



ELSEVIER

Knowledge-Based Systems 15 (2002) 465–471

Knowledge-Based
SYSTEMS

www.elsevier.com/locate/knosys

Knowledge flow management for distributed team software development

Hai Zhuge*

Knowledge Grid Research Group, Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, People's Republic of China

Received 19 April 2001; accepted 22 November 2001

Abstract

Cognitive cooperation is often neglected in current team software development processes. This issue becomes more important than ever when team members are globally distributed. This paper presents a notion of knowledge flow and the related management mechanism for realizing an ordered knowledge sharing and cognitive cooperation in a geographically distributed team software development process. The knowledge flow can carry and accumulate knowledge when it goes through from one team member to another. The coordination between the knowledge flow process and the workflow process of a development team provides a new way to improve traditional team software development processes. A knowledge grid platform has been implemented to support the knowledge flow management across the Internet. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Cognitive cooperation; Knowledge flow; Knowledge grid; Team development; Workflow

1. Introduction

Distributed team software development is a kind of software development management paradigm that focuses on work cooperation and resource sharing among the geographically distributed team members during the development process. Such a sharing needs the support of a distributed and across-platform computing environment.

A team development environment usually needs to support such functions as version management, build management, project management, parallel development, team coordination, and process management. Besides sharing codes and documents, a team development environment should further enable team members to share technical skills with each other. The suggestion of establishing IT skill standard was proposed in Ref. [8].

A workflow is a mechanism that supports work cooperation among team members according to definite process logic. A workflow management system (WfMS) is a system that completely defines, manages and executes the workflow specification through the execution of software whose execution order is driven by a computer representation of the workflow logic [4,5,10]. As a high-level tool, a workflow can be used to integrate the distributed and heterogeneous application processes into a unified process

with the support of the low-level distributed object component techniques. The characteristics of time modeling, reusability, exception handling, and formal model of a workflow have been discussed [1,2,6,7,12]. These characteristics can be incorporated into the WfMS for supporting a distributed team development. But the current workflow model and the WfMS do not concern the knowledge-level cooperation. In fact, the implementation of each task of a workflow is an inter-operation between human and the support work environment. Human cognitive ability is an important impact factor of effectively accomplishing the development tasks.

Team software development can be regarded as a cooperative human problem-solving process with the support environment. Such cooperation exists at both the knowledge level and the work level. At the knowledge level, people can make abstractions and analogies between problems, and use past experiences and skills to solve new problems [3,11]. Learning from human problem-solving processes is an important way to improve the existing team software development processes [11]. Unfortunately, the current research works on team software development only focus on the system-level improvements. Human cognitive characteristics are seldom addressed.

The current Internet technology enables a development team to be globally distributed, but the globalisation causes not only the rising of the communication cost between team

* Fax: +86-1062567724.

E-mail address: zhuge@ict.ac.cn (H. Zhuge).

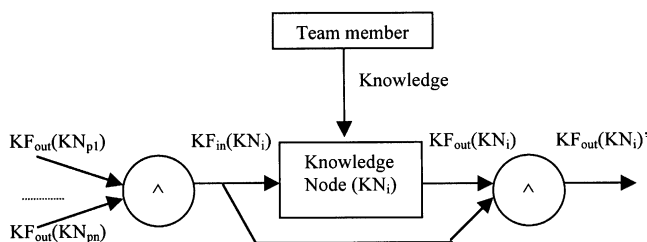


Fig. 1. Input and output knowledge flow of a knowledge node.

members but also the increasing of changing partners [9]. The member change of a development team causes different experienced members to cooperate with each other in the same development process. These new challenges require a distributed team development environment to support the cognitive cooperation.

2. Knowledge flow: coordinating cognitive cooperation among team members

A knowledge flow is a carrier of human knowledge, which passes a team member's knowledge to the succeeding team member (receiver) according to a definite process logic, shares its content with the receiver, and accumulates knowledge of the team members. A complete knowledge flow passing through all team members during a team development process constitutes a knowledge flow network (KFN), which is a mapping image (e.g. an isomorphism) of the workflow process the team is working with. Each node of the KFN, called knowledge node, represents the generation of the corresponding team member's knowledge during his/her task implementation process. The output of a knowledge node is a knowledge flow that depends on the corresponding team member's cognitive ability and the input knowledge flow (i.e. the experience used for reference). Usually, we call a knowledge node active only when the corresponding team member is working on it. Otherwise, the knowledge node is inactive. An inactive knowledge node can be re-active again when the corresponding team member is working on it again according to the process logic of the KFN. The propagation of active node(s) on the KFN usually keeps pace with the execution of the corresponding workflow process.

The knowledge flow can accumulate knowledge generated by previous knowledge nodes during its passing process. Fig. 1 graphically describes the constitution of the input and output knowledge flows of a knowledge node (KN_i). The input knowledge flow is the 'and-join' of the output knowledge flows of its predecessors, represented as $KF_{in}(KN_i) = KF_{out}(KN_{p1}) \wedge \dots \wedge KF_{out}(KN_{pm})$. The final output of the KN_i , $KF_{out}(KN_i)'$ is constituted by the 'and-join' of the input $KF_{in}(KN_i)$ and the output $KF_{out}(KN_i)$, represented as $KF_{out}(KN_i)' = KF_{out}(KN_i) \wedge KF_{in}(KN_i)$. The 'and-join' of knowledge flows represents both the logical

'and' relationship between flows and the content union of the flows.

For a newly created KFN, the knowledge input of the first knowledge node $KF_{in}(KN_0)$ will be initialised by the cooperation rules of the development team. After the first run of the KFN, the output of the end knowledge node of the last run will be the input of the first knowledge node of the current run.

There are two major differences between the knowledge flow and the workflow. First, the knowledge flow content is generated from the team members' task implementation process during the execution of the workflow process and cannot be pre-designed. While, a workflow reflects domain business and is pre-designed by its designer. Second, the knowledge flow carries knowledge of team members, while a workflow reflects either the data dependence relationship or the logical execution dependence relationship between tasks.

3. Knowledge flow representation

The representation of a knowledge flow should have five characteristics. (1) *Information accumulation*, it should be able to accumulate the predecessors' knowledge during the current project development period and keep knowledge generated during developing the previous projects. (2) *Classification*, it should be able to classify knowledge according to different projects and different team members. (3) *Abstraction*, it should be able to reflect knowledge at different abstraction levels and to purify the content. (4) *Analogy*, it should be able to establish analogy associations between the related contents of the flow. (5) *Version management*, it can manage the evolution process of the knowledge flow. A frame structure can be used to represent the knowledge flow:

```
KnowledgeFlowFrame (ProjectVersion, Member
Version, RevisionVersion)
{ AbstractionLevel1: Content1, AnalogyLink1;
...;
AbstractionLeveln: Contentn, AnalogyLinkn }.
```

The 'ProjectVersion' classifies the knowledge flow content according to the project types, i.e. different types of projects correspond to different project versions. The 'MemberVersion' classifies the knowledge flow content according to different team members, i.e. different member versions correspond to different members and development tasks. A project version consists of all the member versions of the project being developed. The 'RevisionVersion' is for keeping the old knowledge content of a member after it is revised. A team member can have several revision versions for tracing the revised content. The 'AbstractionLevel' classifies the knowledge flow content from low to high abstraction level. Since team members are both the writer and reader of the

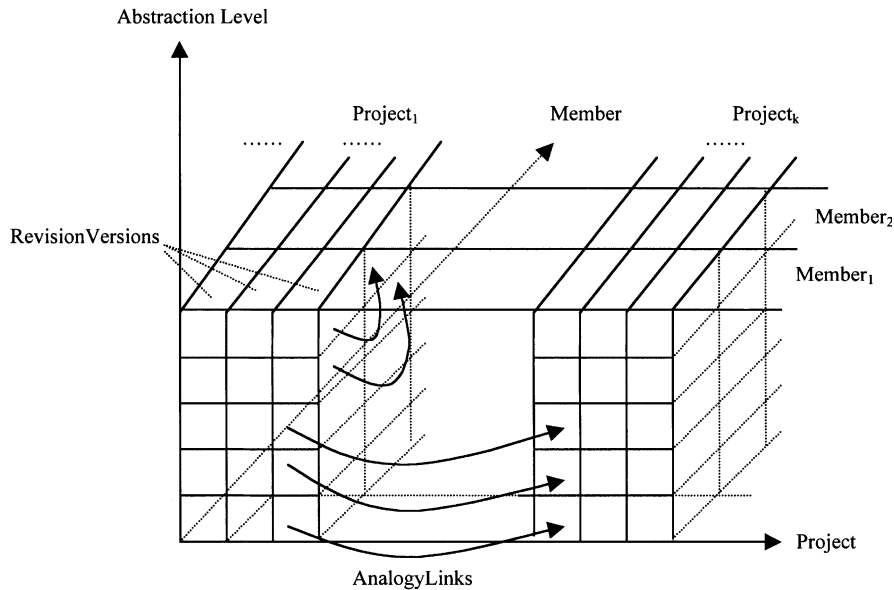


Fig. 2. Spatial structure of the knowledge flow frame.

frame content, the knowledge content description should be easily understood by all team members. The markup languages of the Semantic Web (<http://www.semanticweb.org>) are eligible for this purpose. The ‘AnalogyLink’ specifies the similar associations between the knowledge contents. Analogy link can only associate two different knowledge contents at the same abstraction level. Fig. 2 graphically describes the spatial structure of the frame, where the arrows represent the analogy links.

Five abstraction levels can be classified from low to high according to people’s cognitive process in software development processes.

- *Code level.* This is the lowest abstraction level that helps team members to share programming skills with each other during development processes. The content of this level is the programming skills described as a set of problem-solution pairs. The analogy link of this level points to the similar problem-solution pairs of the related versions.
- *Component level.* This level reflects knowledge about the components being developed by the corresponding team members. It can help team members to reuse components. The content of this level consists of the following two items. (1) *Category*, this item specifies the category the component belongs to. It is a kind of component generalisation. The content of this item is a category name. (2) *Dependent components and categories*, this item enables the succeeding team members to use the current component to identify the dependent components. The content is a set of ⟨component, category-name⟩ pairs. The corresponding analogy link content points to the similar component name and the related versions. This link can extend the usage region of the component.

- *Method level.* This item enables the succeeding team members to reuse the problem-solving method for solving their problems. The content of this level is described as problem-method pairs, where the method can be the process, the pattern, or the algorithm for solving a problem. The corresponding analogy link content points to the similar problem-method pairs with the related versions and the optional methods to the same problem.
- *Rule level.* This level records the development rules generated during development processes and the pre-designed cognitive cooperation rules. With the workflow execution, the development rules will become richer and richer. The development rules enable the succeeding team members to share development rules. Cooperation rules define the cooperation among team members, which are very useful for the newly joined team members to know how to cooperate with the other team members. Rules should be generalised for supporting a general software development.
- *Decision and evaluation level.* This level reflects the decisions made during the development process and the evaluation of these decisions. It is used as the reference for the succeeding team members to make their decisions. The content is a set of 3-tuples: ⟨situation, decision, evaluation⟩. The evaluation reflects a kind of satisfaction degree about the decision. Such an evaluation can help the succeeding team members to avoid unsuccessful decisions when a similar situation is facing. The analogy link content of this level points to the similar situation-decision pairs with the related versions and the optional decisions to the same situation.

The mentioned reuse (or resource sharing) at different abstraction levels can be a complete reuse, a partial reuse, or

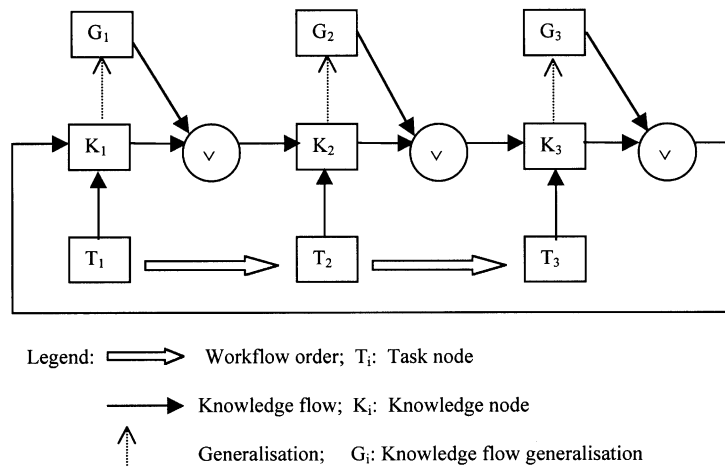


Fig. 3. Knowledge flow generation and generalisation during workflow execution period.

just heuristic information that help the other team members to accomplish their development tasks. Rules at the rule level can take the condition–action–result form as follows:

RuleId : ON \langle Condition \rangle DO \langle Action \rangle
 RESULT \langle ResultRecord \rangle and \langle SuccessRate \rangle ,

where \langle Condition \rangle can be logic ‘and’ or logic ‘or’ of several conditions, \langle Action \rangle can be sequential n actions, \langle Result Record \rangle records the *success* or *failure* of applying the rule, and \langle SuccessRate \rangle = successful-application-times/total-application-times.

4. Knowledge flow management

Knowledge flow management is to execute, control, store, and maintain a knowledge flow during a project development process.

4.1. Execution control and version management

Execution control mechanism is responsible for monitoring and scheduling the knowledge flow passing process. It should enable a knowledge flow process to keep pace with the workflow process. A team member should finish to fill in his/her knowledge into the knowledge flow frame during his/her development period then pass it to the succeeding team member(s). One way to realise the execution control is to incorporate the knowledge flow into the corresponding workflow process as a new flow type and to make full use of the WfMS to manage the knowledge flows and cognitive flows.

The version management consists of the space management and the version control. The creation of a knowledge flow frame requires a space that can be able to contain a certain number of project versions. Before developing a new project, a project space should be arranged for a project version that contains the knowledge generated during the

project development, and every team member should be assigned a space for a member version after the project space is created. The version control needs to manage three kinds of versions: the project version; the member version; and, the revision version. To avoid losing knowledge in case of flow passing exception, each knowledge node should keep a copy of the current knowledge flow (with the saving date) in a member knowledge flow repository before he/she passes the flow to the succeeding member.

4.2. Knowledge flow generalisation

Team members are responsible for not only the generation of knowledge flows, but also for the generalisation of knowledge flows. Fig. 3 graphically describes the relationship among the knowledge flow generation and generalisation as well as the workflow execution. The black arrows represent the workflow process. The dotted arrows denote the generalisation process. Each team member’s knowledge generalisation depends on three kinds of input: (1) the knowledge flow generated during developing the current component (the solid upward arrows in Fig. 3); (2) the knowledge flow output of the direct predecessor(s), the input of the first knowledge node can be the cognitive output of the end knowledge node of the last execution of the flow or the pre-designed initial input when it first executes; (3) the generalised knowledge flow of the direct predecessor. The generalised cognitive input of the first team member will be the generalised cognitive output of the end knowledge node of the last execution of the flow or the pre-designed initial input when it first executes. The generalised knowledge flow can extend its problem-solving region. It can also refine a knowledge flow so as to avoid unlimited expansion of the knowledge flow content.

4.3. Team knowledge repository

Before terminating the development of the present project, the output knowledge flow of the end knowledge

node of a KFN should be saved in a team knowledge repository for the reference of developing new projects. A team knowledge repository is a set of knowledge flow frames with the saving date. Team members can query the required detailed knowledge according to: the project version, the member version, the revision version, and the date during developing the new project period. The out-of-date content in the repository will be updated after the termination of a development process.

4.4. Complement of knowledge flow

Since a team member (especially, a newly joined member) may not be able to fully express his/her knowledge during development period. The knowledge flow management mechanism should allow him/her to complete his/her knowledge flow frame after accomplishing the current development task. The complement of a knowledge flow has two roles: (1) it can help a team member to provide more knowledge when developing new projects; (2) it can be sent to the current active knowledge node as one of the knowledge flow inputs when the current workflow has not been terminated. The knowledge flow complemented by a team member will be saved in the team knowledge flow repository as a new member version of the frame. The reason for keeping the old version is for keeping the match between the previous input and output of the succeeding knowledge node. A new version of them will be formed during the KFN running for developing a new project.

By storing the team's knowledge flow, a development team can adapt to the change of team members. Because all the knowledge of the team member who will leave the post has been recorded, a newly joined team member can become experienced quickly through learning the knowledge stored.

5. Internet-based knowledge exchange approaches

The Internet-based communication is the basis of the cognitive cooperation between the globally distributed team members.

- *E-mail-based approach.* This approach enables a team member to communicate with other team members through e-mail. A team member can send e-mail(s) to the receiver and need to wait for the reply when he/she is puzzled by solving the present problems or by using the receiver's resources. Usually a team member has to send and to reply e-mails to several team members during his/her development period, and repeatedly answer to the same or the similar question of them at different time, further more he/she cannot avoid to receive unnecessary e-mails from the other team members.
- *Blackboard-based approach.* It is a centralised communication approach, every team member can write

down questions on the blackboard and read answers from it. But the historical information is usually not recorded, and the logical order between different team members' knowledge is not reflected (this may cause information confusion). Although improvements can be made to overcome the two shortcomings, each team member still needs to frequently carry out the read and write operations on the virtual blackboard during his/her development process.

- *Knowledge-flow-based approach.* This approach can overcome the shortcomings of the previous approaches. First, a team member is only allowed to communicate with those who have direct work dependence relationship with him/her. Any team member can know the related predecessors' knowledge from the input knowledge flow, so unnecessary communication and frequent information exchange can be avoided. Secondly, the knowledge flow executes in the order coinciding with the corresponding workflow process logic, so information confusion can be avoided. Thirdly, the general and historical knowledge is also available from the knowledge flow, a newly joined team member can raise his/her cognitive ability by learning from the knowledge flow, so a team can adapt to the change of team members. Besides, it is not an overburden for most developers to summarise and input his/her experience within a reasonable short period during his/her development period. Two reasons support this claim. First, the time duration for knowledge input can be divided into two parts: one is the generation duration, and another is the in-key duration. The generation of a team member's knowledge naturally carries out with his/her thinking process during development period, it does not take any extra time. To avoid taking extra time for recollection and loss information, each team member should in-key the newly generated knowledge as soon as possible. Second, team development mainly aims at big projects, which usually need to take quite a long development period, e.g. several months. While the input of the knowledge usually takes a short period comparing with the development period, e.g. several minutes.

The comparison between the above three approaches and the approaches combining the three approaches shows that the combination of the e-mail-based approach and the flow-based approach cannot only keep the mobility of the e-mail-based approach or the blackboard-based approach but also has lower communication cost [13].

6. Reasons for incorporating knowledge flow into software development process

Reasons for incorporating the knowledge flow into the distributed team software development process include the following four aspects:

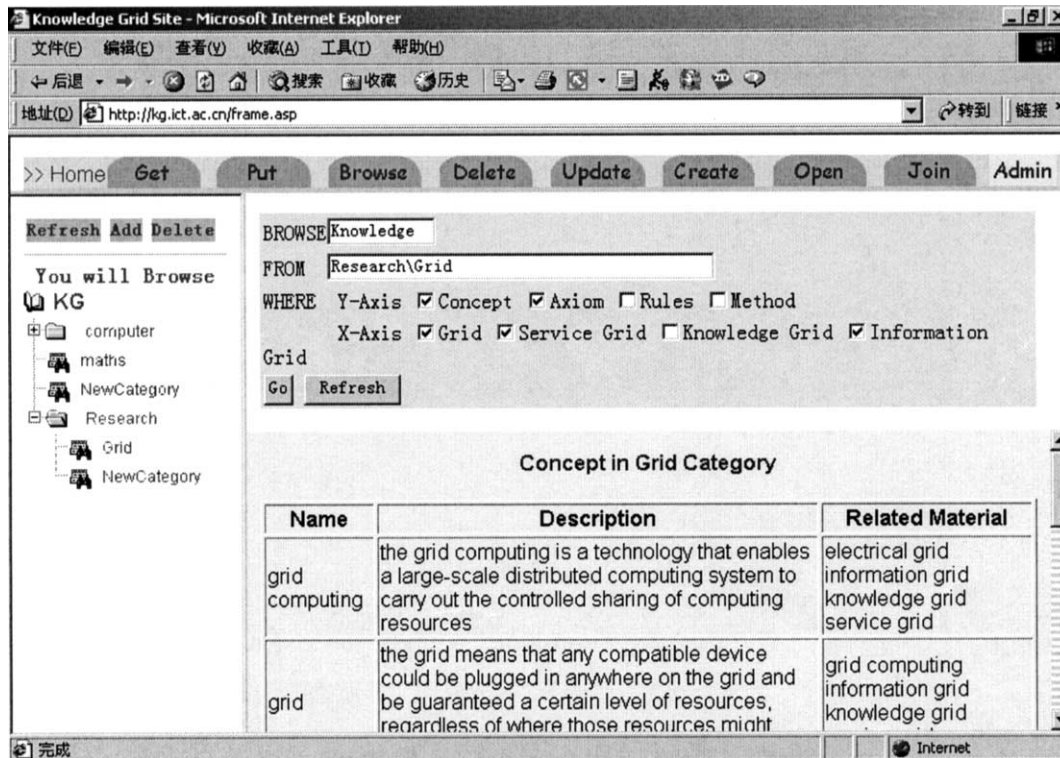


Fig. 4. An operation interface of the knowledge grid.

- *Software development process is a cognitive process.* The team members can improve their work not only with the support of the software tools, but also through cognitive cooperation between team members.
- *Cognitive cooperation cannot be pre-designed.* The team members' knowledge (experience, method, decision, and skill) about a software development is generated and accumulated during the development process, and the cognitive cooperation between them cannot be pre-designed. So a knowledge flow is needed to dynamically reflect the cognitive cooperation process.
- *A distributed team requires an effective and low cost communication.* An ordered communication can reduce the communication cost and can better reflect the real work process logic of a project development.
- *A development team should be supported by an experience accumulation mechanism.* Each team member can use the experience of its predecessor and the experience of the team accumulated during developing the previous projects, so the team members can be able to avoid redundant work, and the team can adapt to the change of team members.

7. Knowledge grid: a knowledge flow management platform

A *knowledge grid* (KG) is a mechanism that includes a knowledge space for uniformly storing knowledge and a knowledge operation interface for sharing and managing the

knowledge in the space. A knowledge space with three dimensions (level, category, location) is introduced in Ref. [14]. A local knowledge grid serves for one or a group of people. A local knowledge grid concerns three kinds of knowledge operation privileges: (1) the public knowledge that can be shared by all the other knowledge grids; (2) the group knowledge that can only be shared by the users of the same group; (3) the private knowledge that can only be used by its author.

A *worldwide* knowledge grid consists of local knowledge grids, and there exists at least one knows the meta-information about the others. The knowledge operations can selectively carry out on either a particular local knowledge grid or the worldwide knowledge grid in a universal view.

We have implemented the knowledge grid platform VEGA-KG for sharing knowledge across the Internet. The prototype is available to use at <http://kg.ict.ac.cn>. Fig. 4 is a knowledge operation interface of the knowledge grid. Users can conveniently create knowledge spaces and can carry out the get, put, delete, update, browse, open, and join operations on the knowledge space by using the SQL-like knowledge operation language embed in the interface. The middle portion of Fig. 4 shows the interface for determining the parameters of the operations.

Each member or a workgroup of a geographically distributed software development team can be equipped with a local knowledge grid for storing and managing local knowledge. A knowledge flow from member A to member B can be realised by informing B the coordinates of the

knowledge to be shared in the local knowledge grid of A and open its access privilege to B. The build-time KFN can be specified by determining the knowledge access privilege of each member and the work dependence relationship between the members.

8. Related works

Efforts to realise knowledge sharing have been made in AI field. Knowledge interchange format (KIF) is a language designed for use in the interchange of knowledge among disparate computer systems (<http://logic.stanford.edu/kif/>). Open knowledge base connectivity (OKBC, <http://www.ai.sri.com/~okbc/>) is a programming interface for accessing knowledge bases stored in knowledge representation systems. OKBC was developed under the sponsorship of DARPA's High Performance Knowledge Base Program. The current version of the OKBC specification, 2.0.3, was released on 9 April 1998. OKBC provides a uniform model of knowledge representation systems based on a common conceptualisation of classes, individuals, slots, facets, and inheritance. OKBC is defined in a programming language independent fashion. It consists of a set of operations that provide a generic interface to underlying knowledge representation systems.

The Semantic Web is an effort toward the next-generation web (see <http://www.semanticweb.org>). Its main intention is to provide information services by making the Web resources machine-understandable, because the current search engine does not know the content of the HTML-based Web pages and the current web pages cannot reflect its machine understandable semantics. Currently, research on the Semantic Web focuses on the new markup languages such as RDF (resource description framework, see www.w3c.org/rdf/), OIL (ontology inference layer), and DAML (DARPA Agent Markup Language). Ontology can establish a certain common understanding between information-processing mechanisms. It usually contains a hierarchy of concepts of a domain and describes each concept's crucial properties through an attribute-value. The WordNet (www.cogsci.princeton.edu/~wn) is a kind of concept-level ontology.

The sharing basis of the proposed knowledge flow is based on the Semantic Web. Currently, the XML (Extensible Markup Language) is used to represent knowledge in the introduced knowledge grid platform.

9. Summary

The advantages of incorporating the knowledge flow into distributed team software development process include two aspects. First, it realises the knowledge-level cooperation among the distributed team members. Such a cooperation is a controlled sharing of knowledge among team members

and has lower communication cost. Incorporating the knowledge flow with the workflow can promote the cooperation from the work level to the knowledge level in a distributed team software development process. This advantage results in the improvement of the problem-solving ability of a development team. Second, it enables a distributed team to be able to loosely depend on the cognitive ability of its member. Usually, the efficiency and quality of a software development depend on the cognitive ability of the team, which further depends on the cognitive ability of every team member. The information accumulation characteristic of the knowledge flow enables a development team to keep its cognitive ability in case of changing team members.

The proposed approach can be applied to any distributed human-computer process where the involved behaviours need the cooperation between human and the support work environment, especially in the globally distributed applications where team members cannot communicate with each other face to face and member recruitment often occurs.

Acknowledgment

The research work was supported by the National Science Foundation (NSF) of China.

References

- [1] F. Casati, M. Fugini, I. Mirbel, An environment for designing exceptions in workflows, *Inform. Syst.* 24 (3) (1999) 255–273.
- [2] A. Geppert, D. Tombros, K.R. Dittrich, Defining the semantics of reactive components in event-driven workflow execution with event histories, *Inform. Syst.* 23 (3/4) (1998) 235–252.
- [3] A.K. Goel, Design, analogy and creativity, *IEEE Expert* (1997) 62–70.
- [4] P. Lawrence, *Workflow Handbook 1997*, Wiley, New York, 1997.
- [5] F. Leymann, D. Roller, Workflow-based applications, *IBM Syst. J.* 36 (1) (1997) 102–122.
- [6] O. Marjanovic, M.E. Orłowska, On modeling and verification of temporal constraints in production workflows, *Knowledge Inform. Syst.* 1 (2) (1999) 157–192.
- [7] J. Puustjarvi, H. Tirri, J. Veijalainen, Reusability and modularity in transactional workflows, *Inform. Syst.* 22 (2/3) (1997) 101–120.
- [8] R. Rada, IT skills standards, *Commun. ACM* 42 (4) (1999) 21–37.
- [9] T. Rose, Visual assessment of engineering processes in virtual enterprises, *Commun. ACM* 41 (12) (1998) 45–52.
- [10] WfMC, The Workflow Reference Model, <http://www.wfmc.org>.
- [11] H. Zhuge, J. Ma, X. Shi, Analogy and abstract in cognitive space: a software process model, *Inform. Software Technol.* 39 (1997) 463–468.
- [12] H. Zhuge, T.Y. Cheung, H.K. Pung, A timed workflow process model, *J. Syst. Software* 57 (3) (2001) 231–243.
- [13] H. Zhuge, X. Shi, Communication cost of cognitive co-operation for team development, *J. Syst. Software* 57 (3) (2001).
- [14] H. Zhuge, A knowledge grid model and platform for global knowledge sharing, *Expert Syst. Appl.* 22 (4) (2002).