*, 6(5), 303–316.
- Baskerville, R., & Pries-Heje, J. (1999). Knowledge capability and maturity in software management. *ACM SIGMIS Database*, 30(2), 26–43.
- Bauer, B., & Roser, S. (2006). Semantic-enabled Software Engineering and Development. In *Proceedings of the 1st International Workshop on Applications of Semantic Technologies* (pp. 293-296). LNI.

- Bennis, W., & Ward Biederman, P. (1998). None of us is as smart as all of us. *IEEE Computer*, 31(3), 116-117. doi:10.1109/2.660195
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web: Scientific American. *Scientific American*. Retrieved from <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&pageNumber=1&catID=2>
- Biezunski, M., & Bryan, M. (1999). *SRN: ISO/IEC 13250: 2000 Topic Maps*. Technical report.
- Birk, A., Surmann, D., & Althoff, K. D. (1999). Applications of knowledge acquisition in experimental software engineering. *Knowledge Acquisition, Modeling and Management*, 67–84.
- Bisson, G., Nédellec, C., & Canamero, L. (2000). Designing clustering methods for ontology building-The Mo'K workbench. In *Proceedings of the ECAI Ontology Learning Workshop* (pp. 13–19).
- Bjornson, F., & Staalhane, T. (2005). Harvesting knowledge through a method framework in an electronic process guide. In *Proceedings of the Seventh International Workshop on Learning Software Organizations* (pp. 86–90). Kaiserslautern, Germany: Springer Verlag.
- Bjornson, F. O., & Dingsoyr, T. (2008). Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, 50(11), 1055–1068.
- Borges, L. M. S., & Falbo, R. A. (2002). Managing Software Process Knowledge. In *Proceedings of CSITeA'2002*.

- Brössler, P. (1999). Knowledge management at a software engineering company—an experience report. In *Proceedings Workshop Learning Software Organizations* (pp. 163–170). Kaiserslautern, Germany.
- Buffa, M. (2006). Intranet wikis. *Proceedings IntraWeb Workshop WWW2006*.
- Buffa, M., & Gandon, F. (2006). SweetWiki. In *Proceedings of the 2006 international symposium on Wikis - WikiSym '06* (p. 69). Presented at the the 2006 international symposium, Odense, Denmark. doi:10.1145/1149453.1149469
- Buitelaar, P., Olejnik, D., & Sintek, M. (2003). OntoLT: A protégé plug-in for ontology extraction from text. In *Proceedings of the International Semantic Web Conference (ISWC)*.
- Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). *Readings in information visualization: using vision to think*. Morgan Kaufmann.
- Chau, T., & Maurer, F. (2006). A Case Study of Wiki-based Experience Repository at a Medium-sized Software Company. *Proceedings Kumar Jain, R. & Prabhakar, R. (Eds.): Wiki - A new wave in web collaboration, The ICFAI University Press*, 60-79.
- Chau, T., & Maurer, F. (2005). A case study of wiki-based experience repository at a medium-sized software company. In *Proceedings of the 3rd international conference on Knowledge capture - K-CAP '05* (p. 185). Presented at the the 3rd international conference, Banff, Alberta, Canada. doi:10.1145/1088622.1088660
- Chewar, C. M., & McCrickard, D. S. (2005). Links for a human-centered science of design: Integrated design knowledge environments for a software development process. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (p. 256c).

- Cheyen, A., Park, J., & Giuli, R. (2005). IRIS: Integrate. Relate. Infer. Share. *1st Workshop on The Semantic Desktop. 4th International Semantic Web Conference*. Retrieved from http://www.ai.sri.com/pub_list/1207
- Cimiano, P. (2006). *Ontology learning and population from text: algorithms, evaluation and applications*. Springer Verlag.
- Cimiano, P., Pivk, A., Schmidt-Thieme, L., & Staab, S. (2004). Learning taxonomic relations from heterogeneous sources. *Proceedings of the ECAI 2004 Ontology Learning and Population Workshop*.
- Cimiano, P., & Völker, J. (2005). Text2Onto. In *Natural Language Processing and Information Systems*, Lecture Notes in Computer Science (Vol. 3513, pp. 227-238). Springer Berlin / Heidelberg. Retrieved from http://dx.doi.org/10.1007/11428817_21
- Collard, M. L., Maletic, J. I., & Marcus, A. (2002). Supporting document and data views of source code. In *Proceedings of the 2002 ACM symposium on Document engineering* (pp. 34–41).
- Cook, C., & Churcher, N. (2006). Constructing real-time collaborative software engineering tools using CAISE, an architecture for supporting tool development. In *Proceedings of the 29th Australasian Computer Science Conference-Volume 48* (pp. 267–276).
- Corby, O., & Dieng, R. (2004). Querying the semantic web with the Corese search engine. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'2004), subconference PAIS* (pp. 705–709).
- Čubranić, D., & Murphy, G. C. (2003). Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering* (pp. 408–418).

- Cunningham, D. H., Maynard, D. D., Bontcheva, D. K., & Tablan, M. V. (2002). GATE: A framework and graphical development environment for robust NLP tools and applications. *Proceedings of the 40th Annual Meeting of the ACL*.
- Cunningham, W., & Leuf, B. (2001). *The wiki way: Collaboration and sharing on the Internet*. Reading, MA: Addison-Wesley.
- Czuchry Jr, A. J., & Harris, D. R. (2002). KBRA: A new paradigm for requirements engineering. *IEEE Expert*, 3(4), 21–24.
- Davenport, T. H., & Prusak, L. (2000). *Working knowledge: How organizations manage what they know*. Harvard Business Press.
- Davis, M. (2006). Semantic Wave 2006-Part 1: Executive Guide to Billion Dollar Markets. *Project10X Special Report, Washington*.
- Decker, B., Ras, E., Rech, J., Klein, B., & Hoecht, C. (2005). Self-organized reuse of software engineering knowledge supported by semantic wikis. In *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE)*.
- Decker, B., Ras, E., Rech, J., Klein, B., Reuschling, C., Höcht, C., Kilian, L., et al. (2005). A framework for agile reuse in software engineering using Wiki Technology. In *KMDAP Workshop*.
- Decker, S., & Frank, M. R. (2004). The networked semantic desktop. In *Proceedings of WWW2004 Workshop Application Design, Development and Implementation Issues in the Semantic Web*.
- Decker, S. (2006). The social semantic desktop: Next generation collaboration infrastructure. *Information Services and Use*, 26(2), 139-144.
- Dello, K., Simperl, E. P., & Tolksdorf, R. (2006). Creating and using Semantic Web

- information with Makna. In *First Workshop on Semantic Wikis* (p. 1).
- Demarest, M. (1997). Understanding knowledge management. *Long Range Planning*, 30(3), 374–384.
- Désilets, A., Paquet, S., & Vinson, N. G. (2005). Are wikis usable? In *The 2005 International Symposium on Wikis* (pp. 3-15). New York, NY: ACM Press.
- Dingsoyr, T., & Conradi, R. (2002). A survey of case studies of the use of knowledge management in software engineering. *International Journal of Software Engineering and Knowledge Engineering*, 12(4), 391–414.
- Dingsoyr, T., Djarraya, H. K., & Royrvik, E. (2005). Practical knowledge management tool use in a software consulting company. *Communications of the ACM*, 48(12), 96–100.
- Dingsoyr, T., & Moe, N. B. (2008). The impact of employee participation on the use of an electronic process guide: a longitudinal case study. *Software Engineering, IEEE Transactions on*, 34(2), 212–225.
- Dingsoyr, T., & Royrvik, E. (2003). An empirical study of an informal knowledge repository in a medium-sized software consulting company. In *Proceedings of the 25th International Conference on Software Engineering* (pp. 84–92).
- Doran, H. D. (2004). Agile knowledge management in practice. In *Proceedings of the Sixth International Workshop on Learning Software Organizations* (pp. 137–143). Banff, Canada: Springer Verlag.
- Draganidis, F., & Mentzas, G. (2006). Competency based management: a review of systems and approaches. *Information Management & Computer Security*, 14(1), 51–64.
- Dutoit, A., & Paech, B. (2003). Eliciting and maintaining knowledge for requirements evolution.

- Dybå, T. (2001). *Enabling Software Process Improvement: An Investigation on the Importance of Organizational Issues* (PhD thesis). Norwegian University of Science and Technology, Department of Computer and Information Science.
- Earl, M. (2001). Knowledge management strategies: towards a taxonomy. *Journal of Management Information Systems*, 215–233.
- Eberhart, A., & Argawal, S. (2004). SmartAPI-Associating Ontologies and APIs for RAD. In *Proceedings of Modellierung*.
- Edwards, J. S. (2003). Managing software engineers and their knowledge. In *Proceedings A. Aurum et al. (Eds.), Managing Software Engineering Knowledge* (pp. 5–27). Springer Verlag.
- van den Elst, J., van Harmelen, F., & Thonnat, M. (1995). Modelling software components for reuse. In *Proceedings of the Seventh International Conference on Software Engineering and Knowledge Engineering* (pp. 350–357). Knowledge Systems Institute.
- Erol Bozsak, Marc Ehrig, Siegfried H, Andreas Hotho, Er Maedche, Boris Motik, Daniel Oberle, et al. (2002). Kaon - towards a large scale semantic web. Springer. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.2.492>
- Fehér, P., & Gábor, A. (2006). The role of knowledge management supporters in software development companies. *Software Process: Improvement and Practice*, 11(3), 251–260.
- Feldmann, R., & Althoff, K. (2001). On the status of learning software organisations in the year 2001. In *Proceedings of the Learning Software Organizations Workshop* (pp. 2–6). Kaiserslautern, Germany: Springer Verlag.

- Fellbaum, C. (1998). *WordNet: An electronic lexical database*. The MIT press.
- Fensel, D. (2001). *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Berlin: Springer-Verlag.
- Fischer, J., Gantner, Z., Rendle, S., Stritt, M., & Schmidt-Thieme, L. (2006). Ideas and improvements for semantic wikis. *The Semantic Web: Research and Applications*, 650–663.
- Folkestad, H., Pilskog, E., & Tessem, B. (2004). Effects of software process in organization development—a case study. In *Proceedings of the Sixth International Workshop on Learning Software Organizations* (pp. 153–164). Banff, Canada: Springer Verlag.
- Fouque, G., & Matwin, S. (1992). CAESAR: a system for case based software reuse. In *Knowledge-Based Software Engineering Conference, 1992* (pp. 90–99).
- Frakes, W. B., & Kang, K. (2005). Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, 31(7), 529–536.
- Frechtling, J. (2002). The 2002 User-Friendly Handbook for Project Evaluation. *National Science Foundation*. Retrieved from <http://www.nsf.gov/pubs/2002/nsf02057/nsf02057.pdf>
- Froehlich, J., & Dourish, P. (2004). Unifying artifacts and activities in a visual tool for distributed software development teams. In *Proceedings of the 26th International Conference on Software Engineering* (pp. 387–396).
- Giesbrecht, E., Stojanovic, L., & Tran, T. (2008). *D29: Second iteration prototype of Semantic Search*. TEAM EU Project.
- Gomez-Perez, A., Fernández-López, M., & Corcho, O. (2003). Ontological engineering. *AI Magazine, Springer*, 36, 56.

- Grabher, G., & Ibert, O. (2006). Bad company? The ambiguity of personal knowledge networks. *Journal of Economic Geography*, 6(3), 251.
- Groza, T., Handschuh, S., Moeller, K., Grimnes, G., Sauermann, L., Minack, E., Mesnage, C., et al. (2007). The nepomuk project-on the way to the social semantic desktop. In *Proceedings of I-Semantics* (Vol. 7, pp. 201–211).
- Handschuh, S., & Staab, S. (2003). CREAM: CREAting metadata for the Semantic Web. *Computer Networks*, 42(5), 579–598.
- Hanebutte, N., & Oman, P. W. (2005). Software vulnerability mitigation as a proper subset of software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(6), 379–400.
- Happel, H. J., Kögel, M., Maalej, W., Narendula, R., Panagiotou, D., Schmidt, R., & Wolf, T. (2007). D5: Report describing State-of-the-Art in Metadata Management.
- Happel, H. J., Korthaus, A., Seedorf, S., & Tomczyk, P. (2006). Kontor: An ontology-enabled approach to software reuse. In *Proceedings of the 18th International Conference on Software Engineering and Knowledge Engineering*.
- Happel, H. J., & Seedorf, S. (2006). Applications of ontologies in software engineering. In *Proceedings of Workshop on Semantic Web Enabled Software Engineering"(SWESE) on the ISWC* (pp. 5–9).
- Harzallah, M., Berio, G., & Vernadat, F. (2005). Analysis and modeling of individual competencies: toward better management of human resources. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 36(1), 187–207.
- Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2* (pp. 539–

545).

- Henninger, S. (1991). Retrieving software objects in an example-based programming environment. In *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 251–260).
- Henninger, S. (1997). Case-Based Knowledge Management Tools for Software Development. *Journal of Automated Software Engineering*, 4, 319--340.
- Hill, R., & Rideout, J. (2004). Automatic method completion. In *Proceedings of the 19th International Conference on Automated Software Engineering* (pp. 228–235).
- Holmes, R., & Murphy, G. C. (2005). Using structural context to recommend source code examples. In *Proceedings of the 27th international conference on Software engineering* (pp. 117–125).
- Holz, H. (2003a). *Process Based Knowledge Management Support for Software Engineering*. dissertation. de-Verl. im Internet GmbH.
- Holz, H. (2003b). *Process Based Knowledge Management Support for Software Engineering*. University of Kaiserslautern, dissertation. de-Verl. im Internet GmbH.
- Humphrey, W. (2005). PSP: a self-improvement process for software engineers.
- Humphrey, W. S. (1999). Pathways to process maturity: The personal software process and team software process. *SEI Interactive*, 2(2), 1–17.
- Humphrey, W. S. (2000). *Introduction to the team software process*. Addison-Wesley Professional.
- Inoue, K., Yokomori, R., Fujiwara, H., Yamamoto, T., Matsushita, M., & Kusumoto, S. (2003). Component rank: relative significance rank for software component search. In *Proceedings of the 25th International Conference on Software Engineering* (pp.

14–24).

John, M., Jugel, M., Schmidt, S., & Wloka, J. (2005). Wikis in der Softwareentwicklung helfen. *Java Magazin*, 7, 88–91.

Kankanhalli, A., Tan, B. C., & Wei, K. K. (2005). Contributing knowledge to electronic knowledge repositories: An empirical investigation. In *Mis Quarterly* (pp. 113–143).

Kant, E. (1992). Knowledge-based support for scientific programming. In *Knowledge-Based Software Engineering Conference, 1992* (pp. 2–4).

Kapor, M. (2005). What's Compelling About Chandler: A Current Perspective.

Karger, D. R., Bakshi, K., Huynh, D., Quan, D., & Sinha, V. (2005). Haystack: A customizable general-purpose information management tool for end users of semistructured data. In *Proc. of the CIDR Conf.*

Klein, B., Hoecht, C., & Decker, B. (2005). Beyond Capturing and Maintaining Software Engineering Knowledge- " Wikitology" as Shared Semantics. In *Workshop on Knowledge Engineering and Software Engineering, at conference of Artificial Intelligence*.

Knublauch, H., Oberle, D., Tetlow, P., & Wallace, E. (1999). A semantic web primer for object-oriented software developers. In *Workshop on Agent Technologies and Their Application Scenarios in Logistics* (Vol. 2000).

Kotelnikov, M., Polonsky, A., Kiesel, M., Völkel, M., Haller, H., Sogrin, M., Lannerö, P., et al. (2007). Interactive Semantic Wikis. *NEPOMUK Deliverable D1.1, 1*.

Krötzsch, M., Vrandečić, D., & Völkel, M. (2006). Semantic mediawiki. *The Semantic Web-ISWC 2006*, 935–942.

Kurniawati, F., & Jeffery, R. (2004). The long-term effects of an EPG/ER in a small

- software organisation. In *Proceedings of the Australian Software Engineering Conference* (pp. 128–136).
- Leitao, A. M. (2004). Detection of redundant code using R 2 D 2. *Software Quality Journal*, 12(4), 361–382.
- Lindvall, M., & Rus, I. (2000). Process diversity in software development. *Software, IEEE*, 17(4), 14-18. doi:10.1109/MS.2000.854063
- Lindvall, M., & Rus, I. (2002). Knowledge management in software engineering. *IEEE Software*, 19(3), 26–38.
- Lindvall, M., & Rus, I. (2003). Knowledge management for software organizations. In *Proceedings of Managing Software Engineering Knowledge*. New York, USA: Springer.
- Lindvall, M., Rus, I., Jammalamadaka, R., & Thakker, R. (2001). Software tools for knowledge management. *Fraunhofer Center for Experimental Software Engineering, Maryland, USA*.
- Louridas, P. (2006). Using wikis in software development. *Software, IEEE*, 23(2), 88–91.
- Lyytinen, K., & Robey, D. (1999). Learning failure in information systems development. *Information Systems Journal*, 9(2), 85–101.
- Maalej, W., Panagiotou, D., & Happel, H. J. (2008). Towards effective management of software knowledge exploiting the semantic wiki paradigm. In *Software Engineering* (Vol. 121, pp. 183–197). GI.
- Maedche, A., & Staab, S. (2000). Discovering conceptual relations from text. In *ECAI* (pp. 321–325).
- Maedche, A., & Staab, S. (2004). Ontology learning. *Handbook on ontologies*, 173–190.

- Malte, K. (2006). Kaukolu: Hub of the Semantic Corporate Intranet. In *Proceedings of the First Workshop on Semantic Wikis—From Wiki To Semantics*.
- Mathiassen, L., & Vogelsang, L. (2005). The role of networks and networking in bringing software methods to practice. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (p. 256b). Big Island, HI, United States.
- Mccarey, F., Cinnéide, M. Ó., & Kushmerick, N. (2005). Rascal: A recommender agent for agile reuse. *Artificial Intelligence Review*, 24(3), 253–276.
- McGarry, F., Pajerski, R., Page, G., Waligora, S., & Basili, V. (1994). *Software process improvement in the NASA software engineering laboratory*. Citeseer.
- Medynskiy, Y. E., Ducheneaut, N., & Farahat, A. (2006). Using hybrid networks for the analysis of online software development communities. In *Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 513–516).
- Mens, K., Poll, B., & González, S. (2003). Using intentional source-code views to aid software maintenance. In *Proceedings of the International Conference on Software Maintenance* (pp. 169–178).
- Mili, A., Mili, R., & Mittermeir, R. T. (1998). A survey of software reuse libraries. *Annals of Software Engineering*, 5(1), 349–414.
- Mullick, N., Bass, M., Houda, Z., Paulish, P., & Cataldo, M. (2006). Siemens Global Studio Project: Experiences Adopting an Integrated GSD Infrastructure. In *Proceedings of the IEEE international conference on Global Software Engineering* (pp. 203–212).
- Natali, A. C., & Falbo, R. A. (2002). Knowledge Management in Software Engineering Environments. In *Proceedings of the XVI Brazilian Symposium on Software Engineering (SBES'2002)* (pp. 238–253).

- Nelson, T. (1999). ZX, a New User Environment [now Floating World(tm)] Preliminary 1999 Specifications. Retrieved November 30, 2010, from <http://www.xanadu.net/zigzag/fw99/index.html>
- Nguyen, T. N., & Munson, E. V. (2003). The software concordance: a new software document management environment. In *Proceedings of the 21st annual international conference on Documentation* (pp. 198–205). ACM Press.
- Noll, R. P., & Ribeiro, M. B. (2007). Enhancing traceability using ontologies. In *Proceedings of the 2007 ACM symposium on Applied computing* (pp. 1496–1497). Seoul, Korea.
- Norbjerg, J., Elisberg, T., & Pries-Heje, J. (2006). Experiences from using knowledge networks for sustaining Software Process Improvement. In *Proceedings of the Eighth International Workshop on Learning Software Organizations* (pp. 9–17). Rio de Janeiro, Brazil.
- O'Keeffe, M., & Cinnéide, M. Ó. (2006). Search-based software maintenance. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering* (pp. 10–260).
- Oren, E. (2005). SemperWiki: a semantic personal Wiki. In *Proc. of 1st WS on The Semantic Desktop, Galway, Ireland*.
- Oren, E., Breslin, J. G., & Decker, S. (2006). How semantics make better wikis. In *Proceedings of the 15th international conference on World Wide Web* (pp. 1071–1072).
- Pan, Z., & Heflin, J. (2003). DLDB: Extending relational databases to support semantic web queries. In *PSSS*. Citeseer.

- Panagiotou, D., & Mentzas, G. (2007). A comparison of semantic wiki engines. In *22nd European Conf. on Operational Research*.
- Panagiotou, D., & Mentzas, G. (2008). Exploiting Semantics in Collaborative Software Development Tasks. In *Proceeding of the 2008 conference on Knowledge-Based Software Engineering: Eighth Joint Conference on Knowledge-Based Software Engineering* (pp. 385–394). IOS Press.
- Panagiotou, D., & Mentzas, G. (2009a). A Knowledge Workbench for Software Development. In *Proceedings of the International Conference on Semantic Systems*. Graz, Austria.
- Panagiotou, D., & Mentzas, G. (2009b). A Semantic Wiki for Software Development. In *Proceedings of the 13th Panhellenic Conference on Informatics*. Corfu, Greece.
- Panagiotou, D., & Mentzas, G. (2010). KnowBench - A semantic user interface for managing knowledge in software development. In *Proceedings of the 5th International Conference on Software and Data Technologies*. Athens, Greece.
- Panagiotou, D., Paraskevopoulos, F., & Mentzas, G. (2011). Knowledge-based interaction in software development. *Intelligent Decision Technologies*.
- Paulk, M. C. (1995). *The capability maturity model: Guidelines for improving the software process* (Vol. 66). Addison-Wesley Reading, MA.
- Probst, G. J. (1998). Practical knowledge management: A model that works. *PRISM-CAMBRIDGE MASSACHUSETTS*-, 17–30.
- Rauschmayer, A. (2005). An RDF editing platform for software engineering. *ISWC Wsh. Semantic Web Enabled Software Engineering (SWESE)*.
- Richter, J., Völkel, M., & Haller, H. (2005). Deepamehta-a semantic desktop. In

Proceedings of the 1st Workshop on The Semantic Desktop. 4th International Semantic Web Conference (Galway, Ireland) (Vol. 175).

Rosson, M. B., & Carroll, J. M. (1996). The reuse of uses in Smalltalk programming. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(3), 219–253.

Rus, I., Lindvall, M., & Sinha, S. S. (2001). Knowledge Management in Software Engineering: A State of the Art Report. *Fraunhofer Center for Experimental Software Engineering and the University of Maryland*, 1–57.

Santos, S., Albuquerque, J., & Meira, S. R. (2008). An Evaluation Approach Based on the Problem-Based Learning in a Software Engineering Master Course.

Sauermann, L. (2005). The semantic desktop-a basis for personal knowledge management. In *Proceedings of the I-KNOW* (Vol. 5, pp. 294–301).

Sauermann, L., Bernardi, A., & Dengel, A. (2005). Overview and outlook on the semantic desktop. In *Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference* (Vol. 175, pp. 1–19).

Sauermann, L., Grimnes, G., Kiesel, M., Fluit, C., Maus, H., Heim, D., Nadeem, D., et al. (2006). Semantic desktop 2.0: The gnows experience. *The Semantic Web-ISWC 2006*, 887–900.

Schaffert, S. (2006). Ikewiki: A semantic wiki for collaborative knowledge management. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE'06. 15th IEEE International Workshops on* (pp. 388–396).

Schaffert, S., Gruber, A., & Westenthaler, R. (2005). *A semantic wiki for collaborative knowledge formation*. Semantics 2005.

Schmidt, K., & Bannon, L. (1992). Taking CSCW seriously. *Computer Supported*

Cooperative Work (CSCW), 1(1), 7–40.

Segal, J. (2001). Organisational learning and software process improvement: a case study. In *Proceedings of the Third International Workshop on Learning Software Organizations* (pp. 68–82). Kaiserslautern, Germany: Springer Verlag.

Silveira, C., Faria, J. P., Aguiar, A., & Vidal, R. (2005). Wiki based requirements documentation of generic software products. In *Proceedings of the 10th Australian Workshop on Requirements Engineering (AWRE)* (pp. 42–51).

Skillscape Competence Manager. (n.d.). Retrieved from <http://www.hrhub.com/storefronts/skillscape.html>

SkillSoft Skillview. (n.d.). Retrieved from http://www.skillssoft.com/products/competency_management/skillview/default.asp

Skuce, D. (1995). Knowledge management in software design: a tool and a trial. *Software Engineering Journal*, 10(5), 183–193.

Smith, R. D. (1990). KIDS - A Knowledge-Based Software Development System. *Automating Software Design*, MIT Press, 483--514.

Smith, T. E., & Setliff, D. E. (1992). Knowledge-based constraint-driven software synthesis. In *Knowledge-Based Software Engineering Conference, 1992* (pp. 18–27).

Souzis, A. (2005). Building a semantic wiki. *IEEE Intelligent Systems*, 87–91.

Sutcliffe, A., Chang, W. C., & Neville, R. (2003). Evolutionary requirements analysis. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International* (pp. 264–273).

Sveiby, K. E. (1997). *The new organizational wealth: managing & measuring knowledge-based assets*. San Francisco: Berrett-Koehler Pub.

- Terveen, L. G., Selfridge, P. G., & Long, M. D. (1995). Living design memory: Framework, implementation, lessons learned. *Human-Computer Interaction*, 10(1), 1–37.
- Thaddeus, S., & Kasmir, R. S. V. (2006). A semantic web tool for knowledge-based software engineering. In *Proceedings of the 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006)*. Athens, G.A., USA: Springer.
- Tiwana, A. (2000). *The knowledge management toolkit: practical techniques for building a knowledge management system*. Prentice Hall PTR Upper Saddle River, NJ, USA.
- Trittmann, R. (2001). The organic and the mechanistic form of managing knowledge in software development. In *Proceedings of the Third International Workshop on Learning Software Organizations* (pp. 22–26). Kaiserslautern, Germany: Springer Verlag.
- TWiki.org. (n.d.). TWiki Success Stories. Retrieved November 30, 2010, from <http://twiki.org/cgi-bin/view/Main/TWikiSuccessStories>
- Velardi, P., Navigli, R., Cuchiarrelli, A., & Neri, F. (2005). Evaluation of ontolearn, a methodology for automatic population of domain ontologies. *Ontology Learning from Text: Methods, Evaluation and Applications*. IOS Press.
- Vitharana, P., Zahedi, F., & Jain, H. (2003). Knowledge-based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis. *IEEE transactions on Software Engineering*, 29(8), 649–664.
- Vrain, G. F. (1992). Building a Tool for Software Code Analysis A Machine Learning Approach. In *Advanced information systems engineering: 4th International Conference CAiSE'92, Manchester, UK, May 12-15, 1992: proceedings* (p. 278).
- Wangenheim, C. G., Weber, S., Hauck, J. C., & Trentin, G. (2006). Experiences on

establishing software processes in small companies. *Information and Software Technology*, 48(9), 890–900.

Weiser, M. D. (1979). *Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method*. University of Michigan.

Witte, R., Zhang, Y., & Rilling, J. (2007). Empowering software maintainers with semantic web technologies. In *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)* (pp. 37–52). Innsbruck, Austria: Springer LNCS 4519.

Wolf, R., & Zhao, J. (2007). JavaDoc + Wiki = WikiDoc - Kollaborative Dokumentationssystem für Java. Retrieved November 30, 2010, from <http://www.inf.fu-berlin.de/w/SE/ThesisWikiDocReportRobert>

Xu, B., Qian, J., Zhang, X., Wu, Z., & Chen, L. (2005). A brief survey of program slicing. *ACM SIGSOFT Software Engineering Notes*, 30(2), 1–36.

Ye, Y., & Fischer, G. (2002). Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the 24th international conference on Software engineering* (pp. 513–523).

PUBLICATIONS RELATED TO THE THESIS

This section contains references to publications resulted from my research activities to date. In summary there are 8 conference publications and 2 journal publication.

1. **Panagiotou, D.** & Mentzas, G. (2011), Leveraging software reuse with knowledge management in software development, *International Journal of Software Engineering and Knowledge Engineering - IJSEKE* (forthcoming issue).
2. **Panagiotou, D.**, Paraskevopoulos, F. & Mentzas, G. (2011), Knowledge-based interaction in software development, *Intelligent Decision Technologies Journal - IDT* (forthcoming issue).
3. **Panagiotou, D.** & Mentzas, G. (2010), A semantic user interface for managing knowledge in software development, in *Proceedings of the 5th International Conference on Software and Data Technologies*, 22-24 July 2010, Athens, Greece.
4. **Panagiotou, D.** & Mentzas, G. (2009), A semantic wiki for Software Development, in *Proceedings of the 13th Panhellenic Conference on Informatics*, 10 - 12 September 2009, Corfu, Greece.
5. **Panagiotou, D.** & Mentzas, G. (2009), A Knowledge Workbench for Software Development, in *Proceedings of the International Conference on Semantic Systems*, 2 - 4 September 2009, Graz, Austria.
6. Maalej, W., **Panagiotou, D.** & Happel, H. J. (2008), Towards effective management of software knowledge exploiting the semantic wiki paradigm, in *Proceedings of the Software Engineering 2008 (SE'08)*, volume 121 of LNI, pp. 183-197, GI, 2008.
7. **Panagiotou, D.** & Mentzas, G. (2008), Exploiting Semantics in Collaborative Software Development Tasks, in *Proceedings Maria Virvou & Taichi Nakamura, ed., JCKBSE 08*, IOS Press, pp. 385-394.
8. **Panagiotou, D.** & Mentzas, G. (2007), A comparison of semantic wiki Engines, in *Proceedings of the Knowledge and Semantic Technologies workshop, Stream: Knowledge Management, EURO 2007, 22nd European Conference on Operational Research*.

9. Papailiou, N., Apostolou, D., **Panagiotou, D.** & Mentzas, G. (2007), Exploring Knowledge Management with a Social semantic desktop Architecture, in Proceedings Roland Wagner; Norman Revell & Günther Pernul, ed., DEXA 2007, Springer, pp. 213-222.
10. Papailiou, N., Apostolou, D., **Panagiotou, D.** & Mentzas, G. (2006), Knowledge Networks in Professional Business Services Firms, in Proceedings of the Conference "Systemic Approaches in Networks of Firms-Organizations", Chios 25-27 May 2006 (conference organised in Greek).

Appendix A. ONTOLOGIES

In this annex the KnowBench ontologies are presented. These ontologies are used to describe the structure and content of knowledge artefacts, their usage and organization and to provide a basis for determining how artefacts are related.

Section A.1 What has to be modelled?

Two layers model different types of information:

- **The content layer** enables the determination of the structure and content of software objects, related problem reports as well as the solution for problems;
- **The organizational layer** describes people, various roles they assume within the organization, the projects they are involved in, etc.

In the rest of this annex the ontologies are introduced for each level and subsequently describe how these ontologies are interlinked. Due to the two ontology layers, the convention **ont:concept** is used to refer to a class **concept** defined in an ontology **ont**.

The ontologies are constructed in OWL⁶¹ (Web Ontology Language) DL, so named due to its correspondence with description logic. However, unlike OWL, the unique names assumption is made (two different names always refer to two different individuals). The explicit assertion that two individuals are equivalent is enough, a case that occurs relatively infrequently in the KnowBench ontologies' context.

Section A.2 Methodology

This section describes how the ontologies, which are introduced in this annex have been derived. Therefore, it is important to understand the goals of ontologies for the KnowBench system. In KnowBench, the goal of ontologies is less to create a “shared understanding” of the software development process, but rather to interconnect and integrate

⁶¹ <http://www.w3.org/TR/owl-ref/>

existing information (that might be useful for the more efficient software development process). Thus, the KnowBench ontology consists of two main modelling layers, namely a content layer and an organizational layer.

In order to come up with an initial set of KnowBench ontologies, it was decided to apply the OTK ontology development method (Gomez-Perez, Fernández-López, & Corcho, 2003). Initially, input from end users was collected, especially with respect to the systems, their data and artefacts they use and produce. Based on this, the end users have provided data schema and dumps with real world data. Additional input was collected from existing ontologies and material as an input for the concrete modelling task. Based on these sources, an initial version of the KnowBench ontologies was built.

In the following formalization step, a first formal representation of the KnowBench ontologies was created which was used for first integration tasks among the components of the KnowBench system architecture.

Section A.3 Content Layer

The content layer consists of several ontologies that are relatively modular and are consequently usable for the most part independently:

- the **knowledge artefact ontology** – that describes different types of knowledge artefacts such as the structure of the project source code, the components to be reused, software documentation, knowledge already existing in some tools, etc.;
- the **problem/solution ontology** – that describes the problems that may arise during coding and how they are resolved;
- the **annotation ontology** – that describes general software development terminology as well as the domain specific knowledge.

Even though the content layer has been sub-divided into the three above-mentioned ontologies that model sufficiently different domains, they are also interlinked in the content layer. These interlinks enable the expression of very complex statements regarding the given domain.

A.3.1 Knowledge artefact ontology

Knowledge artefacts, such as source code, components, documentation, issues from tracking systems, etc. typically contain knowledge that is rich in both structural and semantic information. Providing a uniform ontological representation for various knowledge artefacts enables utilizing semantic information conveyed by these artefacts and establishing their traceability links at the semantic level.

A knowledge artefact is an object that conveys or holds usable representations of different types of software knowledge. It is modelled through the top level class **ka:KnowledgeArtefact**. There are some common properties for all types of knowledge artefacts such as:

ka:hasID – which identifies a knowledge artefact in a unique manner. Recommended best practice is to identify the knowledge artefacts by means of a string conforming to a formal identification system;

ka:hasTitle – which defines a title of a knowledge artefact (e.g. class name or document title);

ka:hasDescription – which describes what a knowledge artefact is about;

ka:createdDate – which models date of creation of a knowledge artefact;

ka:usefulness – which models how frequently a knowledge artefact is used as a useful solution.

All previously mentioned properties are data properties and are modelled in the Knowledge Artefact ontology. However, many additional properties can be defined for a knowledge artefact that link it to the entities defined in the other ontologies at the same layer (e.g. the Annotation ontology) or even between the layers (e.g. the Organizational ontology).

As an example here, a short overview of the properties between the class **ka:KnowledgeArtefact** and the classes defined in the Organizational ontology at the organizational layer is presented. For knowledge artefacts it is useful to know who the author/editor is (see class **org:User**) and in which project (see class **org:Project**) it is used. For this purpose the following object properties are defined: **knowbench:hasAuthor**, **knowbench:hasEditor** and **knowbench:usedInProject**. Additionally, for each of these properties an inverse property is defined. For example, **knowbench:hasAuthor** and

knowbench:isAuthorOf are inverse properties. Since the user and project classes are defined in the Organizational ontology, the previously mentioned properties are not part of the Knowledge Artefact ontology, but rather are defined in the KnowBench ontology, which includes all ontologies that are developed within KnowBench. There are other dependencies between the Knowledge Artefact and Organizational ontology. For example, the property **knowbench:locatedIn** defines where a knowledge artefact is stored (e.g. file on a disk or in a tool). Thus, the range of this concept is the union of **org:File** and **org:Tool** that are defined in the Organizational ontology.

Knowledge artefacts other than source code, such as documentation or entries in issue tracking systems, contain rich semantic information that is not used by existing tools for knowledge management in software engineering. Introducing an ontological representation for software documentation, issue-tracking systems, etc. can assist in “understanding” parts of the semantics conveyed by these artefacts and to establish additional traceability links among these artefacts and the software artefacts. As shown in Figure A.1, in KnowBench different specialisations of the **ka:KnowledgeArtefact** class are considered, namely **ka:SoftwareArtefact**, **ka:KnowledgeDocument** and **ka:Tool-embeddedKnowledge**. The reason for that is that knowledge needed for software development tasks is contained in not only software code or components but also in some documents (cf. knowledge document in Figure A.1) used during software development process as well as in tools (cf. tool-embedded knowledge in Figure A.1) that support the software development process. Knowledge documents represent informal knowledge produced by a human usually as a text document. On the other hand, tool-embedded knowledge is knowledge collected by a tool in a semi-structured form. This structure of the **ka:KnowledgeArtefact** class enables very easy extension of the model by preserving all functionality. For example, if someone finds another software-development supporting tool which can be used as a useful knowledge source, its knowledge should be modelled as an additional subclass of the **ka:Tool-embeddedKnowledge** class.

The previously mentioned subclasses of the class **ka:KnowledgeArtefact** are also defined in the Knowledge Artefact ontology and they inherit all properties defined for the class **ka:KnowledgeArtefact**. The **ka:SoftwareArtefact** class is further split into **ka:SourceFile** and **ka:Component** classes. There are also two specialisations of the

ka:ToolEmbeddedKnowledge class, namely the **ka:Issue** class and the **ka:ChangeSet** class.

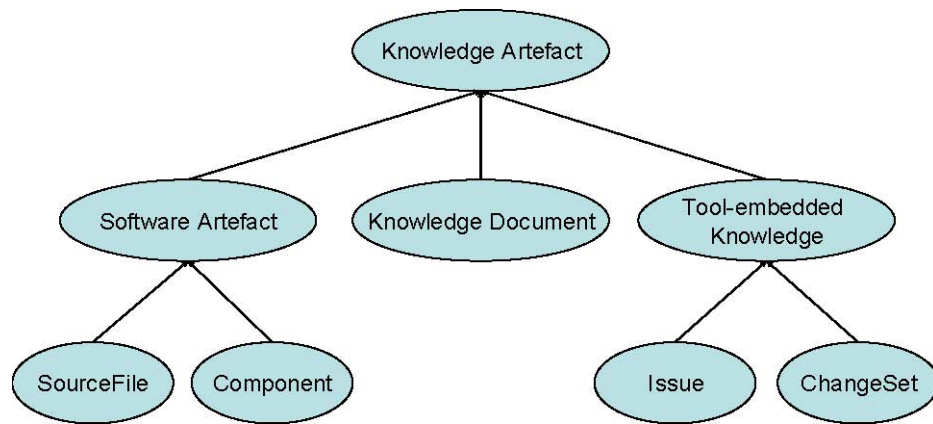


Figure A.1: The ka:KnowledgeArtefact class and its specialisations

Some additional properties are defined between the subclasses of the class **ka:KnowledgeArtefact**, namely:

ka:hasRelatedSoftwareArtefact – property defined between classes **ka:KnowledgeDocument** and **ka:Tool-embeddedKnowledge** and the class **ka:SoftwareArtefact** that models that there is a related software artefact for either a knowledge document or a tool-embedded knowledge entry;

ka:hasRelatedKnowledgeDocument – inverse property for the **ka:hasRelatedSoftwareArtefact** property defined between the class **ka:SoftwareArtefact** and class **ka:KnowledgeDocument** (and consequently all its specialisations) that models that there is a related knowledge document for a software artefact;

ka:hasRelatedToolEmbeddedKnowledge – inverse property for the **ka:hasRelatedSoftwareArtefact** property defined between the class **ka:SoftwareArtefact** and class **ka:Tool-embeddedKnowledge** that models that there is a related tool-embedded knowledge item for a software artefact;

ka:usesComponent – property defined between the class **ka:SourceFile** and class **ka:Component** that models that a component is used in a source code;

ka:isUsedInCode – inverse property for the **ka:usesComponent** property;

ka:isSourceFor – property defined between the class **ka:SourceFile** and class **ka:Component** that models that a source code file is source for a component.

Software Artefact

For the **ka:SoftwareArtefact** class the following properties are defined:

ka:hasVersionNumber – which identifies the version number of a software artefact;

ka:hasPreviousVersion – which establishes link to the previous version of the same software artefact;

ka:hasSize – which models the overall size of a software artefact in bytes;

ka:hasLicenceType – which models the type of the license. Possible values for this field may include "GPL", "BSD", "Creative Commons" etc.

The domain for all previously mentioned properties is the **ka:SoftwareArtefact** class. All properties, except the **ka:hasPreviousVersion** property, are data properties. The range for the **ka:hasPreviousVersion** property is the **ka:SoftwareArtefact** class.

The **ka:hasPreviousVersion** object property is a *transitive* property. Additionally, *consistency constraint rules* can be defined in order to check if the version numbers are correctly defined.

$$\begin{aligned} \text{ErrorVersionNumber}(X) \leftarrow & \text{hasPreviousVersion}(X,Y) \wedge \text{hasVersionNumber}(X,Nx) \wedge \\ & \text{hasVersionNumber}(Y,Yn) \wedge \text{greaterOrEqual}(Yn,Xn) \wedge \text{SoftwareArtefact}(X) \wedge \\ & \text{SoftwareArtefact}(Y) \end{aligned}$$

In what follows, design issues are discussed for the specialisations of the **ka:SoftwareArtefact** class, namely source code and components.

Source Code

The source code is the most important artefact in understanding and resolving problems. All problems occur somewhere in the source code and the ensuing interaction/discussion around the problem also takes place in referential context of the source code.

The top-level class for all software objects in KnowBench is the **ka:CodeEntity** class. This class is linked to the **ka:SourceFile** class through the transitive property **ka:isContainedIn** (or its inverse property **ka:contains**).

In the following, the subclasses of **ka:CodeEntity** are introduced. The KnowBench system should be able to indicate (roughly) the area of code where the problem is likely to be

located. Thus, it is essential to model the structure of the code as granular as possible. Thus, the following subclasses are identified:

ka:SourceObject - objects in source

ka:Package - Package or namespace

ka:Class - Class

ka:Comment - inline comments – `/**...*/`

ka:Variable – variables

ka:ClassVariable - class variable

ka:LocalVariable - local variable **-ka:Method** - method

ka:Type - types such int, float, String **-ka:PrimaryType** - primary datatypes – int, etc.

-ka:AbstractType - abstract types (including classes)

A part of the source code model and its relations to the specialisation of the knowledge artefacts i.e. source files is shown in Figure A.2.

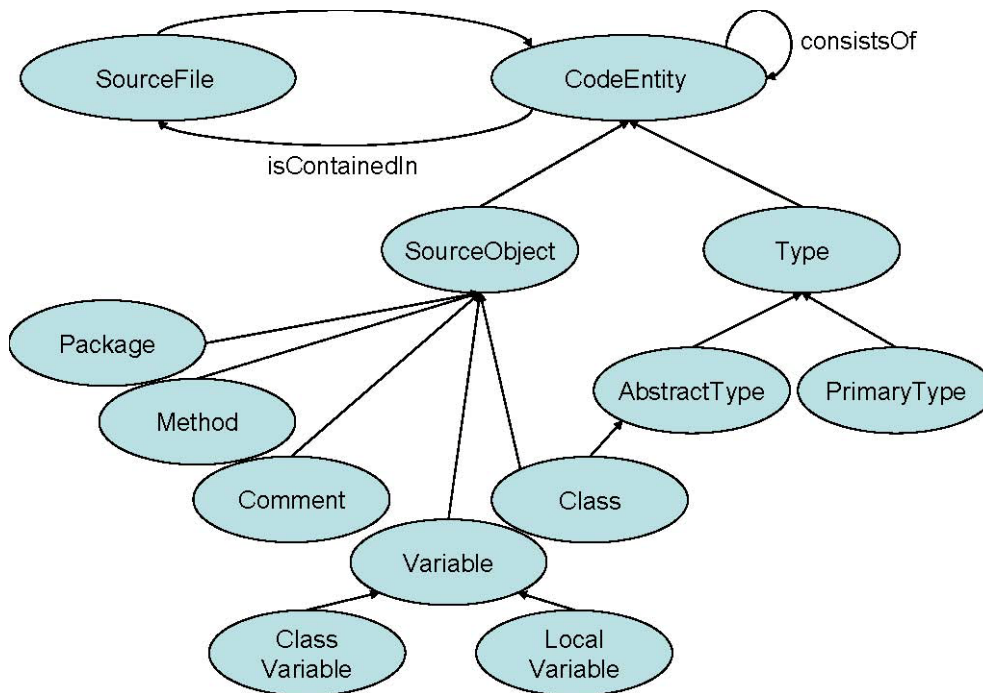


Figure A.2: A part of the source code model

Several properties are defined for the class **ka:CodeEntity**, such as **ka:hasID**, **knowbench:has Author**, transitive property **ka:consistsOf** etc. Additionally, within this sub-ontology, various object properties are defined to characterize the relationships among

classes. For example, two instances of **ka:CodeEntity** may have a **ka:definedIn** relation indicating one is defined in the other; or an instance of method **ka:read** may have an instance of **ka:LocalVariable** indicating the method may read the field in the body of the method.

In the following, the class **ka:Comment** is explained. The class **ka:Comment** models software code comments within a source code file. An instance of **ka:Comment** may be a documentation comment for a whole class or for a method, such as:

*This procedure, given a content item and a privilege,
checks to see if there are any children of the item on
which the user does not have that privilege. It returns
0 if there is any child item on which the user does not
have the privilege. It returns 1 if the user has the
privilege on every child item.*

or an inline comment such as:

FIXME: db blob get file is failing when it used to bind variable

The class **ka:Comment** also has a functional datatype property **ka:hasCommentText**, which links an instance of **ka:Comment** to the textual string of the comment.

Finally, all classes in the code part of the Knowledge Artefact ontology have a direct mapping to source code entities and can therefore be automatically populated through source code analysis. Also note, that by restricting the granularity of the model (e.g. concerning low-level entities such as **ka:LocalVariables**) the volume of extracted data turns out to be too large in practice.

Component

Several definitions of components and reusable components have been provided in the literature (van den Elst, van Harmelen, & Thonnat, 1995). Components can be seen as some part of a software system that is identifiable and reusable; functions and classes are examples of such components. In the context of KnowBench, components can be seen as the next level of abstraction after functions, modules and classes. A component is an object written to a specification that does not share state and communicates by exchanging messages carrying

data. It can be deployed to the system. Thus, a software component is a system element offering a predefined service and is able to communicate with other components.

There is extensive literature on representation formats for reusable software components (see e.g. (A. Mili, R. Mili, & Mittermeir, 1998), (Vitharana, Zahedi, & Jain, 2003)). However, those approaches lack a broad adoption and no standard emerged yet (Frakes & Kang, 2005). One major reason is that they require detailed and heavyweight descriptions, which are seldomly in place.

Therefore, lightweight and collaborative approaches are considered as a possible remedy for these problems. The approach for component description is thus based in the annotation ontology. Recent work such as the DOAP Ontology⁶² shows that such lightweight approaches, combined with semantic web technologies might be a good trade-off between creation effort and usefulness of metadata.

In order to allow a user (e.g. software developer) performing a lookup for the components he is in need of, their differences must be explicit. A component is defined and then specialized. The following properties are defined for the top class **ka:Component**:

ka:hasExtensionCapability – this property describes the possibilities for customisation;

ka:implementsInterface – a component is identified by its interface and thus has attributes like the name of the interface it implements. The interface of a component is modelled as the **ka:Interface** class and **ka:consists** of connection parameters and operations. The **ka:parameters** refer to other components. **ka:Operations** play a role in the implementation of a component; they are services that are provided to other components.

ka:dependsOn – other instances of **ka:Component**, which are required in order to execute a component. There is an inverse relationship **ka:hasDependentComponent**.

Note that the properties **content:isAbout** and **content:isRelatedToSEEntity** are also available to annotate components with technical and domain concepts.

⁶² <http://usefulinc.com/doap/>

In order to enable better search for components it would be necessary to specify for each component what kind of component it is. Thus, the **ka:Component** class is specialized based on functionality in the following way (van den Elst et al., 1995):

SystemComponent: component providing basic system functionality, e.g. the registry or a connector;

ka:FunctionalComponent: component that is of interest to the client and can be looked up;

ka:ExternalService: an external service cannot be deployed directly as it may be programmed in a different language, live on a different computing platform, uses unknown interfaces, etc. It equals a functional component from a client perspective. This is achieved by having a proxy component deployed as surrogate for the external service;

ka:ProxyComponent: special type of functional component that manages the communication to an external service.

Knowledge Document

Documentation is considered as an important part of software engineering since it contains knowledge (e.g. about source code) that can be shared. However, existing source-document traceability research (Antoniol, Canfora, Casazza, & De Lucia, 2000) mainly focuses on connecting documents and source code using information retrieval techniques. Consequently, these approaches typically ignore structural and semantic information that can be found in both documents and source code, limiting therefore both their precision and applicability.

Thus, the Knowledge Document ontology models physical documents, e.g. word documents or PDF documents and classifies them based on their type. The top level class is the **ka:KnowledgeDocument** class and it is specialized at the first level into:

ka:Architecture/DesignDocumentation – this type of documentation gives an overview of software, includes relations to an environment and construction principles to be used in design of software components;

ka:TechnicalDocumentation – it is documentation of code, algorithms, interfaces, and APIs;

ka:EndUserDocumentation – it models manuals for the end-user, system administrators and support staff.

The most important properties of the class **ka:KnowledgeDocument** (besides properties defined for the superclass **ka:KnowledgeArtefact**) are:

ka:keyword – the topic of the content of a knowledge document, as keyword. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme;

ka:writtenInLanguage – language a knowledge document is written in;

ka:dependsOn – a knowledge document depends on another knowledge document in order to be properly understood/used/interpreted;

ka:hasComplexityLevel – indicator of the pre-requested knowledge level for using a document. Possible values are in range 1 to 10;

ka:hasSize – the overall size of a knowledge document in bytes.

Tool-embedded Knowledge

The **ka:Tool-embedded Knowledge** class models knowledge already captured using different tools that are used during the software development process such as issue tracking systems or configuration management systems. These tools acquire information that is very useful for KnowBench's most important tasks. However, this solution still requires significant human processing/understanding, and does not provide any support for integration.

Thus, explicitly and formally knowledge was modelled that is embedded in the issue tracking systems and/or configuration management systems. This knowledge is represented in the Tool-embedded Knowledge ontology. Explicit modelling enables machine understandability (and not only human understandability as in the existing systems).

This ontology facilitates interoperability between different issue tracking systems and/or configuration management systems by providing a shared understanding of the

domain in question. In this way problems caused by structural and semantic heterogeneity⁶³ of different models can be avoided.

The most important concept of the tool-embedded Knowledge is the class **ka:Tool-embeddedKnowledge** and its subclasses such as **ka:Issue** and **ka:ChangeSet**. Several properties are defined:

ka:hasRelatedKnowledgeArtefact – this property establishes explicit link between a tool-embedded knowledge item and a corresponding knowledge artefact⁶⁴;

knowbench:isStoredIn – this property identifies a tool where a tool-embedded knowledge item is captured. Since the class **org:Tool** is modelled in the Organization ontology, this object property is defined in the global KnowBench ontology.

A short explanation is given for the reason to integrate knowledge that is captured in the issue tracking and the configuration management systems:

Why issue tracking systems: Identifying concepts that appear in issue reports is essential for knowledge management support that the KnowBench system provides. Certain concepts, such as variable names, function names and log messages, are likely to appear in issue/bug reports and provide valuable hints about the code that is involved in producing the bug. These concepts are modelled by the Knowledge Artefact ontology (in the code part of it) in order to link from information in an issue/ bug report to areas in the code that cause the problem. The Issue class therefore contains property **ka:hasRelatedKnowledgeArtefact** that is used for establishing the link;

Why configuration management systems: Identifying people who committed changes to code is also very important from the knowledge management point of view: The evolution of a code base is logged in commit logs as commits are made by developers to various files. It is important to model commits in the KnowBench ontology to be able to locate people who have often committed to a particular file and are likely to have expertise in the code contained in the file.

⁶³ Structural heterogeneity results when different systems store their data in different schemes. Semantic heterogeneity involves similar problems in the context and intended meaning of information

⁶⁴ An issue does not refer only to software artefacts but rather can refer to other issues or attached documents

Issue

An issue is a general notion in Software Engineering which represents a “request”, mostly from a user/customer perspective that should be implemented by the system developer. Thus, an issue is often also called work item or a task, from a developer’s perspective. Issues are managed or “tracked” in so-called issue tracking software such as JIRA, Bugzilla or Mantis. Ticketing systems are also much related to issue tracking and indeed used for that purpose by software developers.

Information from such issue tracking systems is very valuable from a knowledge management perspective, since it provides rationale for changes and other contextual information which is directly related to source code. In system maintenance, many changes are actually driven by issues. It is a development best practice, to include an issue ID in commit messages after changing the code.

The general class of an **ka:Issue** can be subclassed with the classes **ka:Bug**, **ka:Defect** or **ka:Enhancement**, which represent different types of issues a user might have. An issue inherits all properties defined for its parents (e.g. **ka:hasID**, **ka:hasTitle**, **ka:hasDescription**, etc.) The general properties of an issue are as follows:

knowbench:hasReporter – user who raised the issue;

knowbench:hasAssignee – developer who is assigned to work on the issue;

ka:startDate – timestamp for the creation of the issue;

ka:stopDate – timestamp for closing the issue;

knowbench:hasProject – project which the issue relates to;

ka:hasComponent – sub-components or packages which the issue relates to;

ka:relatesToVersion – version of the project the bug relates to;

ka:hasStatus – status of the issue;

ka:hasSeverity – level of severity;

ka:relatedIssues – other issues which are dependent;

ka:hasAttachment – files (e.g. screenshots) attached to the issue;

content:hasComment – comments to the issue – i.e. using **content:Comment**.

ChangeSet

Configuration management (or version control) systems are commonly used in software development to allow for collaborative development. Configuration management systems allow developers to work isolated from each other and merge their changes once they finished their tasks.

Thus, configuration management systems contain valuable information about the “history” of an artefact, such as the developers who changed them, commit messages which textually describe changes and other artefacts that were changed in parallel. In most common configuration management systems this information is captured in an abstraction called **ka:ChangeSet** which accordingly has the following attributes:

ka:id – a unique identifier for the **ka:ChangeSet** within the scope of its system;

ka:parentId – a pointer to the previous **ka:ChangeSet**;

ka:file – a list of **org:File** instances which was changed;

ka:commitMessage – textual description of changes;

knowbench:hasAuthor – author of the change;

ka:timestamp – timestamp of the commit.

A.3.2 Problem/solution ontology

Problem-solution ontology models the problems occurring during the software development process as well as how these problems can be resolved. Such problems may be manually created by a developer, using the GUI of the KnowBench system.

Problem

A problem is an obstacle which makes it difficult to achieve a desired goal, objective or purpose. It is modelled through the class **ps:Problem** and its properties: **-ps:hasID** – a problem is identified by its identifier; **-ps:hasTitle** – a summary of a problem; **-ps:hasDescription** – a textual description of a problem; **-ps:hasSeverity** – it models how severe the problem is;

ps:hasSolution – every problem asks for a solution. This is the inverse property for the property **ps:isRelatedToProblem** defined for the class **ps:Solution**;

ps:hasSimilarProblem – it models similar problems (see below);

knowbench:isDetectedBy – a problem is detected by an event;

content:hasContext – a problem occurs in some context, i.e. it refers to a software artefact. This information is captured automatically by applying rule-based inference as described below;

knowbench:isIdentifiedBy – a problem can be identified by some **org:User**.

There are different types of problems that are modelled as subclasses of the class **ps:Problem**. Some of them are:

ps:Error – it refers to an incorrect action or calculation performed by software.

ps:LogicError – It is a bug in a program that cause it to operate incorrectly, but not to fail. Because a logic error will not cause the program to stop working, it can produce incorrect data that may not be immediately recognizable. With a logic error, the program can be compiled or interpreted (supposing there are no other errors), but produces the wrong answer when executed.

ps:RuntimeError – It is a bug that is not discovered until the program is tested by a test case or in a "live" environment with real data, despite sophisticated compile-time checking and pre-release testing.

ps:InformationNeedProblem

ps:ModellingProblem: Software developers may not be able to find the right information;

ps:QueryRepresentationProblem: This problem occurs when a user represents his/her need ambiguously in a query.

ps:AnnotationProblem: This problem occurs when software artefacts are not semantically correct annotated, which means that the content of an artefact is represented either ambiguously or incorrectly.

ps:RepositoryProblem:

ps:InformationGap: There is no relevant artefact for the user's information need.

ps:InformationOverload: There are too many relevant artefacts for the user's information need.

ps:KnowledgeSupportProblem: The problem occurs when the found artefact does not contain all the information and information in right/understandable format needed by the user (e.g. user needs example code on how to code against an external component w.r.t the component he is working on).

ps:SystemProblem: a problem in the mechanism/model used for calculating relevance which results in placing a highly relevant artefact below a low-relevant artefact in the list of results.

ps:UnderstandingProblem: software developers may not be able to understand how to search for or apply the new information;

ps:KnowledgeablePeerSearchProblem: software developers may not know to whom they can turn for help on this particular problem;

ps:KnowledgeSharingProblem: experts who are able to help may not be willing to due to the interruption to their own work and other various reasons;

ps:Issue: it is a problem such as a requested feature, missing documentation and so forth.

ps:ComplexityProblem: This problem arises because of the very large number of software elements, states assumed by these elements and the nonlinear interaction between these elements;

ps:UsabilityProblem: These problems are related to the effectiveness, efficiency and satisfaction of a user to achieve specified goals in a specified context of use;

ps:EffectivenessProblem: These problems take place when a user cannot achieve specified goals accurately and competently;

ps:EfficiencyProblem: These problems occur when too many resources (time, space, etc.) are needed in relation to the accuracy and completeness with which a user achieves goals;

ps:SatisfactionProblem: These problems are subjective and are related to the freedom from discomfort and positive attitudes towards the use of the software;

ps:RedundancyProblem: This problem occurs when a code that is executed has no effect on the output of a program;

ps:Dead-codeProblem: This problem occurs when a code that exists in the source code a program can never be executed;

ps:ScalabilityProblem: This problem occurs when a code cannot function effectively as the size of data increases;

ps:MaintenanceProblem: This problem arises when there is a need to modify a code.

Each of these classes has its own properties. For example, error codes are enumerated messages that correspond to errors. They are typically used to identify faulty hardware, software, or incorrect user input. Error codes are typically identified by number, each indicating a specific error condition.

Solution

A solution is a statement that solves a problem or explains how to solve the problem. It is represented through the ps:Solution class and its properties:

ps:hasID – an unambiguous reference to a solution;

ps:isRelatedToProblem – a solution is defined as the means to solve a problem;

knowbench:isDefinedBy – a solution is defined by an org:User;

ps:hasCreationDate – date of creation of a solution;

ps:hasDescription – a textual description of a solution;

knowbench:isResolvedBy – a set of events that occurred while solving the problem;

ps:hasPrerequisite – a solution requires a level of expertise in order to be used;

content:hasComment – a solution can be commented by the people that used it. The range of this property is the class content:Comment;

content:suggestsKnowledgeArtefact – a solution recommends using a knowledge artefact in order to resolve a problem the solution is defined for;

content:isAbout – a solution can be annotated with the domain entities;

content:isRelatedToSEEntity – a solution can be annotated with the general knowledge about software engineering domain.

There are additional properties (e.g. the `knowbench:hasAuthor`) that are modelled in the KnowBench ontology, since the target of these properties is the class `org:User` defined in the Organizational ontology. Similarly, the properties `content:isAbout`, `content:isRelatedToSEEntity` and `content:hasComment` are modelled in the Content Ontology that includes all ontologies at the content layer.

ps:frequencyOfUsage – it is calculated based on the number of request for the solution;

ps:usefulness – usefulness of a solution is calculated as average values of the `ps:hasUsefulnessLevel` property instantiation;

ps:hasSimilarSolution – two solutions related to similar problems of the same problem type are similar.

A.3.3 Annotation ontology

This part of the content layer identifies a unified vocabulary that ensures unambiguous communication within a heterogeneous community. This vocabulary can be used for the annotation of the knowledge artefacts. Two different types of annotations can be distinguished:

Domain annotation – Software providers in different domains should classify their software using a common vocabulary for each domain. Common vocabulary is important because it makes both the users and providers to talk to each other in the same language;

Software engineering annotation – General knowledge about software domain including software development methodologies, design patterns, programming languages can be also used for annotation.

By supporting different types of annotation it is possible to consider information about several different aspects of the knowledge artefacts.

The vocabulary will be used for the annotation of the knowledge artefacts, e.g. code, knowledge documents, etc. as well as expertise of the users and solutions of problems. These annotations can be used to establish the links between these different knowledge artefacts.

Domain Ontology

Based on the description of the ontologies that was given in the previous section, it can be concluded that, these ontologies are general enough to be applied and model software developed for any domain. On the other hand, the entities from the domain ontology are domain specific and cannot be easily generalized to other software projects. Consequently, the domain ontology has to be developed for each software project separately.

The content of knowledge artefacts themselves is essential toward determining particular topical interests and understanding the relationships between knowledge artefacts. The domain ontology should consist of concepts and relations modelling meaning of content of a knowledge artefact. This may include already existing categorisation of services (e.g. content management) as well as typical community terminology (e.g. types of existing documents such as invoice, receipt, etc.).

Construction of this ontology can be a very time consuming and error-prone task for the domain experts. In order to get a feeling on what are the relevant topics for knowledge artefacts, what are the relations between them and at the end to assign each knowledge artefact to some certain domain entities; the domain expert has to read all the knowledge artefacts and to understand them. This can be overcome by a special tool based on NLP methods applied on the content of knowledge artefacts which helps domain expert by suggesting the topics, showing the importance of topics so far, and putting them into the right place into the ontology (including the relationships with existing entities).

Software Engineering Ontology

The Software Engineering ontology consists of a large body of concepts that are expected to be discovered in software documents or in the comments in code. These concepts are based on various programming domains, including programming languages, algorithms, data structures, and design decisions such as design patterns and software architectures. This ontology is used for the annotation of different knowledge artefacts (e.g. code). The annotations can be created automatically through a text mining system by adapting it for the software engineering domain.

The software engineering ontology will be used for the annotation of knowledge artefacts (e.g. code), solutions as well as for users to state their areas of expertise.

Section A.4 Organizational Layer

The semantics of the organizational layer is captured in the Organizational ontology. There are several key classes in this ontology:

-class org:Organization – it represents an organization. An organization has short and long name, address, size, etc., may be involved in many projects and has resources. There is a distinction between private and public organizations. Each of them can be further specialized;

-class org:Resource – Resource is something which can be used to satisfy organizational needs – in the context of KnowBench it means that a resource is required to carry out software development tasks. A resource can be almost anything, including both real-world entities, such as a specific piece of hardware (a CPU) or software (a database), and virtual entities, such as business applications or logical IT services. The resource model is central to representing what is being managed. The class org:Resource represents all resources that can be used for support or help by the organization. There exist two types of resources: (1) human resources (i.e. user) who perform an activity (e.g. coding) and (2) non-human resources that are occupied by the activity. The non-human resources include: (i) equipment (i.e. hardware, tools etc.) that is essential for the production of software; and (ii) technology which refers to the process or method by which software is produced;

-class org:File and org:Tool are subclasses of org:Resource which represent physical files resp. a specific tool (e.g. as a source for tool-embedded knowledge).

-class org:Project – A project is a temporary endeavour undertaken to create a unique product or service. A project has a start date and end date and is related to some entities (either from the Domain ontology or from the Software Engineering ontology). A project needs resources to deliver the results. It is realized by a team of people and can be divided into subprojects;

-class org:Team – A team comprises a group of people linked for a common purpose. Teams normally have members with complementary skills and generate synergy through a coordinated effort;

-class org:User – it is a subclass of the class org:Resource and models only individual users and not teams. The following properties are defined:

- personal category that contains information about names, contacts, etc. (e.g. `org:hasID`, `org:hasFirstName`, `org:hasLastName`, `org:hasEMailAddress`, etc.);
- relations category that specifies relationships between users (e.g. `org:colleagueOf`, `org:managerOf`, etc.);
- user areas of expertise (explicitly stored in a user profile). This is modelled through the property `knowbench:hasAreaOfExpertise`. The range of this property is the top class in the Software Engineering ontology;
- user expertise (level) for software engineering artefacts – this information is needed to justify quality of suggested knowledge. An example is given below:
 - A user who is author of frequently-used and positively assessed solutions for a problem about a software engineering artefact has expertise level N for that artefact;
 - user experience (level) for software engineering artefacts.

There are different ways to model levels of expertise/experience:

- as a list of possible values (e.g. novice, expert, intermediate, knowledgeable, etc.) or
- in the range [0, ..., N]

In KnowBench, the second approach is applied, where the value for N as well as interpretation of values depends on the organization that uses the KnowBench system. 0 means that a user does not have any expertise for a software engineering artefact, whereas N means that he/she is specialist for that topic.

To model expertise/experience for a software engineering artefact, an additional class is introduced, namely `org:Competence`. This class is a target for the `org:hasExpertiseForSEEntity` and `org:hasExperienceForSEEntity` properties. Additionally, the class `org:Competence` has two properties:

- the data property `org:hasLevel` whose values are in range 1 to N and models the level of competence and
- the object property `knowbench:hasCompetenceFor` whose range is the top class in the Software Engineering ontology and determines what kind of competence.

This part of the Organisational ontology is shown in Figure A.3.

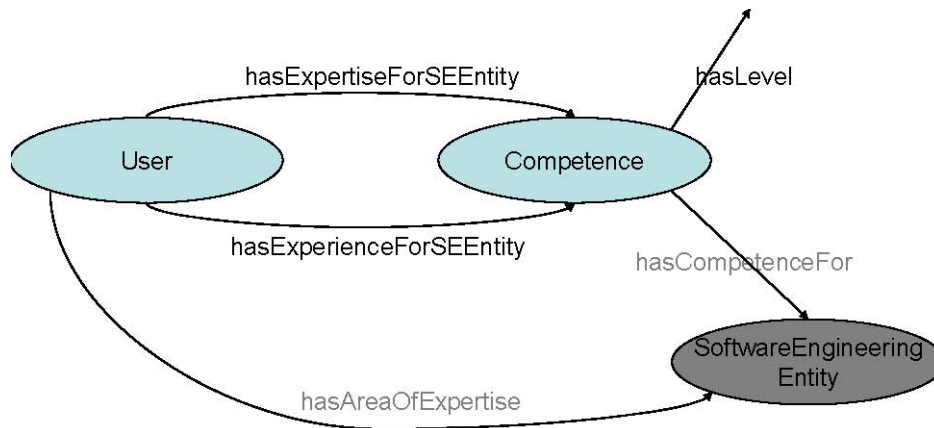


Figure A.3: Modelling of expertise and experience for a software engineering artefact. Grey colour means that the entities are not defined in the Organisational ontology.

Modelling the level of competence (i.e. expertise and/or experience) in a certain software engineering area is very important from the knowledge management point of view. It provides support for knowledge provenance (i.e. who/what is origin or source from which knowledge comes). Additionally, this accounts for the fact that problem solving strategies:

- a. are in general different for different levels of competence;
- b. can sometimes only be applied if the user has a certain level of competence.

Different ways to exploit user expertise are possible in KnowBench: -User expertise serves to drive query compilation when a problem is discovered. For example, problem discovery might depend on expertise level; -User expertise serves query execution (when the solution should be ranked). Some examples:

- i. A solution to be applied requires certain level of expertise;
- ii. Solutions written by user with higher expertise level in domain are ranked firstly;
- iii. Solutions positively assessed by user with higher expertise level in domain should be highly ranked;

User expertise determines the way of data delivery. For example, solutions for different expertise levels should be presented differently.

There are many relations between the above introduced classes: the class `org:Organization` that contains numerous resources (`org:Resource`), is involved in many projects (`org:Project`) and employs users (`org:User`) organised in teams (`org:Team`) who work to develop the code, fix problems, provide support on using the program, discuss future

evolution of the software and so on. A user represents a member of an organization, linked to an instance of the organization through the object property `org:worksFor`. Users work in projects or manage them, they can be either members or managers of a team. Each project needs resources to be performed. Consequently, non-human resources must be used in projects. Some relations are mutually inverse (e.g. `org:employs` and `org:worksFor`). Some properties are marked as symmetric (e.g. `org:cooperatesWith`) or transitive. Further, there exist many rules that enable inferring new knowledge. One example would be “If a Project has a Team and a User is a member of that Team, then he or she works in the Project”.

$$\text{worksIn}(U,P) \leftarrow \text{User}(U) \wedge \text{Project}(P) \wedge \text{Team}(T) \wedge \text{hasTeam}(P,T) \wedge \text{isMemberOf}(U,T)$$

Section A.5 KnowBench Ontology

There exist some dependencies between ontologies on different layers. In order to enable keep these ontologies as modular as possible, the interdependencies between ontologies are modelled in the so-called KnowBench ontology that includes all separate ontologies. Indeed, the top-level KnowBench ontology essentially consists of all ontologies defined in the Content and Organizational layers. The purpose of the ontologies is to model the structures of the two layers, enabling artefacts and information from various parts of the project to be linked.

The content layer of KnowBench is modelled through the Content ontology that consists of three different ontologies:

- Knowledge Artefact ontology, which describes the project source code, components, knowledge documents and knowledge in the existing systems,
- Problem-Solution ontology, which describes possible problems, their solutions and their attributes, and
- Annotation ontology, which describes common terms from software engineering or particular domains.

These ontologies are connected as already described. For example, code files modelled in the code part of the Knowledge Artefact ontology might be causes of problems that are modelled in the Problem-Solution ontology.

The Organizational ontology describes the people in software development projects and various roles they assume within the organization.

Figure A.4 shows relations between Content and Organizational ontologies. A user modelled in the Organization Ontology located in the organizational layer can be an author and an editor of knowledge artefacts modelled in the Knowledge Artefact ontology that is included in the Content ontology. A knowledge artefact is stored as a file on a disk or in a tool which are modelled in the Organizational ontology. Additionally, solutions for problems and comments of solution modelled in the content layer are defined/added by the users as well. Moreover, tool-embedded knowledge (i.e. issues) that is part of the content layer is stored in tools modelled at the organizational layer. Tool-embedded is also related to the users. For example, an issue is raised by and/or is assigned to a user or a user is an author of a change stored in a configuration management system. Next, knowledge artefacts are used in projects that are also part of the Organizational ontology. An issue is also related to a project. A project is about some domain entities and/or can be related to the software engineering artefacts. Finally, a user has area of expertise and/or competences in software engineering artefacts.

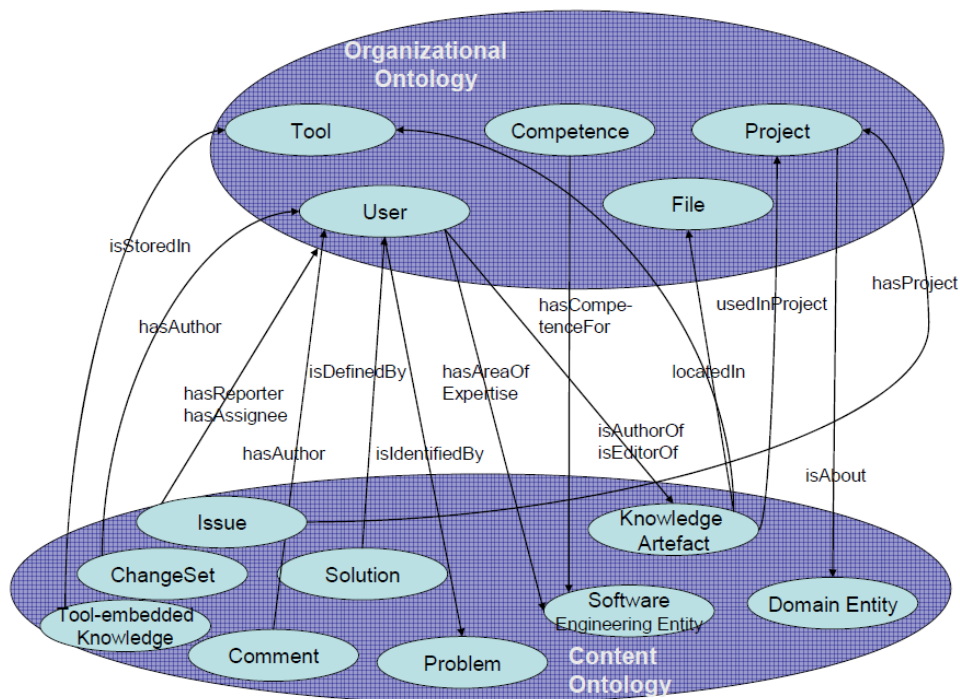


Figure A.4: Relations between Content and Organizational ontologies

Appendix B. EVALUATION QUESTIONNAIRES

The aim of this questionnaire was to assist to the summative evaluation of the KnowBench system. The end-users completed the questionnaire after the KnowBench system was developed.

Section B.1 General Knowledge Management Questions

Q1: What is your perception of Knowledge Management?

Q2: Do you have experience with Knowledge Management systems (used before)?

Q3: What were your expectations from the KnowBench knowledge management system?

Q4: Would you agree with the entitlement of KnowBench as knowledge management system?

Q5: Do you find the KnowBench knowledge management functionalities sufficient for the needs of your job/practice? If not, please explain

Section B.2 KnowBench performance during Knowledge Identification

Goal	Purpose	Ease													
	Issue	Knowledge identification													
	Object	KnowBench system													
	Viewpoint	Developer													
Question		Q1: How adequate is the level of detail of knowledge created in KnowBench?													
Metrics		<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>				1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question		Q2: Do you feel that part of the information/knowledge that could be of use to you (regardless of the development phase) is not captured in KnowBench?													
Metrics		<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>				1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
If “Not at all”, please specify what is not captured															
Question		Q3: Do you find it useful to use abstraction mechanisms (i.e. hierarchies) in dealing with the content (e.g. to be able to classify an error in several types)?													
Metrics		<table border="1"> <tr> <td>1 (Strongly disagree)</td><td>2 (Disagree)</td><td>3 (Neither agree nor disagree)</td><td>4 (Agree)</td><td>5 (Strongly agree)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>				1 (Strongly disagree)	2 (Disagree)	3 (Neither agree nor disagree)	4 (Agree)	5 (Strongly agree)					
1 (Strongly disagree)	2 (Disagree)	3 (Neither agree nor disagree)	4 (Agree)	5 (Strongly agree)											
Question		Q4: Is it easy to navigate through the hierarchy?													

Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)							
1 (Not at all)	2	3	4	5 (Definitely)									
Question	Q5: Are class hierarchies rich enough to model knowledge created/used during coding?												
Metrics													
	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table>	YES		NO									
YES		NO											
Question	Q6: Do you think that background knowledge of ontologies is required in order to effectively use KnowBench system?												
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table> If “Not at all”, please clarify what kind of knowledge would be useful	1 (Not at all)	2	3	4	5 (Definitely)							
1 (Not at all)	2	3	4	5 (Definitely)									
Question	Q7: Are there any classes that you used very frequently?												
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table>	YES		NO									
YES		NO											
Question	Q8: Are there any classes that you did not use at all?												
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table>	YES		NO									
YES		NO											
Question	Q9: What proportion of properties is instantiated during creating individuals?												
Metrics	<table border="1"> <tr> <td>1 (≤ 10)</td><td>>10</td><td>>50</td><td>>75</td><td>>90</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table> <table border="1"> <tr> <td>%</td><td></td></tr> </table>	1 (≤ 10)	>10	>50	>75	>90						%	
1 (≤ 10)	>10	>50	>75	>90									
%													
Question	Q10: Were KnowBench ontologies stable during trial?												
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table>	YES		NO									
YES		NO											
Question	Q11: Are there any classes in hierarchies which are missing?												
Metrics													

	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table>	YES		NO	
YES		NO			
	If Yes, please provide more details				
Question	Q12: Are there any properties missing?				
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table>	YES		NO	
YES		NO			
	If Yes, please provide more details				
Question	Q13: How many new tags (i.e. annotation classes) did you create during trial?				
Metrics	<table border="1"> <tr> <td>Nr.</td><td></td></tr> </table>	Nr.			
Nr.					

Q14: What would be your suggestion for improvement of knowledge structure in KnowBench?

Section B.3 KnowBench performance during Knowledge Acquisition

Goal	Purpose	Improve													
	Issue	Knowledge acquisition of existing knowledge													
	Object	KnowBench system													
	Viewpoint	Developer													
Question		Q1: How understandable for you is the purpose of the crawling process?													
Metrics		<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>				1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question		Q2: Do you find it useful to have access to knowledge acquired in the past?													
Metrics		<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>				1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question		Q3: Is the crawling process clear and easy to follow?													

Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q4: Is it clear what the result of the crawling is?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q5: Are the supported types of knowledge sources (i.e. repositories) satisfactory?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q6: Is there any type of knowledge sources relevant for coding that is not supported?										
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table> If Yes, please explain	YES		NO							
YES		NO									
Question	Q7: How many knowledge source types were used in trial?										
Metrics	<table border="1"> <tr> <td><3</td><td>3<=x<5</td><td>5<=x<7</td><td>7<=x<10</td><td><=10</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	<3	3<=x<5	5<=x<7	7<=x<10	<=10					
<3	3<=x<5	5<=x<7	7<=x<10	<=10							
Question	Q8: Is it easy to identify which information should be provided in order to crawl a repository?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q9: Did you always receive notification when the crawling is completed?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							

Question	Q10: Are you satisfied with the support for consistency checking of the provided input for crawling?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td> <td>2</td> <td>3</td> <td>4</td> <td>5 (Definitely)</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q11: Can you easily manage (i.e. add, remove, change) the repositories that will be crawled?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td> <td>2</td> <td>3</td> <td>4</td> <td>5 (Definitely)</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q12: How long is the average response time?														
Metrics	<table border="1"> <tr> <td>1 (Not acceptable)</td> <td>2</td> <td>3</td> <td>4</td> <td>5 (Acceptable)</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>					1 (Not acceptable)	2	3	4	5 (Acceptable)					
1 (Not acceptable)	2	3	4	5 (Acceptable)											

Q13: What would be your suggestion for improvement regarding run-time knowledge acquisition in KnowBench?

Q14: What would be your suggestion for improvement regarding acquisition of existing knowledge?

Section B.4 KnowBench performance during Knowledge Development

Q1: Before answering the questions below please explain what the main reason to create knowledge items was

Goal	Purpose	Ease
	Issue	Knowledge development
	Object	KnowBench system
	Viewpoint	Developer
Question	Q2: How understandable is the meaning of knowledge items?	
Metrics		

	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q3: How many knowledge items did you create?										
Metrics	<table border="1"> <tr> <td>None</td><td>< 10</td><td>< 30</td><td>< 50</td><td>A lot</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	None	< 10	< 30	< 50	A lot					
None	< 10	< 30	< 50	A lot							
Question	Q4: Does the creation of knowledge items disturb you in your daily work?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q5: How easily can you understand how to create a new knowledge item using knowledge editor?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q6: How easily can you understand how to create a new knowledge item using wiki-based editor?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q7: How easily can you understand what is required as input data for creating knowledge item?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q8: How much effort is required on defining a new knowledge item using knowledge editor?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q9: How much effort is required on defining a new										

	knowledge item using wiki-based editor?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q10: What is the time taken to create a knowledge item using knowledge editor?														
Metrics	<table border="1"> <tr> <td>1 (A lot of)</td><td>2</td><td>3</td><td>4</td><td>5 (very little)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (A lot of)	2	3	4	5 (very little)					
1 (A lot of)	2	3	4	5 (very little)											
Question	Q11: What is the time taken to create a knowledge item using wiki-based editor?														
Metrics	<table border="1"> <tr> <td>1 (A lot of)</td><td>2</td><td>3</td><td>4</td><td>5 (very little)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (A lot of)	2	3	4	5 (very little)					
1 (A lot of)	2	3	4	5 (very little)											
Question	Q12: Does the KnowBench system provides you with the friendly and easy-to-use forms for creating knowledge items?														
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td><td></td></tr> </table>					YES		NO							
YES		NO													
Question	Q13: How much guidance is provided in creating knowledge items using wiki-based editor?														
Metrics	<table border="1"> <tr> <td>1 (very little)</td><td>2</td><td>3</td><td>4</td><td>5 (A lot of)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (very little)	2	3	4	5 (A lot of)					
1 (very little)	2	3	4	5 (A lot of)											
Question	Q14: Does KnowBench allow for timely access to information?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											

Question	Q15: How clear is the meaning of annotation?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q16: How clear is the purpose of annotation?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q17: Is the granularity level of source code that can be annotated sufficient?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q18: How understandable is the manual approach for annotation creation?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q19: How much effort is required for manual annotation creation?														
Metrics	<table border="1"> <tr> <td>1 (A lot of)</td><td>2</td><td>3</td><td>4</td><td>5 (None)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (A lot of)	2	3	4	5 (None)					
1 (A lot of)	2	3	4	5 (None)											
Question	Q20: How understandable is the semi-automatic approach for annotation creation?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q21: How much effort is required for semi-automatic creation of annotations?														
Metrics	<table border="1"> <tr> <td>1 (A lot</td><td>2</td><td>3</td><td>4</td><td>5</td></tr> </table>					1 (A lot	2	3	4	5					
1 (A lot	2	3	4	5											

	of)				(None)										
Question	Q22: How much are you satisfied with the suggestions for semi-automatic creation of annotations?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q23: How many annotations proposed by the semi-automatic approach have you chosen?														
Metrics	<table border="1"> <tr> <td>None</td><td>< 3</td><td>< 10</td><td>< 20</td><td>A lot</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					None	< 3	< 10	< 20	A lot					
None	< 3	< 10	< 20	A lot											
Question	Q24: How many annotations did you create?														
Metrics	<table border="1"> <tr> <td>None</td><td>< 10</td><td>< 30</td><td>< 50</td><td>A lot</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					None	< 10	< 30	< 50	A lot					
None	< 10	< 30	< 50	A lot											
Question	Q25: Does KnowBench system provides you with the friendly and easy-to-use forms for creating annotations?														
Metrics															
	YES		NO												

Q26: What would be your suggestion for the improvement of knowledge development in KnowBench?

Q27: What would be your suggestion for the improvement of annotation creation in KnowBench?

Section B.5 KnowBench performance during Knowledge Sharing

Goal	Purpose	Increase
	Issue	Knowledge sharing
	Object	KnowBench system
	Viewpoint	Developer
Question	Q1: How easy is it to share knowledge?	
Metrics		

	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q2: How time-consuming is knowledge sharing?										
Metrics	<table border="1"> <tr> <td>1 (A lot)</td><td>2</td><td>3</td><td>4</td><td>5 (Not at all)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (A lot)	2	3	4	5 (Not at all)					
1 (A lot)	2	3	4	5 (Not at all)							
Question	Q3: How adequate is the level of details to be specified in order to share knowledge?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q4: How understandable is meaning of policy rules for knowledge sharing?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q5: How easy is creation of new policy rule for sharing?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q6: How often did you share knowledge?										
Metrics	<table border="1"> <tr> <td>1 (Never)</td><td>2</td><td>3</td><td>4</td><td>5 (Frequently)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Never)	2	3	4	5 (Frequently)					
1 (Never)	2	3	4	5 (Frequently)							
Question	Q7: Are you aware when the knowledge is shared?										
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table>	YES		NO							
YES		NO									
Question	Q8: How understandable is the access to the shared knowledge?										
Metrics	<table border="1"> <tr> <td>1 (Not at</td><td>2</td><td>3</td><td>4</td><td>5</td></tr> </table>	1 (Not at	2	3	4	5					
1 (Not at	2	3	4	5							

	all)				(Definitely)										
Question	Q9: How often is shared information accessed?														
Metrics	<table border="1"> <tr> <td>1 (Never)</td><td>2</td><td>3</td><td>4</td><td>5 (Frequently)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Never)	2	3	4	5 (Frequently)					
1 (Never)	2	3	4	5 (Frequently)											
Question	Q10: How useful is the shared information?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q11: Does the shared knowledge contain enough information for making decision during coding?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											

Q12: Do you have any suggestion for the improvement of knowledge sharing?

Q13: Do you have any suggestion for the improvement of access to shared knowledge?

Section B.6 KnowBench performance during Knowledge Usage

	Issue	Knowledge use													
	Object	KnowBench System													
	Viewpoint	Developer													
Question	Q1: Does Know Bench provide adequate “search” mechanisms?														
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td><td></td></tr> </table>					YES		NO							
YES		NO													
Question	Q2: How often did you use the search system?														
Metrics	<table border="1"> <tr> <td>1 (Never)</td><td>2</td><td>3</td><td>4</td><td>5 (Frequently)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Never)	2	3	4	5 (Frequently)					
1 (Never)	2	3	4	5 (Frequently)											
Question	Q3: How often do search results satisfy your information needs?														
Metrics	<table border="1"> <tr> <td>1 (Never)</td><td>2</td><td>3</td><td>4</td><td>5 (Frequently)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Never)	2	3	4	5 (Frequently)					
1 (Never)	2	3	4	5 (Frequently)											
Question	Q4: How useful do you find the possibility to search through heterogeneous knowledge sources?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q5: Could you always find that what you searched for?														
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td><td></td></tr> </table>					YES		NO							
YES		NO													
Question	Q6: How often was the list of results too long?														
Metrics	<table border="1"> <tr> <td>1 (Never)</td><td>2</td><td>3</td><td>4</td><td>5 (Frequently)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>					1 (Never)	2	3	4	5 (Frequently)					
1 (Never)	2	3	4	5 (Frequently)											
Question	Q7: Are there a lot of irrelevant results?														
Metrics															

	YES		NO							
Question	Q8: How do you consider the time needed to find results?									
Metrics	<table border="1"> <tr> <td>1 (Not acceptable)</td> <td>2</td> <td>3</td> <td>4</td> <td>5 (Acceptable)</td> </tr> </table>					1 (Not acceptable)	2	3	4	5 (Acceptable)
1 (Not acceptable)	2	3	4	5 (Acceptable)						
Question	Q9: Are you satisfied with the refinement of search?									
Metrics	<table border="1"> <tr> <td>YES</td> <td></td> <td>NO</td> <td></td> <td></td> </tr> </table> If not, please explain why					YES		NO		
YES		NO								
Question	Q10: Are you satisfied with the ranking of search results?									
Metrics	<table border="1"> <tr> <td>YES</td> <td></td> <td>NO</td> <td></td> <td></td> </tr> </table>					YES		NO		
YES		NO								

Section B.7 KnowBench performance during Knowledge Preservation

Goal	Purpose	Improve													
	Issue	Knowledge preservation													
	Object	KnowBench system													
	Viewpoint	Developer													
Question	by KnowBench Q1: Is knowledge provided h always up-to-date?														
Metrics	<table border="1"> <tr> <td>YES</td> <td></td> <td>NO</td> <td></td> <td></td> </tr> </table> If no, please explain					YES		NO							
YES		NO													
Question	Q2: How easy is for a developer to modify a knowledge item?														
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td> <td>2</td> <td>3</td> <td>4</td> <td>5 (Definitely)</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)											
Question	Q3: Can the knowledge item be updated within an														

	acceptable time scale?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q4: How easy is for a developer to delete a knowledge item?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q5: Can the knowledge item be deleted within an acceptable time scale?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Q6: Have you experienced any inconsistency in deleting knowledge items?										
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table> If yes, please explain	YES		NO							
YES		NO									

Section B.8 General Perceptions of KnowBench Performance

Goal	Purpose	Ease												
	Issue	Knowledge management												
	Object	KnowBench system												
	Viewpoint	Developer												
Question	Q1: Was KnowBench simple to use in performing the daily tasks?													
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>				1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)										

	If "Not at all", please explain										
Question	Do you think the KnowBench trial was useful for the acceptance of the KnowBench system? Q2:										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	KnowBench caused any problems in your daily routine activities? Q3: Has										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Do you find the KnowBench system useful for accomplishing your tasks? Q4:										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table>	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Did you notice any improvements compared to the way you were performing your job previously? Q5:										
Metrics	<table border="1"> <tr> <td>YES</td><td></td><td>NO</td><td></td></tr> </table> If yes, please enumerate them. If not, please explain	YES		NO							
YES		NO									
Question	Do you feel that you were able to resolve problems that occurred during coding faster/better with KnowBench? Q6:										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table> If "Not at all", please explain	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							
Question	Do you feel that at some point the KnowBench system was overloading you with information, thus delaying										

	the execution of your tasks?										
Metrics	<table border="1"> <tr> <td>1 (Definitely)</td><td>2</td><td>3</td><td>4</td><td>5 (Not at all)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table> If "Definitely", please give an example	1 (Definitely)	2	3	4	5 (Not at all)					
1 (Definitely)	2	3	4	5 (Not at all)							
Question	Q8: Do you think KnowBench is capable of improving your overall working experience?										
Metrics	<table border="1"> <tr> <td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td></tr> </table> If "Not at all", please explain	1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)							

Q9: Do you have any improvement suggestions for KnowBench? What additional functionalities or services would be of greatest value to add to KnowBench?

Section B.9 System-specific Evaluation

1. Usability

Goal	Purpose	Maximize				
	Issue	User experience				
	Object	KnowBench system				
	Viewpoint	Developers				
Question		Q1: Does the Graphical User Interface (GUI) behave as the users expect?				
Metrics						
		1 (Not at all)	2	3	4	5 (Definitely)
Question		Q2: How easy and intuitive is the use of the GUI?				
Metrics						
		1 (Not at all)	2	3	4	5 (Definitely)

Question	Q3: Does the system allow the user to select data from predefined lists, where appropriate?				
Metrics					
	1 (Not at all)	2	3	4	5 (Definitely)
Question	Q4: Can the user easily understand the messages displayed by the system (e.g. verification or error messages)?				
Metrics					
	1 (Not at all)	2	3	4	5 (Definitely)
Question	Q5: Do the users perceive awkward/redundant the interactions with the KnowBench system?				
Metrics					
	1 (Not at all)	2	3	4	5 (Definitely)
Question	Q6: Does the user know where he is at all times, how he got there, and how to get back to the point from which he started?				
Metrics					
	1 (Not at all)	2	3	4	5 (Definitely)
Question	Q7: How simple and easy is the navigation in the system's functions?				
Metrics					
	1 (Not at all)	2	3	4	5 (Definitely)
Question	Q8: How attractive is the appearance of the system (design, aesthetics, colours, etc.) to the user?				
Metrics					
	1 (Not at all)	2	3	4	5

	all)				(Definitely)

2. Reliability

Goal	Purpose	Minimize			
	Issue	System failure			
	Object	KnowBench System			
	Viewpoint	System			
Question		Q1: How frequently does the system fail in operation?			
Metrics		1 (Never)	2	3	4
					5 (Frequently)

3. Scalability & availability

Goal	Purpose	Ensure			
	Issue	Scalability & availability of the system			
	Object	KnowBench System			
	Viewpoint	System			
Question		Q1: Does the system behave normally when the size of knowledge base increased?			
Metrics		1 (Not at all)	2	3	4
					5 (Definitely)
Question		Q2: Is the response time of the system acceptable?			
Metrics		1 (Not at all)	2	3	4
					5 (Definitely)
Question		Q3: Does the system require a lot of memory?			
Metrics		1 (Not at all)	2	3	4
					5 (Definitely)

Question	Q4: Does the system occupy a significant size of hard disk space?					
Metrics	1 (Not at all)	2	3	4	5 (Definitely)	
Question	Q5: Does the system introduce a significant raise in network traffic?					
Metrics	1 (Not at all)	2	3	4	5 (Definitely)	

4. Security

Goal	Purpose	Ensure														
	Issue	Security of the system														
	Object	KnowBench System														
	Viewpoint	System														
Question		Q1: Is access to shared information restricted to users with appropriate rights?														
Metrics		<table><tr><td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)												
Question		Q2: Does the system provide adequate protection of shared knowledge (e.g. encryption etc.)?														
Metrics		<table><tr><td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)												

5. Deployment

Goal	Purpose	Ensure														
	Issue	Deployment														
	Object	KnowBench System														
	Viewpoint	Developer														
Question		Q1: How easy is it to install the KnowBench system?														
Metrics		<table><tr><td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)												
Question		Q2: How easy is it to un-install the KnowBench system?														
Metrics		<table><tr><td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)												
Question		Q3: How easy is it to repeat setup operation in the case that an error occurred?														
Metrics		<table><tr><td>1 (Not at all)</td><td>2</td><td>3</td><td>4</td><td>5 (Definitely)</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>					1 (Not at all)	2	3	4	5 (Definitely)					
1 (Not at all)	2	3	4	5 (Definitely)												