Elicitation, Representation and Management of Software Design Knowledge

Behrouz H. Far and Mohsen Afsharchi
Department of Electrical & Computer Engineering, University of Calgary
{far, mafsharc}@ucalgary.ca

Abstract

In this research we focus on understanding the nature of the knowledge used during the various phases of the software development process. We have found that there are two types of knowledge involved in software development: (1) descriptive knowledge represented by conversion and coding rules, e.g., a rule for splitting a class into two; and (2) prescriptive knowledge required for deployment of global or local strategies at a micro design level; e.g., knowledge required to answer the question "why should a class should be split into two?" Most of the already existing knowledge management solutions address descriptive knowledge. Elicitation and management of the prescriptive knowledge is difficult in the sense that it is probabilistic, personalized, distributed and context specific. Also we have found that prescriptive knowledge tends to be used in decision making processes involving multiple stakeholders with different perspectives (e.g., designer, tester, software architect and project manager). We also report on a prototype system called ISS-OKM to extract and reuse both the descriptive and prescriptive knowledge.

1. Introduction

Today's software industry is characterized by shorter product lifecycles, faster delivery, multiplicity of development technologies, increased employee turnover, physical and logical distribution of assets and ubiquitous information technologies. A software organization's ability to manage its knowledge assets, including humans, is a major source of competitive advantage. This research is devoted to integrating concepts and approaches from various disciplines, including software engineering (SE), decision support, ontology-based knowledge management, semantic integration and computational intelligence in order to provide optimal and unified solutions for the current challenges in knowledge management applications.

Software Engineering's knowledge is dynamic and evolves with technology, organizational culture and the changing needs of an organization's development practices. There are two viewpoints related to Knowledge management (KM) in the software industry: (1) information processing view, which has been widely implemented; e.g., various flavors of the Experience Factory (EF) [1], sees KM as archiving explicit knowledge of individuals in technology-based repositories, and (2) human-centric view, which incorporates social and individual dimensions into KM [2]. Kess et al. argue that software processes are essentially knowledge processes, structured within a KM framework [3]. Conventional KM solutions are mainly based on a centralized architecture commonly comprised of a central knowledge repository accessible through formalized queries. Experience shows that such architecture cannot be utilized effectively in real life. Basili et al. acknowledge that for an organization to implement the EF approach, a number of potential barriers to success, such as the need to capture and distribute knowledge quickly, must be overcame [1].

The core requirements for KM solutions in SE are [4]: (1) Incorporate the management of knowledge assets, which are distributed and belong to both people and departments; (2) allow for dynamic classification and distribution of knowledge; (3) allow for adapting to diversified contents, representation and personalized styles; (4) incorporate efficient (i.e., fast and effortless) retrieval mechanisms; and (5) facilitate interaction between distributed knowledge bases in order to support social process of knowledge management. There is an obvious need to investigate, design and implement such a KM solution for SE.

In this research we investigate how to empower distributed knowledge management with ontology-based knowledge management, peer-to-peer architectures and software agent technology in order to build an orchestrated knowledge management (OKM) solution to manage intellectual assets for software development organizations.

2. Related Works

Research in the following areas is particularly important to this work:

2.1 Software design knowledge

Object-Oriented Analysis and Design (OOAD) has become a very popular software development approach since the 1990's. Object elicitation and class modeling are among the central activities in OOAD. The objects are identified from the requirements, and the class model is generated based on them as well. Generally, there are two ways to specify the requirements: using formal languages or using natural languages (NL). The research community has focused on methods based on formal language requirements [5,6,7], while NL is widely used for requirements documentation in industry. It is hard to automate NL requirements analysis, because NL is inherently complex, vague and ambiguous [8].

Most of the commercial CASE (Computer Aided Software Engineering) tools do not supply the functionality of NL requirements analysis. However, there are several such tools that have been developed for research. CoGenTex Inc. developed a prototype tool named LIDA (Linguistic assistant for Domain Analysis), which provides linguistic assistance in model development [9]. The tool can process textual documents and help the user to generate a class model visualized in UML (Unified Modeling Language). NIBA (Natural Language Requirements Analysis in German) is an interdisciplinary project between computer scientists and computer linguists at the University of Klagenfurt, Austria [10]. The tool can parse requirements documents in German, interpret and transform output of the parser to conceptual pre-design schemas, validate the schemas and finally generate a conceptual model in UML. These approaches only generate the conceptual model, but the behavior of classes still need to be identified separately. So far, it is impossible for machines to automatically perform the whole OOAD process, however it is possible to automate some micro-activities in OOAD [11].

We have found that there are two types of knowledge involved in software design: (1) descriptive knowledge represented by conversion and coding rules, e.g., a rule for splitting a class into two; and (2) prescriptive knowledge required for deployment of global or local strategies at a micro design level; e.g., knowledge required to answer the question "why should a class should be split into two?" Most of the already existing knowledge management solutions address descriptive knowledge (e.g., Experience Factory). Elicitation and management of the prescriptive knowledge is difficult in the sense that it is probabilistic, personalized, distributed and context specific. Also we have found that prescriptive knowledge tends to be used in decision making processes involving multiple stakeholders with different perspectives (e.g., designer, test engineer, software architect and project manager).

2.2 Distributed architecture for KM systems

Nowadays, there is an increasing interest in the use of peer-to-peer and multi-agent concepts in KM, mainly motivated by the fact that KM domains involve an inherent distribution of resources, problem solving capabilities and responsibilities [12,13]. That is, the integrity of the existing organizational structures and the autonomy of participants must be maintained, which calls for an autonomous and distributed representation of KM systems [14]. The use of shared representational ontologies has been questioned in [15] and a distributed architecture based on explicitly distributed ontologies has been proposed in [16]. We have discussed enabling technologies and the research trends from Web-based centralized KM to the distributed agent mediated knowledge management in [4]. Other projects that address these aspects are: COMMA [12], FRODO [17] and EDAMOK [16].

2.3 Ontology-based knowledge management

The term "ontology" was first used in conjunction with knowledge sharing and reuse by [18] and since then it has been extensively used in knowledge management research [17]. A general architecture for ontology-based knowledge management has been proposed in [19]. In the majority of KM applications, ontologies are typically used for three purposes: (1) to support knowledge visualization, where ontologies are inspected in order to create new knowledge by analysis and recombination of existing knowledge [20]; (2) to support knowledge search, retrieval and personalization, where ontologies are used to improve search and retrieval of information by exploitation of ontological background knowledge about the application domain; and (3) to serve as the basis for information gathering and integration, where some degree of formality of ontologies allows partial automation of problem solving and integration of information retrieval into business application [16]. Using ontologies for managing software experiences has been studied in [21].

2.4 Semantic integration

Different software systems may use individualized conceptualizations of a certain domain. To achieve ontology-based semantic integration, two software systems (or agents) must find a way to share the semantics of the terms in their ontologies, this can be done in several possible directions: (1) using a single centralized global ontology- a single centralized global ontology is defined for the application domain- and all agents or computer programs in communication use terms from this ontology; (2) merging source ontologies into a unified ontology: ontologies defined on a common domain by different applications have lots of overlap, therefore merging the source ontologies into one unified ontology before agent interactions is a way to fulfill semantic integration [22]; (3) searching a set of mappings (or matches) between two ontologies: instead of trying to merge two source ontologies, finding a set of mapping rules between them is an alternative way to achieve semantic integration [22,23]; (4) runtime ontology resolution: for a multiagent system, agents are often from different heterogeneous environments, it is impractical to restrict all agents to use a single ontology or to have ontology merging, matching, and translation services available prior to the deployment of the agent system. A better way is to resolve semantic differences when they arise during run-time interaction [24]. In a multiagent environment, agents may often want to maintain their own diverse ontology but still be able to identify when they are referring to the same concept. This allows for each agent to maintain control over its own ontology but still be able to communicate with the others without first converging to a common ontology, as in [25].

2.5 Semantic Web

There are two attempts to convert information into machine-processable knowledge. (1) Intelligent data preprocessing [26]: these techniques try to extract knowledge from mainly textual documents. Some of the techniques are Web-mining techniques, including Web link structure mining, Web content mining and Web log mining. (2) Semantic Web [27,28,19]: this method tries to inject machine-understandable knowledge into documents by enriching the documents semantically and has been developed using XML. Although XML has had great impact on the software industry, it only specifies syntactic conventions and any intended semantics are outside the domain of the XML specification. The Resource Description Framework (RDF) is a recent W3C recommendation designed to standardize the definition and use of metadata descriptions of Web-based resources. As with XML, the RDF data model provides no mechanisms for declaring property names that are to be used. RDF Schema (RDFS) takes a step further towards a richer representation formalism and introduces basic ontological modeling primitives. RDFS allows definition of classes, subclasses, properties, sub-properties, domain and range restrictions of the properties of concepts. RDFS provides a standard syntax for writing ontologies and a standard set for modeling primitives, such as instance-of relationships. Although RDFS can be regarded as an ontology language, there are many knowledge elements that cannot be expressed using RDFS. DARPA has released the DARPA Agent Markup Language (DAML) (http://www.DAML.org), a simple language for expressing more sophisticated class definitions than those permitted by RDFS. The DAML group joined efforts with the Ontology Inference Layer (OIL) [29], another effort providing more sophisticated classification using constructs from frame-based artificial intelligence. The result of these efforts is DAML+OIL, a language for expressing sophisticated classifications and properties of resources. Another variation of DAML+OIL, pursued by W3C is the Web Ontology Language (OWL) which is based on RDFS (http://www.w3.org/TR/owl-features).

2.6 Agent mediated knowledge management

Agent-based software development combines and builds on various computing technologies, including object-orientation, parallel processing, distributed computing, and mobile code. The principles and perspectives are abstracted from artificial intelligence, biology, system science and mathematics. Agent Mediated Knowledge Management (AMKM) has been an active research area in the software agent research community in recent years [14]. It uses agent concepts to analyze and model organizations and their knowledge needs and to provide a reusable architecture to build KM systems. AMKM is different from peer-to-peer KM in the sense that (1) agents can learn and adapt themselves to changes in environment; and (2) agents can get involved in complex interactions. These two points come from the fact that agents are autonomous and social entities.

3. Representation and reuse of software descriptive knowledge

We have developed a method for use-case model generation, object identification and class modeling with respect to natural language requirements based on the Rational Unified Process (RUP). Use-case language schemas are proposed to reduce complexity and vagueness of natural language. Some rules are identified and used to automate class model generation from use-case specifications. A CASE tool named Use-Case driven Development Assistant (UCDA) is implemented to support the methodology. UCDA can assist the developer to generate use-case diagrams, use-case specifications, robustness diagrams, collaboration diagrams and class diagrams in IBM Rational Rose. Another version of the same tool has been developed for Eclipse [30]. They both help accelerate requirements analysis and class modeling, and reduce the time to market in software development.

3.1 Methodology

Based on the Rational Unified Process (RUP), the activities and corresponding artifacts during requirements, analysis and design are specified as follows:

- Identify actors and use cases from stakeholder requests.
- Structure the use cases into use-case diagrams.
- Generate the use-case specifications.
- Review the use-case specifications.
- Analyze the use-case specifications and generate the analysis model.
- Review the analysis model.
- Generate the design model based on the analysis model.

The whole process is divided into two parts based on different concerns. The first part addresses NL requirements analysis and use-case modeling. The second part is concerned with the use-case realization and class model generation. The artifacts and activities in the process are shown in Figure 1. The output of the requirements phase is a use-case model. Use cases are means to capture the contracts between the stakeholders of a system and its behavior [5]. A use-case model comprises diagrams in UML and specifications that record sequences of actions that a system can perform by interacting with outside actors.

3.2 Implementation

To implement the methodology, we develop a CASE (Computer Aided Software Engineering) tool named UCDA (Use-Case driven Development Assistant) composed of two parts: the first part for NL requirements processing and use-case modeling, and the other for use-case realization and class model generation. The tool is integrated seamlessly with IBM Rational Rose. The user can manage the tool with Rational Rose's add-in manager. Most artifacts generated by UCDA are represented in XML and visualized in Rose. Another version has also been developed for Eclipse.

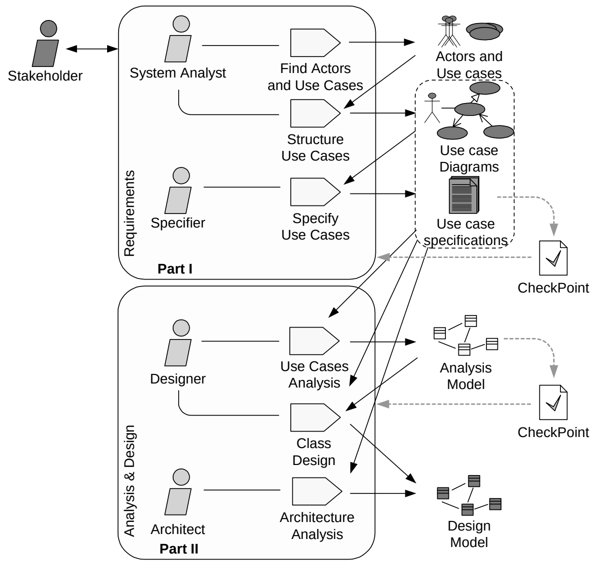


Figure 1. UCDA system architecture

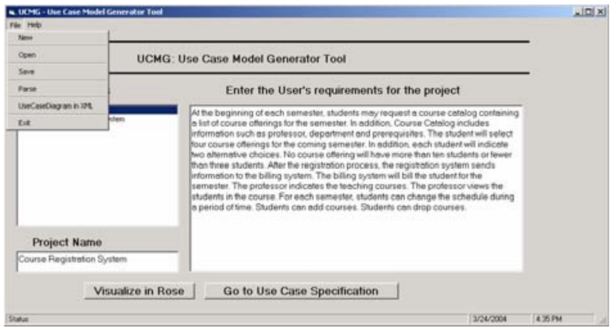
The features currently implemented are as follows:

1. Parse the NL requirements and identify actors and use cases, and then generate the use-case diagram in Rational Rose.
2. Assist the user to finish use-case specification.
3. Realize the use cases, identify the classes, and generate robustness diagrams and collaboration diagrams in Rational Rose.
4. Validate the analysis class model via robustness diagrams.
5. Generate the class model in Rational Rose.

Not all the activities shall be fully automated. The user needs to interact with UCDA to supply the necessary information, and the tool will help the user to develop a model in UML for further revision.

3.3 Use-Case Modeling Environment

When the user has only the requests, s/he can start to analyze with UCDA. Figure 2 is the environment for requirements parsing. The user needs to paste or edit the requests of a project in it. Then UCDA can help the user identify the use cases from the request and generate the use-case diagram in Rational Rose.


UCMG - Use Case Model Generator Tool
File Help
New
Open
Save
Parse
UseCaseDiagram in XML
Exit
UCMG: Use Case Model Generator Tool
Enter the User's requirements for the project
Project Name
Visualize in Rose
Go to Use Case Specification
Status
3/24/2004
4:35 PM

At the beginning of each semester, students may request a course catalog containing a list of course offerings for the semester. In addition, Course Catalog includes information such as professor, department and prerequisites. The student will select four course offerings for the coming semester. In addition, each student will indicate two alternative choices. No course offering will have more than ten students or fewer than three students. After the registration process, the registration system sends information to the billing system. The billing system will bill the student for the semester. The professor indicates the teaching courses. The professor views the students in the course. For each semester, students can change the schedule during a period of time. Students can add courses. Students can drop courses.

Course Registration System

Figure 2. NL parsing and use-case identification

Then the user can specify the use cases with the assistance of UCDA. UCDA parses the user's input information and normalizes it based on use-case language schemas. The structure of each statement in the flow of events is identified, and all statements are encoded in XML. An example use-case specification with XML markups removed is given below.

Actors: customer, bank
Flow of Events:
Basic Flow:

1. the system starts withdrawal transaction;
2. the customer selects the account on the customer console;
3. the system gets the account from the customer console;
4. the customer selects the amount on the customer console;
5. the system gets the amount from the customer console;
6. the system generates the withdrawal transaction information;
7. the system sends the withdrawal transaction information to the network connection;
8. the bank gets the withdrawal transaction information from the network connection;
9. the bank sends the withdrawal transaction approval to the network connection;
10. the system gets the withdrawal transaction approval from the network connection;
11. the system dispenses the cash in the cash dispenser;
12. the customer gets the cash from the cash dispenser;
13. the system records the withdrawal transaction information into the log;
14. the withdrawal transaction end;

Alternative Flow:

If the bank does not approve the withdrawal transaction, then

i. the system displays error message on the customer console;
ii. the system records the withdrawal transaction information into the log;
iii. the withdrawal transaction end;

3.4 Use-Case Realization Environment

When the use-case model is ready, the user can use UCDA to realize the use cases and generate the class model. All the diagrams generated by the tool are visualized in Rational Rose. The environment for use-case realization is shown in Figure 3. The user can set the glossary and select a use case to realize. When collaboration diagrams are generated, the tool can distribute the behavior and generate the class model in Rational Rose.

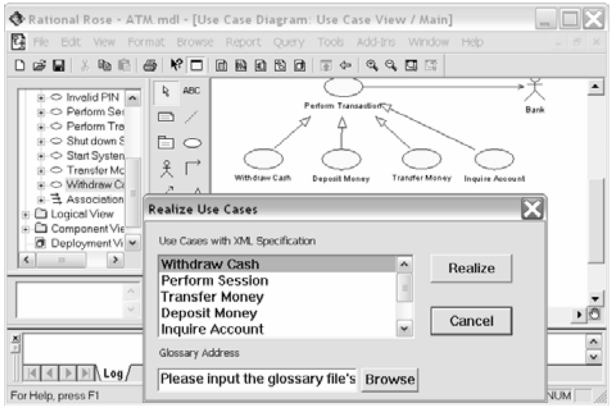


Figure 3. Use case realization within the Rose environment

The robustness diagram generated by UCDA based on the example specification is shown in Figure 4.

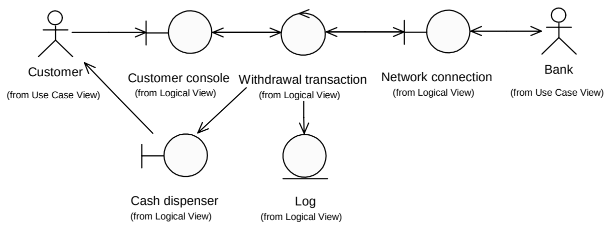


Figure 4. Robustness diagram generated by UCDA

4. Representation and reuse of software perspective knowledge

Emergence of distributed knowledge management in organizational environments and the Semantic Web in the World Wide Web which allow diversity of ontologies, make it necessary for information retrieval to be managed by intelligent software programs. These software programs (i.e. Agents) should be able to use machine learning techniques to improve information retrieval efficiency. They also should be able to know each other and know who knows what and query and learn concept from each other. In addition, they should be able to negotiate with each other to provide the best response to a query.

4.1 Methodology

Figure 5 shows a typical software organization with several departments each having their own concept structure and each maintaining their own knowledge resources.

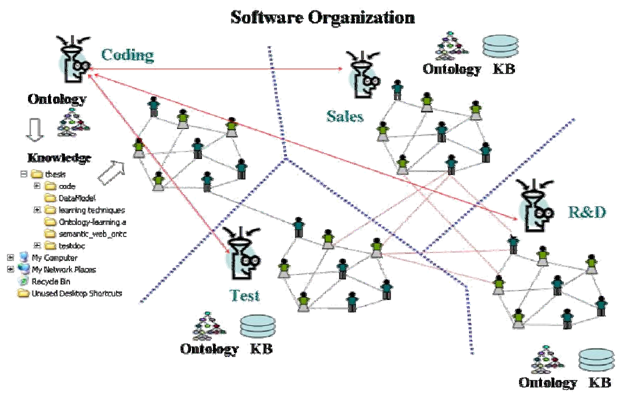


Figure 5. Software engineering organization

For each community within the organization a software agent (or a coalition of software agents) is responsible for eliciting, organizing, maintaining and sharing information. Ontology in our agents is a mix of meta-concepts and fine grained hierarchical concept structures. The agents use a common core ontology which could be the whole of meta-concept level or part of it. We specifically mention that ontologies of our agents in part of meta-concept level and also in fine grained hierarchical concept structure are diverse.

Figure 6 shows an example of ontological structure for an agent which is deployed in a software engineering knowledge management environment. Meta-concepts in this environment are concepts that are not directly related to examples (e.g. documents). For example in a software engineering knowledge management environment, software is a very general concept and it should be divided as sub-concepts to increase knowledge management efficiency. Meta-concepts are meta-knowledge about a given domain. We consider meta-concepts and their correlation as core ontology for our multi-agent system.

Fine grained concepts are concepts which are directly connected to examples. We allow diversity of ontologies at this level. As depicted in Figure 7, a lattice structure represents "measurement". This lattice contains a tree sub-concept of "software measurement" which are directly pointing to corresponding lattices. Based on the application in which the agents are deployed, the granularity of concept structure may differ. For example in Figure 6 the Design node could directly point to a lattice or it could be divided more to other sub-concepts which are pointing to their corresponding lattices. The agents should be able to divide a concept to sub-concepts to avoid lattice complexity.

We believe that this conceptual structure is flexible (as opposed to the methods such as Experience Factory) and yet is well suited to information sharing and retrieval.

Suppose this agent is queried by another agent "do you know anything about software measurement?" Answering this query, agent locates concept nodes for these words and following downward in lattice to find related documents and features and send them back. "I suggest you to include engineering in your query", "If you exclude 'measurement' from query I can give you more examples" or "All of my examples which have 'measurement' also have 'software' in them" are some examples of information that agent could send to querying agent in the case of negotiation.

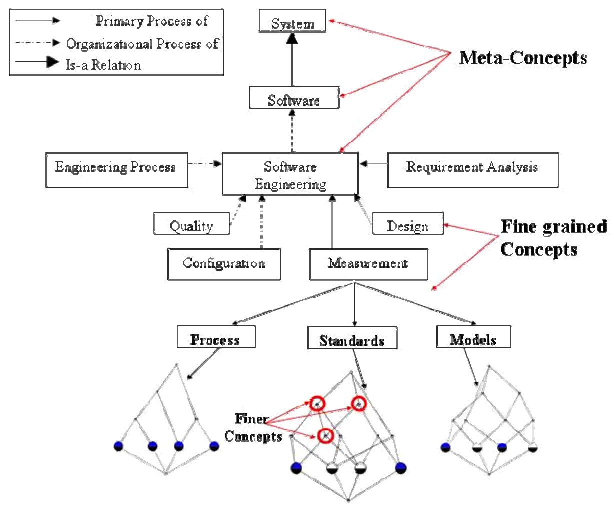


Figure 6. Concept hierarchy

Each software agent responsible for a domain (or a work space) must learn (1) from the existing chunks of information added to its repository; and/or (2) from the other agents who share partial representation of the domain. In the following subsections we present the individualized and collaborative concept learning mechanisms handling these two problems, respectively.

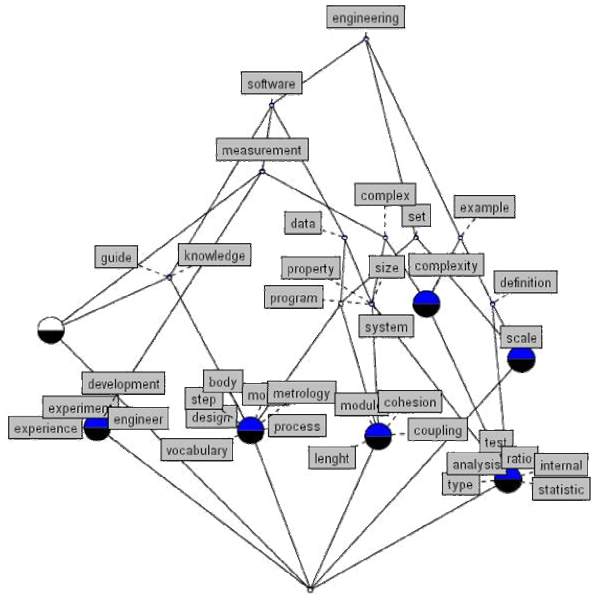


Figure 7. Lattice of concepts

4.2 Individual concept learning mechanism

Figure 8 shows the concept formation process for an individual agent. For each concept or set of concepts our agents have some objects (i.e., documents) and features (e.g. bag of words) representing them. Using rules and algorithms of formal concept analysis, the agent builds a formal context and its corresponding concept lattice. This structure can be gradually improved when new objects and features become available. An important point here is that the formation of formal context is both automated and supervised. This means that higher level concepts in the concept lattice are generated automatically but can explicitly be labeled to show the name of that concept by the supervisor.

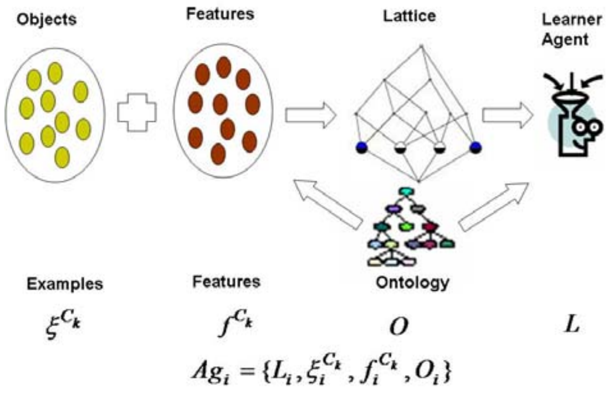


Figure 8. Individual concept learning mechanism

4.3 Collaborative concept learning mechanism

In collaborative learning, the goal of the learner agent is to learn a concept from the other agents. For the learner agent, a major problem is how to figures out that it does not know a concept. The simplest method to handle this is tracking the incoming queries. When a learner agent has failed to answer some queries, it tries to find coherence among previous unanswered queries. Using this coherence and the elements of incoming queries the learner agent makes a new query and submits it to a set of teacher agents to find out about the probable concept.

Based on our agent model, the teacher agents possess examples and features regarding a certain concept and they can also judge whether an example belongs to a concept or not. Therefore they can support learning agents with examples and features and also they can answer learning agents' questions regarding classification of a certain example. As stated previously, the flexible hierarchical structure of lattice lets teacher agents to traverse their concept structure from the concept toward its examples and features or to traverse the concept structure from features to concepts. This flexibility can help teacher agents to answer the learner agent queries based on features or the concept name.

Collaboration in this method is a modified version of weighted majority voting. Here the learner agent uses advice of teacher agents to evaluate examples received and weight their evaluation utilizing a distance function. In the meantime learner agent uses Formal Concept Analysis (FCA) and considers examples as objects and makes a formal context using selected examples and features. Then it locates the most situated place in its ontological structure to place the concept lattice. In the best case the learner agent can construct a formal concept structure for a concept which includes all examples and features.

5. Conclusions

In this paper a novel methodology for elicitation, representation and management of knowledge for software development organizations was presented. This method integrates both the descriptive and perspective knowledge was presented. This methodology has several advantages over the existing ones, including ease of elicitation, representation and management of organizational knowledge.

Acknowledgements.
Several former students of the Intelligent Software Systems (ISS) Laboratory, Faculty of Engineering,

University of Calgary, have contributed to this project and the authors acknowledge their effort and thank them collectively.

6. References

[1] Basili, V., Costa, P., Lindvall, M., Mendonca, M., Seaman, C., "An Experience Management System for a Software Engineering Research Organization," Proc. 26th Annual NASA Goddard Software Engineering Workshop, pp. 29-35, 2002.

[2] Malhotra, Y., "Knowledge management and new organization forms: A framework for business model innovation," Information Resources Management Journal, vol. 13, no. 1, pp. 5-14, 2000.

[3] Kess, P., Haapasalo, H., "Knowledge Creation through a Project Review Process in Software Production," Intl. J. of Production Economics, vol. 80, no. 1, pp. 49-55, 2002.

[4] Afsharchi M., B.H. Far, "Knowledge Orchestration Agency: Knowledge Management Using Intelligent Software Agents," Decision Support in Agent Mediated Environments, Chapter 3, pp. 71-89, G. Phillips-Wren and L. Jain (Edts.), IOS Press B.V., 2005.

[5] Cockburn, A., "Writing effective use cases," Addison-Wesley, 2000.

[6] Bois, P. D., Dubois, E., Zeippen, J.M., "On the use of a formal RE language-the generalized railroad crossing problem," Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, Annapolis MD, pp. 128-137, 1997.

[7] Li, X., Liu, Z., He, J., "Formal and use-case driven requirement analysis in UML," 25th Annual International Computer Software and Applications Conference, COMPSAC2001 Chicago, pp. 215-224, 2001.

[8] Boyd, N., "Using natural language in software development," Journal of Object-Oriented Programming vol. 11, no. 9, pp. 45-55, 1999.

[9] Overmyer, Scott P., Lavoie, B., Rambow, O., "Conceptual modeling through linguistic analysis using LIDA," Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, Toronto, pp. 401-410, 2001.

[10] Niba, L.C., "The NIBA workflow: From textual requirements specifications to UML-schemata," Proceedings of International Conference on Software & Systems Engineering and their Applications, ICSSEA 2002, Paris, 2002.

[11] Liu, D., "Automating transition from use cases to class model," MSc Thesis, University of Calgary, 2003.

[12] Gandon, F., Dieng, R., Corby, O., Giboin, A., "A Multiagent System to Support Exploiting an XML-based Corporate Memory," Proc. PAKM'00, Basel, 2000.

[13] Susarla, A., Liu, D., and Winston, A., "Peer-to-Peer Knowledge Management," Handbook of Knowledge Management, vol. 2, ch. 39, 2002.

[14] Dignum, V., "Using Agents to Support Knowledge Sharing," Proc. Workshop on Autonomy, Delegation and Control, AAMAS'03, Melbourne, 2003.

[15] Wang, J., and Gasser. L., "Mutual Online Concept Learning for Multiple Agents," Proc. AAMAS'02, Bologna, Italy, pp. 362-369, 2002.

[16] Bonifacio, M., Bouquet, P., Manzardo, A., "A Distributed Intelligence Paradigm for Knowledge Management, Bringing Knowledge to the Business Process," Proc. AAAI Spring Symposium, Technical Report SS-00-03, AAAI Press, 2000.

[17] Abecker A., van Elst, L., "Ontologies for Knowledge Management," in Handbook on Ontologies, S. Staab and R. Studer (eds.), Ch. 22, pp. 435-454, Springer, 2003.

[18] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., Swartout, W., "Enabling Technology for Knowledge Sharing," AI magazine, vol. 12, no. 3, pp. 36-56, 1991.

[19] Fensel D., "Ontology-based Knowledge Management," IEEE Computer, vol. 35, no. 11, 2002.

[20] Kumar, V., Furuta, R., and Allen, R.B., "Metadata Visualization for Digital Libraries: Interactive Timeline Editing and Review," ACM Digital Libraries, pp. 126-133, 1998.

[21] Nour P., "Ontology-based Retrieval of Software Engineering Experiences," MSc thesis, Department of Computer Science, University of Calgary 2003.

[22] Noy, N.F., Musen, M.A., "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment." Proc. 17th National Conf. on Artificial Intelligence (AAAI-2000), Austin, TX, 2000.

[23] Doan, A., Madhavan, J., Domingos, P., Halevy, A., "Ontology Matching: A Machine Learning Approach," Handbook on Ontologies in Information Systems, S. Staab and R. Studer (eds.), pp. 397-416, Springer, 2004.

[24] Wiesman, F., Roos, N., Vogt, P., "Automatic Ontology Mapping for Agent Communication. Technical Report, MERIT, 2001.

[25] Steels, L., "The Origins of Ontologies and Communication Conventions in Multi-agent Systems," J. Autonomous Agents and Multi-Agent Systems, vol. 1, no. 2, pp. 169-194, Kluwer, 1998.

[26] Han J., Chang K., "Data mining for Web Intelligence," in Web Intelligence, Springer Monograph 2003.

[27] Martin P., "Knowledge Representation, Sharing, and Retrieval on the Web," in Web Intelligence, Springer, 2003.

[28] Sure Y., "On-To-Knowledge: Ontology-based Knowledge Management Tools and their Application," in: German Journal (KI), vol. 1, pages 35-37, 2002.

[29] Bechhofer S., "An informal description of Standard OIL and Instance OIL," Ontology Inference Layer OIL Whitepaper, 2000.

[30] Chang J., P. Chwiecko, M. Doan, and L. Ko, "Eclipse Plug-in for Automation of Requirements Analysis, Modeling and Code Generation for Object-Oriented Software Design," Proceedings of the 2nd Symposium on Electrical, Computer and Software Engineering, University of Calgary, 2005.