Design and Implementation of a Collaborative Virtual Problem-Based Learning Environment

Die Dissertationsschrift
Vorgelegt am Fachbereich Informatik
Der Technischen Universität Darmstadt von

Yongwu Miao
(Master of Engineering)

aus Wuhan, Volksrepublick China

zur Erlangung des Grades eines
Doktor-Ingenieur (Dr.-Ing.)

Erstreferent: Prof. Dr. Ralf Steinmetz
Korreferent: Prof. Dr. Heinz Ulrich Hoppe
Tag der Einreichung: 2. Oktober 2000
Tag der Disputation: 17. November 2000

Darmstadt 2000
Hochschulkennziffer D17

TABLE OF CONTENT

PREFACE 1

ABSTRACT 3

ZUSAMMENFASSUNG 7

1 INTRODUCTION 11

1.1 MOTIVATION 11
1.1.1 PEDAGOGICAL CONSIDERATIONS 12
1.1.2 TECHNOLOGICAL DEVELOPMENT 14
1.2 GOAL OF THE THESIS 15
1.3 ORGANIZATION OF THE THESIS 17

2 PROBLEM ANALYSIS 19

2.1 A SCENARIO OF PBL 19
2.2 CHARACTERISTICS OF PBL 21
2.2.1 RICH FORMS OF SOCIAL INTERACTION 22
2.2.2 ILL-STRUCTURED PROBLEM 22
2.2.3 SITUATED ROLES 23
2.2.4 SELF-DIRECTED LEARNING PROCESSES 24
2.3 MAJOR IMPLEMENTATION REQUIREMENTS 24

3 RELATED WORK 27

3.1 CCL 27
3.2 CSILE 28
3.3 CALE 30
3.4 CNB 31
3.5 BELVEDERE 32
3.6 MCBAGEL 34
3.7 WEB-SMILE 35
3.8 ANALYSIS OF THE STATE OF THE ART 39
3.9 SUMMARY 41

4 APPROACH 43

4.1 OVERVIEW OF THE APPROACH 43
4.1.1 PERSPECTIVE OF ACTIVITY THEORY 43
4.1.2 AN ANALYSIS OF PROBLEM BASED LEARNING ACCORDING TO

Activity Theory 45
4.1.3 A Conceptual Framework for the Design of Virtual PBL Environments 46
4.1.4 The Conceptual Architecture of a Virtual PBL Environment 48
4.1.5 Summary 50
4.2 Notation of Z 50
4.2.1 Basic Knowledge 51
4.2.2 The Z Notations Used in This Thesis 53
4.3 Virtual Institute Metaphor: A Context-based Virtual Learning Environment 58
4.3.1 Theoretical Background 59
4.3.2 Requirements 60
4.3.3 Design of a Context-based Virtual Learning Environment 61
4.3.4 Related Work and Discussion 93
4.3.5 Summary 94
4.4 PBL-net: An Activity-oriented, Graphical Knowledge Representation Language for PBL 95
4.4.1 Theoretical Background and a Conceptual Model of Collaborative Learning 96
4.4.2 Requirements 99
4.4.3 An Activity-oriented, Graphical Knowledge Representation Method 101
4.4.4 PBL-net 105
4.4.5 Related Work and Discussion 117
4.4.6 Summary 119
4.5 PBL-protocols: Guiding and Controlling Social Interaction in PBL Processes 120
4.5.1 Theoretical Background 121
4.5.2 Collaboration Protocol 122
4.5.3 PBL-protocols 143
4.5.4 Related Work and Discussion 152
4.5.5 Summary 153
4.6 PBL-plan: Coordinating Actions in PBL Processes 153
4.6.1 Theoretical Background 154
4.6.2 PBL-plan 156
4.6.3 From Learning Issues and Necessary Learning Resources to a PBL-plan 176
4.6.4 Modifying and Refining PBL-plans Interactively 186
4.6.5 Execution of PBL-plans 190
4.6.6 Related Work and Discussion 200
4.6.7 Summary 202

5 IMPLEMENTATION 205

5.1 System Architecture 205
5.2 A Mapping from an Abstract Implementation Model to The System Architecture 206
5.2.1 Abstract Implementation Model 206
5.2.2 A Mapping from the Components to the System Architecture 211
5.3 Implementation of the System Architecture 211
5.4 A Cooperative Hypermedia Approach 214
5.4.1 Implementation of the Virtual Institute Metaphor 214
5.4.2 Implementation of the PBL-net 217
5.4.3 Implementation of the PBL-protocol 219

5.4.4 IMPLEMENTATION OF THE PBL-PLAN 220

6 A USAGE SCENARIO AND EXPERIENCES 223

6.1 A USAGE SCENARIO 223
6.1.1 VIRTUAL INSTITUTE EDITOR AND BROWSER 223
6.1.2 PBL-NET SCHEMA EDITOR 226
6.1.3 PBL-PROTOCOL SCHEMA EDITOR 227
6.1.4 GROUP DEFINITION TOOL 228
6.1.5 PROTOCOL CONTROL PANEL 229
6.1.6 PBL-PLAN DEFINITION, MONITORING, AND EXECUTION TOOL 230
6.1.7 HYPERDOCUMENT EDITOR AND BROWSER 231
6.2 EXPERIENCE 234

7 CONCLUSIONS AND FUTURE WORK 237

7.1 MAIN CONTRIBUTIONS OF THIS THESIS 237
7.1.1 A CONCEPTUAL FRAMEWORK FOR THE DEVELOPMENT OF VIRTUAL PBL ENVIRONMENTS 237
7.1.2 A CONTEXT-BASED APPROACH TO THE DESIGN OF COLLABORATIVE VIRTUAL LEARNING ENVIRONMENTS: THE VIRTUAL INSTITUTE METAPHOR 238
7.1.3 AN ACTIVITY-ORIENTED APPROACH TOWARDS A GRAPHICAL KNOWLEDGE REPRESENTATION: THE PBL-NET 238
7.1.4 AN APPROACH TO GUIDE AND CONTROL SOCIAL INTERACTION: THE PBL-PROTOCOL 239
7.1.5 AN APPROACH TO SUPPORT SESSION-BASED COLLABORATIVE PROCESSES: THE PBL-PLAN 239
7.1.6 EXTENSION AND APPLICATION OF A COOPERATIVE HYPERMEDIA MODEL 240
7.2 COMPARISON WITH OTHER PBL SUPPORT SYSTEMS 241
7.3 OPEN ISSUES 244
7.3.1 SHORT-TERM DIRECTIONS 244
7.3.2 LONG-TERM DIRECTIONS 245

8 REFERENCES 247

APPENDIX A: LIST OF DEFINITIONS 261

APPENDIX B: LIST OF FIGURES 263

APPENDIX C: LIST OF TABLES 265

Preface

The work described in this thesis was performed at the Integrated Publication and Information Systems Institute (IPSI) of the German National Research Center for Information Technology (GMD) in Darmstadt. This work would not have reached the present point without the support of many people. It is impossible to fully acknowledge all of them who contributed either directly or indirectly to its completion. For those contributors whom I do not explicitly acknowledge, please be assured that your support did not go unnoticed.

First of all, I would like to express my gratitude to my advisors Professor Ralf Steinmetz (GMD-IPSI / Darmstadt University of Technology) and Professor Ulrich Hoppe (University of Duisburg) for their comments and advisce. I am also grateful to other members of my committee for their insightful questions and the helpful discussion during my final defense.

I would like to express particular thanks to Dr. Jörg M. Haake, the leader of the CONCERT division at GMD-IPSI, for his excellent guidance and intensive discussions during the course of completing this thesis work. I am deeply grateful to him for his careful reviewing and constructive criticism on several versions of this thesis. It would have been impossible to complete this work without his support and help.

Furthermore, I wish to thank all members of CONCERT division. As a researcher in collaborative work and collaborative learning, I view this work as a joint effort. In particular, the discussions conducted in the CLear project were very helpful in performing this thesis work. I feel fortunate for being part of a multi-disciplinary group with psychologists, pedagogues, and computer scientists. An effective collaboration resulted in a series of joint publications. Therefore, I thank both current and former members of the CLear project: Martin Wessner, Dr. Hans-Rüdiger Pfister, Jennifer Beck-Wilson, Torsten Holmer, Jutta Maria Fleschutz, Peter Zentel, and Dr. Shirley Holst. In addition, when I implemented CROCODILE, I received a lot of help from the people in the COAST development team including Christian Schuckmann, Holger Kleinsorgen, Hans Scholz, and Till Schümmer.

Many other colleagues deserve special mention. These include Emil Wetzel, Ute Sotnik, and Dr. Wolfgang Schuler for their friendship and for their willingness to help me with many administrative matters and in learning German. Cordial thanks go to the former and current Chinese colleagues at GMD-IPSI, who help me in many respects and prevented me from being homesick. I extend my thanks to Professor Zhipeng Tong, the former president of China Academy of Electronics and Information Technology, for his concern and encouragement.

Finally, I wish to express special gratitude to my family members. My parents instilled in me the desire to learn and the drive to pursue this work. In particular, I can not thank my wife Yi Li enough for the encouragement, support, patience, and help she has given no matter whether my work ran smoothly or was difficult. Her love and faith in me is a constant source of intelligence and strength for me. For this reason, I dedicate this thesis to her.

Abstract

Problem-based learning (PBL) is a student-centered learning method that provides students with resources, guidance, and instruction. In PBL, guided by tutors in the role of facilitators of learning, students learn domain knowledge together with developing problem-solving, collaborative learning, and self-directed learning skills.

Many collaborative virtual PBL environments have been developed in the research area of computer supported collaborative learning in the past decade. However, existing systems are limited in the sense that they focus mainly on providing and maintaining shared information resources and shared workspaces. The role of cultural factors and social factors as mediators in PBL processes are insufficiently addressed in these systems. The research described in this paper can be viewed as a first step towards addressing this limitation. The objective of this thesis work is to develop concepts and approaches to build a comprehensive collaborative virtual environment for distributed PBL.

The approach used to develop a collaborative virtual PBL environment is based on activity theory. According to activity theory, a human activity is defined as the engagement of a subject towards a certain goal or objective in a community. In most human contexts, activities are mediated through the use of culturally established instruments (such as language and artifacts) and socially established relations (such as regulations and work procedures). In addition, human activity can be described as a three-level structure: activity, action, and operation. Based on activity theory, a conceptual framework for the design of virtual PBL environments is proposed, which consists of eight components: *agent, place, tool, language, document, action, work description,* and *behavior rule*. According to this conceptual framework, a conceptual architecture of a collaborative virtual PBL environment is designed, which is depicted in the following figure. This conceptual architecture consists of four modules: the *virtual institute metaphor*, the *PBL-net*, the *PBL-protocol*, and the *PBL-plan*. Each module realizes a concept that contributes to meet the requirements for achieving the goal.

Based on the theory of situated learning, which emphasizes the importance of context and social interactions, the concept of *learning context* and an approach to develop *context-based virtual learning environments* have been developed. This approach is used to design a hierarchically structured learning context, called a *virtual institute metaphor*. The virtual institute metaphor reflects parts of the culture used in traditional learning environments. The virtual institute metaphor enables users to be aware of learning contexts, to interact with learning contexts, and to create and modify their learning environments. It therefore support customized learning contexts in which learning processes and interaction between learners can be situated.

Based on the theory of constructivism and situated learning, an *abstract model of collaborative learning* is developed. This model addresses potential conflicts at the individual memory level and at the group memory level. Considering the state-of-the-art in terms of graphical knowledge representation methods, it is found that existing graphical knowledge representation approaches are not suitable to support the PBL activity. An *activity-oriented approach* to a graphical knowledge representation is

developed. This approach is appropriate for supporting the representation, exploration and negotiation of shared knowledge in ill-structured knowledge domains. This approach is applied to support PBL activities. As a consequence, the *PBL-net schema*, a graphical knowledge representation language for PBL, is developed. It is designed to support PBL-specific tasks. By means of such a knowledge representation language, users can construct a *PBL-net*, representing their shared knowledge. The PBL-net reflects the status of collaborative learning and shared knowledge. It facilitates pursuing common understanding and constructing shared knowledge in the PBL processes.

Based on schema theory, the concept of *collaboration protocol* and an *approach to model and execute collaboration protocols* is developed. A collaboration protocol is a computerized script of a collaboration strategy. As a description of a collaboration strategy, a collaboration protocol may have one or more instances, called *protocol instances.* The concept of collaboration protocol is applied to develop *PBL-protocols* that support PBL groups within virtual learning environments. PBL-protocols can guide and control construction of the shared PBL-nets by suggesting and restricting behaviors of roles in each state of the PBL process. In addition, the idea of sub-protocol is used to develop a *negotiation protocol*, which can facilitate negotiation processes for the construction of shared knowledge. The PBL-protocol mediates PBL processes at the operation level and coordinates the contributions of people based on the distribution of the subject (different roles) in the community.

Based on self-directed learning, the concept of PBL-plan is developed. A PBL-plan is a description of a PBL procedure in a computerized form. A PBL-plan consists of a set of *action nodes*, a set of *connection nodes*, and a set of *artifact nodes*. A PBL-plan may have *sub-plans*. Action nodes can be connected in sequence or in parallel by using connection nodes. Therefore, users are supported to decompose their learning goals, to arrange actions to achieve these goals, to allocate resources, to project time line, in short, to make their own learning plan. They can carry out a PBL process by automatically executing their learning plan to coordinate actions performed by learners or sub-groups at same/different time and in same/different virtual places. In addition, in order to ease the creation of such plans, an approach to automatically create a preliminary PBL-plan and to support interactive modification and refinement of a PBL-plan is developed. By means of the PBL-plan, PBL groups are supported to conduct self-directed learning processes in a virtual learning environment. The PBL-plan mediates PBL processes at the action level and coordinates the contributions of people based on the distribution of the objects (different goals and tasks) in the community.

The architecture described above has been implemented as a prototype, called CROCODILE, that demonstrates the concepts. In CROCODILE the cooperative hypermedia model is extended and applied to implement the virtual institute metaphor, the PBL-net, the PBL-protocol, and the PBL-plan in a unified way. The prototype system is implemented by using a client/server architecture. Each client provides a user interface for users to interact with the application. Therefore, geographically distributed and co-located people are supported to conduct synchronously and asynchronously collaborative PBL activities in the virtual learning environment. The prototype has been tested. Preliminary results demonstrate that the experience and skills of social interaction in the real world can be intuitively reused in

CROCODILE. The system facilitates conducting a PBL activity in a virtual learning environment.

In comparison to existing virtual PBL environments, CROCODILE focuses on supporting scheduled, synchronous performed by a small group of adult learners. It emphasizes the role of mediation of cultural and social factors to support collaborative PBL activities. Finally, directions for future research based on this thesis work are discussed. The short-term directions include further evaluating the approach and the prototype system and solving problems raised by this research. The long-term direction aims at developing an integrated environment to support an integration of collaborative work and collaborative learning.

Zusammenfassung

Problem-basiertes Lernen (PBL) ist eine studentenzentrierte Lernmethode, die die Studenten mit Ressourcen, Anleitung, und Instruktionen versorgt. In PBL fungiert der Tutor als Moderator beim Lernen. Die Studenten erlernen dabei inhaltsbezogene Kenntnisse zusammen mit Fähigkeiten der Problemlösung, des kooperativen Lernens, und des selbstbestimmten Lernens.

Im vergangenen Jahrzehnt wurden im Forschungsgebiet des computer-unterstützten kooperativen Lernens viele kooperative virtuelle Umgebungen für PBL entwickelt. Existierende Systeme fokussieren hauptsächlich die Unterstützung der Kommunikation und Zusammenarbeit zeitlich und räumlich verteilter Lerner. Die Rolle kultureller und sozialer Faktoren als Mediatoren in PBL-Prozessen wurden jedoch in diesen Systemen nur unzureichend berücksichtigt. Diese Arbeit kann als ein erster Schritt angesehen werden, diese Einschränkungen aufzuheben. Das Ziel dieser Arbeit ist die Entwicklung von Konzepten und Ansätzen für den Aufbau einer umfassenden kooperativen virtuellen Umgebung für verteiltes PBL.

Der Lösungsansatz basiert auf der Aktivitätstheorie. Gemäss der Aktivitätstheorie wird eine Aktivität als das Engagement eines Subjekts für ein spezielles Objekt definiert. In den meisten menschlichen Kontexten leben und arbeiten Leute in einer Gemeinschaft. Im Laufe der Entwicklung der menschlichen Zivilisation wurden zur Unterstützung von Aktivitäten immer mehr kulturell etablierte Instrumente (wie zum Beispiel Sprache und Werkzeuge) benutzt. Außerdem bildeten und etablierten sich soziale Beziehungen (wie zum Beispiel Regeln und Arbeitsteilung). Zusätzlich kann eine menschliche Aktivität innerhalb einer Drei-Ebenen-Struktur beschrieben werden. Die drei Ebenen sind: Aktivität, Aktion und Operation. Basierend auf der Aktivitätstheorie wird ein konzeptionelles Rahmenwerk für den Entwurf kooperativer virtueller Umgebungen für PBL vorgeschlagen. Das konzeptionelle Rahmenwerk besteht aus acht Komponenten: Agent, Ort, Werkzeug, Sprache, Dokument, Aktion, Regeln und Arbeitsbeschreibung. Gemäß diesem konzeptionellen Framework wird eine konzeptionelle Architektur für kooperative virtuelle Umgebungen für PBL entworfen. Diese Architektur besteht aus vier Modulen: der Methapher des „Virtuellen Instituts", dem PBL-Protokoll, dem PBL-Netz und dem PBL-Plan. Jedes dieser vier Module realisiert ein Lösungskonzept, das zur Erfüllung der Anforderungen beiträgt.

Basiert auf der Theorie des situierten Lernens, die die Wichtigkeit von Kontext und sozialen Interaktionen betont, wird das Konzept des Lernkontext und ein Ansatz für kontext-basierte virtuelle Lernenumgebungen entwickelt. Dieser Ansatz wird zum Design eines hierarchisch strukturierten Lernkontextes beziehungsweise der Metapher des „Virtuellen Instituts" benutzt. Die Metapher des „Virtuellen Instituts" erbt einen Teil der Kultur, wie sie in realen Lernumgebungen existiert. Sie unterstützt die soziale Orientierung, Gruppenbewusstsein, reiche Formen sozialer Interaktion und die Anpassung der Lernumgebung.

Basierend auf der Theorie des Konstruktivismus und des situierten Lernens wird ein abstraktes Modell des kooperativen Lernens entwickelt. Dieses Modell berücksichtigt potentielle Konflikte auf der Ebene der individuellen Erinnerung und der

Gruppenerinnerung. In Anbetracht des aktuellen Entwicklungsstandes der Methoden für die graphische Darstellung von Wissen kann festgestellt werden, dass existierende graphische Methoden zur Wissensrepräsentation nicht für die Unterstützung von PBL Aktivitäten geeignet sind. Deshalb wird ein Aktivitäts-orientierter Ansatz für die graphische Wissensrepräsentation entwickelt. Dieser Ansatz ist geeignet für die Darstellung, der Erkundung und Verhandlung gemeinsamen bzw. geteilten Wissens in schlecht-strukturierten Wissensdomänen. Dieser Ansatz wird daher für die Unterstützung von PBL-Aktivitäten in dieser Arbeit verwendet. Für diese Aufgabe wird das PBL-Netz-Schema, eine graphische Wissensrepräsentations-Sprache für PBL, entwickelt. Es dient zur Unterstützung PBL-spezifischer Aufgaben. Mittels eine solche Sprache zur Wissensrepräsentation können Benutzer ein PBL-Netz aufbauen, sowie ihr gemeinsames Wissen repräsentieren. Das PBL-Netz reflektiert den Status des kooperativen Lernens und des gemeinsamen Wissens. Es unterstützt das Erreichen von gegenseitigem Verständnis und das Aufbauen gemeinsamen Wissens in PBL-Prozessen.

Basierend auf der Schema-Theorie werden das Konzept der Kooperationsprotokolle und ein Ansatz zum Modellieren und Ausführen von Kooperationsprotokollen entwickelt. Ein Kooperationsprotokoll ist ein computerunterstütztes Skript für eine Kooperations-strategie. Als Beschreibung einer Kooperations-strategie kann ein Kooperationsprotokoll in konkreten Prozessen mehrfach eingesetzt werden (Protokoll-Instanzen). Das Konzept der Kooperationsprotokolle wird für die Entwickeln des PBL-Protokolls benutzt, das die PBL-Gruppen innerhalb der kooperativen virtuellen Lernenumgebung unterstützt. PBL-Protokolle können die Konstruktion der gemeinsam genutzten PBL-Netze führen und kontrollieren, indem sie das Verhalten der jeweiligen Rollen durch Vorschlagen und Einschränken von Operationen in jedem Zustand des PBL-Prozesses beinflussen. Zusätzlich wird die Idee des Unterprotokolls für die Entwicklung eines Verhandlungsprotokolls benutzt, das den Verhandlungsprozess bei der Konstruktion gemeinsamen Wissens unterstützen kann. Das PBL-Protokoll steuert den PBL-Prozess auf der Operationsebene und koordiniert die Beiträge der Lerner aufgrund der Verteilung von Subjekten (mit verschiedenen Rollen) in der Gemeinschaft.

Basierend auf selbstgeleitetem bzw. selbstbestimmtem Lernen wird das Konzept des PBL-Plans entwickelt. Ein PBL-Plan ist eine Beschreibung einer PBL-Prozedur in einer computerunterstützten Form. Ein PBL-Plan besteht aus einer Menge von Aktionsknoten, einer Menge von Verbindungsknoten, und einer Menge von Artefakt-knoten. Ein PBL-Plan kann Unterpläne haben. Aktionsknoten können in serieller Reihenfolge oder parallel zueinander durch Verbindungsknoten verbunden werden. Hierdurch werden Benutzer beim Erstellen ihren eigenen Lernplans untestützt, z.B. beim Unterteilen ihrer Ziele, beim Arrangieren der Aktionen zum Erreichen dieser Ziele, beim Zuteilen der Ressourcen, bei der Zeitplanung, kurz gesagt. Sie können einen PBL-Prozess durch automatisches Ausführen ihres Lernplans realisieren. Dabei werden die Aktionen mehrerer Lernern oder Untergruppen zur gleichen/zu verschiedenen Zeiten und an gleichen/verschiedenen virtuellen Orten koordiniert. Um das Erzeugen solcher Pläne zu vereinfachen wird zusätzlich ein Ansatz entwickelt, der einen vorläufigen PBL-Plan automatisch generiert und die interaktive Änderung und Verfeinerung eines PBL-Plans unterstützt. Mittels des PBL-Plans werden PBL-Gruppen beim Durchführen der selbst-gesteuerten Lernprozesse in einer kooperativen virtuellen Lernumgebung unterstützt. Der PBL-Plan koordiniert den PBL-Prozess auf der Aktionsebene und koordiniert die Beiträge der Lerner aufgrund der Verteilung von Objekten (gemäss verschiedener Aufgaben oder Ziele) in der Gemeinschaft.

Diese Architektur ist als ein Prototyp realisiert worden, der die Lösungskonzepte demonstriert. Der Prototyp heißt CROCODILE. In CROCODILE wird das kooperative Hypermedia-Modell erweitert und zum Implementieren der Metapher des „Virtuellen Instituts“, des PBL-Netzes, von PBL-Protokollen und des PBL-Plans verwendet. Das Prototypsystem wird mit einer Client/Server- Architektur implementiert. Jeder Client bietet eine Benutzungsoberfläche für die Interaktion zwischen Benutzer und der Anwendung. Daher werden sowohl räumlich verteilte Leute als auch Leute an einem Ort unterstützt, um synchron und asynchron kooperative PBL-Aktivitäten in der virtuellen Lernumgebung durchzuführen. Das Prototypsystem wird getestet. Die ersten Ergebnisse unterstützen die Annahme, dass die Erfahrungen und Fähigkeiten der sozialen Interaktion in der realen Welt intuitiv in CROCODILE wiederbenutzt werden können. Das System erleichtert das Durchführen von PBL-Prozessen in einer virtuellen Lernenumgebung.

In Vergleich zu verwandten virtuellen Umgebungen für PBL fokusiert CROCODILE auf die Unterstützung von geplanten, synchronen kooperativen PBL-Prozessen, die von kleinen Gruppen von erwachsenen Lernern durchgeführt werden. CROCODILE betont die Unterstützung vom kulturellen und sozialen Aspekt in kooperativen PBL-Prozessen. Zum Abschluss werden Richtungen für kurzfristige und langfristige zukünftige Forschungsarbeiten diskutiert. Die kurzfristigen Richtungen schliessen die tiefergehende Evaluation des Ansatzes und des Prototypsystems sowie das Lösen einiger Probleme ein, die sich bei der Anwendung des Ansatzes gezeigt haben. Die langfristige, Forschungsthemen zielen auf die Entwicklung einer integrierten Umgebung zur Integration von kooperativen Arbeiten und kooperativen Lernen.

1 Introduction

Nowadays, problems faced in industry and academy have become increasingly complex. The ability and skills needed to solve these problems are often not taught in the usual teacher-centered approach. Problems typically taught in schools often are well-structured that lead to predetermined or fully predictable results. The ability to solve well-structured problems does little to increase the relevant and critical thinking skills, which are very important for students to solve problems they will face in their future work, community, and personal lives. In addition, problem solving today is often the collaborative activity of a multi-disciplinary team. Students should have skills to interact with others who may have different disciplines. Unfortunately, learners taught in a teacher-centered learning approach are not adequately prepared when they face to real-world problems [Pross99].

According to Norman et al., a revolution in education is taking place, in which we see a shift from the teacher-centered approach to the learner-centered approach. The underlying philosophy is that people learn best when they actively engage in acquiring the knowledge and skills, which they need to solve the problem at hand. The advance of technology has accelerated this change and enables many ideas of modern education theories and instruction methods to be carried out. Learner-centered is often accompanied by a problem-based approach [Norman96]. In the past decade, many systems have been developed to support learner-centered approach in computer-based learning environments. In the same vein, the concern of this thesis work is to develop an approach and a computer-based learning environment for the support of problem based learning, a typical learner-centered approach.

1.1 Motivation

This research is motivated by two major factors. The first major influence comes from pedagogical considerations. In the pedagogical literature there is an increasing recognition of the importance of collaboration and coordination in learning. An understanding of how student's knowledge structures, worldviews, motivations and interpersonal interactions interact with learning environments is central. The specific focus of this thesis is on the problem based learning (PBL) method, which is heavily influenced by such pedagogical considerations. The implementation requirements for a computer-supported problem based learning environment can be derived from the PBL method and its practice. The second major factor motivating this research is technological advances such as the development of hypertext/hypermedia, computer-supported cooperative work (CSCW), and computer-supported cooperative learning (CSCL) technologies. Swiftly advancing technologies propel the opportunities for computational support beyond information keeping and sharing among distributed people involved in PBL. It makes it possible to build a new version of the PBL environment enhanced by new technologies.

1.1.1 Pedagogical Considerations

PBL is a learner-centered instructional method. Barrows et al. defined PBL as "...the learning which results from the process of working towards the understanding of, or resolution of, a problem" [Barrows80]. Boud and Feletti claimed: "Problem based learning is an approach to structuring the curriculum which involves confronting students with problems from practice which provide a stimulus for learning" [Boud91]. Mayo et al. stated that PBL posed significant, contextualized, authentic situations, and provided students with resources, guidance, and instruction when students developed and applied domain knowledge and problem-solving skills [Mayo93].

Three reasons led to choosing PBL as the research domain of this thesis work: 1) PBL is consistent with the philosophical view of modern learning theories; 2) PBL embodies most of the principles that improve learning; 3) PBL is increasingly popular in practice.

PBL is consistent with the philosophical view of contemporary learning theories

PBL is based on contemporary learning theories such as constructivism, situated learning, and adult learning.

Constructivism: PBL is consistent with the view of constructivism. According to the constructivist perspective, learners are active constructors of knowledge for themselves. Constructivism recognizes the importance of an authentic learning task, the importance of the context in which the student works, and the importance of collaborative learning. For Dewey, knowledge is not something that is changeless, but something that depends on an activity, a process of discovery [Dewey38a]. Piaget believed: "to understand is to discover, or reconstruct by rediscovery, and such conditions must be complied with if in the future individuals are to be formed who are capable of production and creativity and not simply repetition" [Piaget73]. Von Glaserfeld has written: "… learners construct understanding. They do not simply mirror and reflect what they are told or what they read. Learners look for meaning and will try to find regularity and order in the events of the world even in the absence of full or complete information" [VonGlaserfeld84]. Woolfolk described the constructivist view of the learning process as follows: "… students actively construct their own knowledge: the mind of the student mediates input from the outside world to determine what the student will learn. Learning is active mental work, not passive reception of teaching" [Woolfolk93]. Duffy et al. wrote: "rather than 'teaching' the skills, the skills are developed through working on the problem, i.e., through authentic activity" [Duffy96]. Tam [Tam00] noted: "learning is determined by the complex interplay among learners' existing knowledge, the social context, and the problem to be solved. Instruction, then refers to providing learners with a collaborative situation in which they have both the means and the opportunity to construct 'new and situationally-specific understandings by assembling prior knowledge from diverse sources' [Ertmer93]". Grabe et al. believed: "constructivist learning experiences and appropriate classroom practices include reflective thinking and productivity; authentic activities, including student collaboration and consideration of multiple perspectives, and student access to content area experts who can model domain-specific skills" [Grabe98].

According to Savery and Duffy [Savery95], PBL environments are based on the following constructivist assumptions:

1) Anchor all learning activities to a larger task or problem.
2) Support the learner in developing ownership for the overall problem or task.
3) Design an authentic task.
4) Design the task and the learning environment to reflect the complexity of the environment they should be able to function in at the end of learning.
5) Give the learner ownership of the process used to develop a solution.
6) Design the learning environment to support and challenge the learner's thinking.
7) Encourage testing ideas against alternative views and alternative contexts.
8) Provide opportunity for and support reflection on both the content learned and the learning process.

Situated learning: PBL is consistent with the theory of situated learning. In the view of situated learning, knowledge and understanding is fundamentally a product of the learning situation and the nature of the learning activity [Lave91a]. Brown et al. believed that knowledge " … is situated. A corollary of this is that learning methods that are embedded in authentic situations are not merely useful; they are essential" [Brown89]. They argued: "activity, concept, and culture are interdependent. No one can be totally understood without the other two. Learning must involve all three" [Brown89]. Jonassen argued: "the most effective learning contexts are those which are problem or case based and activity oriented, that immerse the learner in the situation requiring him or her to acquire skills or knowledge in order to solve the problem or manipulate the solution" [Jonassen91]. Lave wrote: "'situated'…does not imply that something is concrete and particular, or that it is not generalizable, or not imaginary. It implies that a given social practice is multiply interconnected with other aspects of ongoing social processes in activity systems at many levels of particularity and generality" [Lave91b]. Wenger uses *Communities of Practice* to describe the impact of social learning [Wenger98]. It emphasizes sharing and doing, and the construction of meaning in a social unit [Roschelle95]. McDermott believed: "Communities of practice focus on learning within functions or disciplines, sharing information and insight, collaborating on common problems, stimulating new ideas" [McDermott99].

Adult learning: PBL is consistent with the theory of adult learning. Camp wrote: "PBL, at least in the "pure" implementation form, fits with tenets of adult learning theory. Student autonomy, building on previous knowledge and experiences, and the opportunity for immediate application are all well-known to facilitate learning in adults, and thus should foster the success of a PBL approach with students who are adult learners" [Camp96]. As Knowles et. al. pointed out, adult students are more motivated to learn when they know why they need to learn something, when their previous knowledge and experience is used as a starting point of learning and a resource for learning, when they approach learning as problem-solving, and when the learning topic have immediate relevance to their job or personal life [Knowles84]. As Brundage and MacKeracher claimed, adult students learn better when learning activities express a tolerance for uncertainty, inconsistency, and diversity rather than demand a unique and correct answer. Adult learning is facilitated when adult students are given the opportunity to ask and answer questions and to find and solve problems, when they can evaluate their own skills and strategies to discover inadequacies and deficiencies, and when they are responsible for planning and implementing their own learning activities [Brundage80]. Camp stated that professional schools of all types would have an interest in the potential of PBL to facilitate learning in their students [Camp96].

PBL embodies most of the principles that improve learning

Woods [Woods96] pointed out that PBL embodied most of the general principles that improve learning. These principles are listed below:

1) Students should actively engage in the learning activities. They should not passively listening to lecture [Johnson82] [Johnson91].
2) Students should cooperate in learning processes to help one another [Johnson82] [Johnson91].
3) It is not necessary that all students learn the same way. Learning activities should be provided such that each student is able to have a preferred style [Keller68] [Grayson74] [Felder88].
4) Students should have explicit goals and criteria that can be used to check whether their goals have been achieved [Mager62] [Kibler74] [Popham70].
5) Give students feedback about their performance as quick as possible [Woods96].
6) Students should be empowered to have some role in the assessment such as peer- and self-assessment [Novak89] [Brown92].
7) A work environment should be provided with the clear expectation that students will succeed. It is expected that each student has a personal learning interest [Woods85].
8) Rich tutor-student interaction should be promoted through many different types of inside class and outside class events [Woods96].
9) It is not expected that "processing skills" can be developed by providing "opportunities." [Woods93] [Norman93].

PBL is increasingly popular in practice

PBL is an increasingly popular instructional method. Although the intellectual history of PBL is far older, its modern history began at McMaster University in Canada over 25 years ago. Until recently, the PBL approach are popular mainly in medical and professional schools. Gradually, other fields have begun adopting this method [Rhem98]. Although the history of PBL is not long and this instructional method is still under development, PBL has been used in many settings such as health sciences, nursing, dentistry, pharmacy, veterinary medicine, public health, architecture, business, law, engineering, forestry, police science, social work, education and many other professional fields [Camp96]. It can be used apparently in any subject and at most levels [Woods96].

1.1.2 Technological Development

Recently, information technologies such as computer and communication technologies have rapidly advanced. The digitization of our cultures is providing schools with access to a breadth of intellectual and cultural resources far greater than ever before; it is providing new, sophisticated and customizable tools for inquiry and investigation; it is enabling modes of interaction, communication, and collaboration not formerly possible [Lave91].

The advantages of computer-based learning environments are summarized by Neal [Neal98] as follows.

1) "Potential time savings
2) Potential cost savings
3) Increased accessibility to education for traditional and non-traditional students
4) Increased accessibility to experts
5) "Just-in-time" learning and training, especially important for reskilling and upgrading technical skills
6) More options and flexibility for class structure"

Currently, CSCW and hypertext/hypermedia are the two most widely used technologies to develop computer-based learning environments. These two technologies have different but related foci. In the CSCL community, both technologies are mainly used to support information sharing. CSCW technology is often used to remove the geographical and temporal constraints of face-to-face interactions in traditional classrooms by providing virtual classrooms. Hypertext/hypermedia technology is often used to overcome the linear structure of information by allowing non-linear integration of information chunks, which may be represented by different forms of media such as text, audio, video, image, graphics, etc.

It is important to note that both technologies are rapidly developing. These technologies can potentially provide support for developing advanced collaborative learning environments. For example, CSCW technology can be used to develop computational mechanisms to guide and control behaviors of people with different roles. Hypertext can be used to represent a set of hierarchically structured, connected places. Fully exploiting technologies to implement computer supported learning environments depend on how to apply the technologies under the guidelines of learning theories.

1.2 Goal of the thesis

In spite of many advantages, some barriers have to be overcome in order to implement PBL successfully even in conventional learning environments.

1) *Resistance of changing roles:* Teachers and learners who are unfamiliar with PBL tend to be reluctant to change their traditional roles [Jones94] [Bridges92] [Aspy93]. Teachers lack the skills of a facilitator in guiding learners to discover information for themselves. They do not want to shift their role from lecturers to tutors. Learners are also slow to adjust to the PBL method, and to the change in their role from that of passively receiving information to actively engaging in a problem-solving process.
2) *Lack of self-directed learning skills*: Students often express difficulties with self directed learning [Schmidt92]. It is difficult for learners to set their own learning goals or create and implement their own learning plans [Course Material]. It is

difficult for learners to identify, seek, manage, and utilize resources (experts and information).

3) *More cost*: Instead of one classroom (as regarded by traditional teaching), the PBL curriculum requires a number of small rooms equipped with tools for teaching and adequate copies of learning resources. The PBL curriculum requires more tutors [Aspy93].

When supporting PBL in a computer-based learning environment, on the one hand, computer-based learning environments can enhance the conventional learning environments to some extent. For example, the rooms with tools and copies of learning materials are easily created in the electronic form. On the other hand, in computer-based learning environments most of problems described above will be compounded and new problems will rise. The following are the major barriers to successful PBL in computer-based learning environments:

1) Learners and tutors have no experience of interacting with socially unfamiliar computer-based learning environments. It is not easy to construct and maintain the shared learning context that enables effective collaboration.
2) Exchange of ideas mainly relies on the shared information space, because people are distributed in space and time. This makes it difficult to pursue mutual understanding and to construct shared knowledge.
3) Weak communication channels make social interactions using social protocols (which we use in the conventional learning environment) difficult.
4) It is more difficult to coordinate learning activities, to make progress efficiently, and to keep track of progress toward the learning goals.

This thesis work aims at developing concepts and approaches to build computer-based learning environments, which help people to ease and overcome the difficulties described above. The target users of such virtual PBL environments are geographically distributed and co-located adult learners, who may conduct professional training and collaborative learning at work. Generally speaking, the goal is to design and implement a collaborative virtual PBL environment. The overall goal of this thesis can be decomposed into four sub-goals:

1) ***Support for social orientation and social interaction***: When conducting PBL in conventional learning environments, social orientation and social interactions are needed without question. People have rich experience to navigate in the real world and to interact with learning environments and with other people by using naturally inherited capability (e.g., walking, seeing, and gesture) and instruments (tools, language, and books) available. Therefore, one of the aims of this thesis is to build a virtual PBL environment in which PBL group members can use their experience intuitively, when interacting with the learning environment. It should enable people to customize their learning environment, and to communicate and collaborate with other people in same/different time/place cooperation modes.
2) ***Support for the pursuit of mutual understanding and the construction of shared knowledge***: When conducting PBL in a conventional learning environment, people have rich communication channels to exchange ideas to pursue mutual or common understanding. When conducting PBL in a virtual learning environment, because of weak communication channels and distribution of people in time and in space, the exchange of ideas mainly relies on shared information spaces, where

people can browse and manipulate their shared knowledge representation. One of the aims of this thesis is to develop a knowledge representation method for PBL, which can facilitate for PBL group members to represent their ideas and intentions, to understand others' ideas and intentions, to structure the shared information space, and to construct and negotiate shared knowledge.
3) ***Support for change of roles***: The roles of teachers and students played in PBL are different from their traditional roles. When carrying out different tasks in a PBL process, the expected behaviors of teachers and students are distinctive. One of the aims of this thesis is to support PBL group members (including learners, tutors, and experts) to become familiar with their new roles, to guide social interaction in PBL processes, and to avoid unexpected behaviors and unpredictable conflicts.
4) ***Support for self-directed learning***: In PBL, learners are responsible for setting learning goals, making learning plans, and searching for learning resources, rather than passively accepting knowledge as it is arranged by teachers in a lecture format. One of the aims of this thesis is to help PBL group members to set their learning goals, to make learning plans, to allocate tasks to various members of the learning group, to allocate learning resources, to keep track of progress towards learning goals, and to coordinate their actions.

1.3 Organization of the Thesis

The remainder of this thesis is structured as follows:

Chapter 2 starts by describing a real PBL scenario. The characteristics of PBL processes are analyzed based on the scenario and literature. Then, requirements for the development of a collaborative virtual PBL environment are identified.

Chapter 3 provides a survey of related computer-based PBL support systems. Each system is examined for its ability to meet the identified requirements. A summary of deficits of existing systems is given.

Chapter 4 presents the design and implementation of a collaborative virtual PBL environment. This chapter consists of six parts.

The first section of Chapter 4 presents the general approach adopted by this research work. It begins from a brief introduction of activity theory. Based on activity theory, a conceptual framework for the design of virtual PBL environments is developed. According to this conceptual framework, a conceptual architecture of a virtual PBL environment is designed.

The second section of Chapter 4 briefly introduces the Z language, which is used to formally specify the design of the collaborative virtual PBL environment. The basic knowledge about the Z language is described and the Z notations used in the thesis are listed.

The third section of Chapter 4 begins with an introduction of situated learning. Following the guidelines of this educational theory, requirements for the support of social orientation and social interaction are identified. The main body of this chapter describes the basic concepts of a context-based virtual learning environment and describes an approach to the development of a context-based virtual learning environment. The context-based virtual learning environment is formally described in order to demonstrate how to support the construction and maintenance of learning contexts, how to support awareness of learning contexts, and how to support social interaction. Finally, we summarize this chapter by comparing our approach with other approaches

The fourth section of Chapter 4 discusses the main principles of constructivism and situated learning. Following these principles, a model of collaborative learning is developed. This model is used to derive requirements for the design of a graphical knowledge representation method for collaborative learning. The main body of this chapter describes an activity-oriented approach to knowledge representation for learning. The approach is applied specifically to problem-based learning. It is compared to the activity-oriented approach and the content-based approach and the didactic-based approach.

The fifth section of Chapter 4 starts by briefly introducing schema theory. In the light of schema theory, an approach to model and execute a special kind of collaborative processes, called multiple-state collaborative processes, is specified. Then, how this approach is applied to guide and control problem-based learning processes is presented. Finally, a comparison with other approaches for the modelling cooperative processes is presented.

The sixth section of Chapter 4 begins with discussing principles of self-directed learning theory. These principles are used to derive requirements for the support of self-directed learning in PBL. An approach to enable learners to set their learning goals and make a learning plan is presented. It helps learners by creating a preliminary learning plan automatically and by modifying and refining the learning plan interactively. It supports coordination of actions by executing the defined learning plan. Finally, a comparison with workflow management systems is presented.

Chapter 5 discusses the implementation issues of the proposed collaborative virtual PBL environment. This chapter presents the system architecture of the prototype system and how to map abstract implementation model to the system architecture. It describes how cooperation support is implemented. Finally, it describes the cooperative hypermedia technology, which is used as an implementation approach.

Chapter 6 describes a usage scenario of the prototype system and preliminary experiences.

Chapter 7 begins by summarizing the thesis and outlining the main contributions of this thesis. Then, a comparison with existing PBL support systems is presented. The chapter ends by discussing open issues and directions of future work.

2 Problem Analysis

This chapter describes the problem-based learning process in detail through describing a scenario. This scenario can be used to analyze the characteristics of PBL and derive technical requirements for a virtual PBL environment.

2.1 A Scenario of PBL

The scenario presented here is developed from a real scenario [Summer Sleuths Program], in which the learning process lasted for four days. A dozen learners were involved in this course. In this section, we use only parts of the complete scenario presented in [Summer Sleuths Program]. This scenario is described according to different tasks performed in the PBL process.

Preparing students for PBL

Goal: Support learners as they encounter problem-based learning in an authentic, learning experience. The goal is achieved by performing the following steps:

1) Students sign in and join their pre-assigned groups in the auditorium.
2) Students are welcomed into program.
3) Students go to rooms for small groups to identify and get to know members of their group.
4) Students return to auditorium where speaker engages them with introductory remarks about PBL.

Task 1: Identifying the problem

Goal: Support learners as they become engaged in the problem with personal enthusiasm and interest as well as intellectual rigor.

Four guest speakers have asked to talk with students about the discovery of deformed frogs in local area. Pictures of deformed frogs are shown and a video titled "The Frogs – What Are They Really Telling Us?" is played in the auditorium room. They challenge the students to investigate the status of the frog population and will encourage them to take a proactive stand on this environmental concern.

Then the tutor coach students to identify and understand the problem. After discussion, students identify the problem: "what is cause of deformity of frog and how to prevent it from spreading?" Furthermore, they decomposed the problem into sub-problems such as what are the possible implications for humans? Finally, students post a preliminary problem statement and define relations among the problems.

Task 2: Identifying what learners know and need to know

Goal: Support learners in developing an awareness of what they know and need to know from the circumstances they have encountered and from their experience.

Students identify major issues connected to the problem. The identified issues are frog habitat, the various types of deformities in frog, wetlands, watersheds, the effects of pollution on a natural habitat, and so on. An issue can be decomposed into several sub-issues; for example, frog habitat includes frog food, living environment, etc. The tutor coach students in what they know and what they need to know (KNK) by asking questions, commenting, and giving hints. An example question may be "what would be some good resources?" An example comment is "the relationship between this issue and the identified problem is too weak to be considered." An example hint is "you forget a kind of insect, living in grass, is also a kind of frogs' food." Finally, they post the first version of KNK charts.

Task 3: Setting learning goals and making learning plan

Goal: Support learners to identify and decompose learning goals according to the KNK charts and to create effective information-gathering, information-sharing, and decision-making plan. The process includes the following steps.

1) Prioritize the needs to know (according to importance) and identify the prerequisite relation among them.
2) Set and decompose learning goals and objectives.
3) Arrange a set of coordinated learning actions to achieve the goals.
4) Identify and allocate resources (tutors/experts, materials, and rooms) so that learners know what is expected of them, by when and where.
5) If learners will work in a group, each member may wish to identify which tasks each will be engaged and what roles each will take.

Students finish this task by making a learning plan.

Task 4: Learning knowledge

Goal: Support learners in gathering data, acquiring knowledge, and understanding how new information contributes to an understanding of the problem and how information is assessed in light of its contribution to that understanding.

According to the learning plan, Students individually or in teams collect information from identified articles, books, videos, web sites, and other resources. They take notes and recordings in notebooks. They return to their team room to share information, and transfer those to other group rooms in which the group needs this information to perform certain subsequent actions.

In addition, students in teams are coached to perform science actions including habitat exploration, soil and water testing, population counts, etc, in order to help them understand the environmental factors affecting the frog. Each team writes reports collaboratively or by a delegate.

Students develop a set of questions for interviews. Then they in teams or individually interview persons such as the frog experts and others concerned by phone and face-to-face. They return to their home or team rooms to classify, count, and analyze the results of the interviews. They may transfer the information to other student groups concerned.

Students often report to the tutor about the progress during carrying out these actions. If some unexpected events occur, the tutor will help students to modify their learning plan to fit changes. Finally, students collect all information and make connections with and from collected information in a panel session action. They review the KNK chart. They decide whether they need to learn more knowledge, because they have deeper understanding about problem.

Task 5: Applying knowledge

Goal: Support learners in articulating the issues and the problem in the circumstances they are given and in identifying conflicts. Support learners to present and argue the hypotheses and solutions.

Students discuss problem situation, refine the problem statement, and propose tentative hypotheses and solutions. Tutor requires students to communicate, orally and/or in writing, their findings, hypotheses, and solutions. They find that they have different opinions on hypotheses and solutions. They use what they learned including evidences and principles to debate different perspectives. They evaluate the reliability of the findings and consider the reliability of the hypotheses and solutions. When they realize that they can not propose satisfied hypotheses and solutions or that necessary knowledge to good hypotheses and solutions is still insufficient, they repeat their work from task 2 to task 5. In this scenario, they repeat above tasks for two times. Finally, learners adopt a set of confirmed hypotheses and acceptable solutions. They submit their report with solutions to the City Council.

Task 6: Evaluating and assessing learning

Goal: Support learners in using the benchmarks of good thinking to evaluate the benefits and consequences of each possible solution in order to create the most acceptable set of outcomes for the conditions set by the learners. This goal is achieved by performing following actions (not in this order):

1) Evaluate solutions generated against conditions of problem.
2) Select solution of best fit.
3) Discuss panel presentations and visual aids.
4) Consider the concerns of the City Council and how to communicate effectively with them.

It is important to note that a lot of details of the scenario are not described. In the whole learning process, students pose a number of problems, and the group members negotiate on the nature of the problem, the problem statements, the hypothesis, solutions, goals' constraints, evidences, and so on. A number of decision-making methods are used in an attempt to reach consensus.

2.2 Characteristics of PBL

Generally speaking, problem based learning is active, adult-oriented, student-centered, collaborative, integrated, and interdisciplinary. It utilizes small groups and ill-structured problems, and operates in a domain context [Camp96]. In order to identify requirements for implementing a virtual PBL environment, this section discusses four important characteristics of PBL according to the scenario and literature.

2.2.1 Rich Forms of Social Interaction

In traditional instructions, learners passively and individually receive knowledge that is well organized as a series of units by teachers. Social interaction between learners is not important, because social interaction mainly takes place in a form of one way communication from teachers to students. Although all students sit in the same place, they seldom really cooperate with each other. In addition, students do homework and take examination individually in most situations. However, PBL incorporates collaborative teams in the solving of relevant problems. This method promotes student interaction and teamwork, thereby enhancing students' interpersonal skills. Observed from the scenario described in the last section, a number of small rooms are arranged for sub-groups. A group discussion room maintains a shared learning context for learners' collaboration. Social interactions occur not only in classrooms, but also beyond the boundary of classrooms, such as at home, in laboratories, and even at the pond and on the road. Because they share common learning goals, when they meet somewhere, they may do something related to learning. In PBL, students organized as sub-groups are the active agents who are responsible for the whole learning process. Teachers act as facilitators in learning processes. As described in the scenario, other people also involved in the learning process such as guest speakers, frog expert, and even citizens near the pond. Rich forms of communication and collaboration can be observed in the scenario described in the last section. In discussion rooms, the communication between teachers and students and among students are interactive. Students collaborate with each other by using tools to define problems, to generate solutions, to debate different perspectives, to conduct experiments, to write reports, and so on. Students use phones to interview experts as well. In the scenario, students collect information from identified articles, books, web sites, and other resources. Different forms of media are used such as text, picture, video, etc.

2.2.2 Ill-structured Problem

The problem learners faced are closely related to real-world problems. Real-world problems are often ill-structured and "… often require integrated instruction, which blends disciplines into thematic or problem-based pursuits, and instruction that incorporates problem-based learning and curriculum by project" [Jones94]. Observed from the scenario described in the last section, in order to understand and solve the problem of deformed frogs, it is needed to mix knowledge together from biology, chemical, environmental science, and so on. The more deeply the learners understand the problem, the more questions they have. They have to identify what learning issues they need to study and what information they should collect. As Jones et al. pointed out: "Missing information will help them understand what is occurring and help them decide what actions, if any, are required for resolution." The information and knowledge collected by the learners individually or in teams is organized and integrated around the problem to be solved. This information and knowledge will be shared by the whole learning group or sub-groups. “They see themselves and ideas as others see them, articulate their ideas to others, and are fair-minded in dealing with contradictory or conflict views” [Jones94]. They will use the knowledge they learned to construct knowledge such as hypotheses and solutions to the problem. “There is no right way or fixed formula for conducting the investigation, because each problem is unique. There may be no single ‘right’ answer” [Stepien93a]. Learners have to make decisions and provide solutions to such kind of problems. Learners will apply principles, and evidences derived from collected information resources for reasoning. Through collaborative reasoning processes, they can identify inconsistent knowledge, discover the missing knowledge, and construct shared knowledge.

2.2.3 Situated Roles

In traditional subject-based learning method, the membership and the responsibilities of a nominated role (e.g., teacher and student) are stable, often don’t change even when the situation changes. In PBL, as observed in the scenario, the responsibilities of a teacher and a student are different in different situations.

Roles of Students in PBL: In PBL, according to literatures, students take different roles in different situations.

1) Problem finders and *solvers*: Students anticipate, explore, analyze and solve problems. They can investigate causes from multiple perspectives and propose possible hypotheses and solutions.
2) Planners and producers: Students can plan and design methods and strategies for solving issues and problems that result in original products or processes.
3) Initiators and organizers: Students initiate, coordinate, and facilitate the accomplishment of collective tasks by predicting and defining intended results, deciding how they might be accomplished, anticipating roadblocks, and enlisting the support of others to achieve the results.
4) Implementers and *performers*: Students apply basic and complex skills, information, ideas, tools, and technologies to carry out the responsibilities needed to complete group or individual task that develop life skills.
5) Communicators and *negotiators*: Students can express ideas, information, intention, feeling, and concern for others in ways that are clearly understood and accepted. Students seek agreement on goals, procedures, responsibilities, and perspectives in order to accomplish tasks and goals.
6) Explorer and partners: Students explore in the physical world, materials, and technology to collect information they need. Students also interact with other people and contribute their best efforts to collaborative work.

Roles of Teachers in PBL: The principle role of the teacher in PBL has shifted from the primary role of information giver to that of facilitator and educational coach (often referred to in jargon of PBL as a "tutor"). Jones et al. summarized the roles of teachers in PBL as below [Jones94]:

1) *“Facilitator*. The teacher provides rich environments, experiences, and activities for learning by incorporating opportunities for collaborative work, problem solving, authentic tasks, and shared knowledge and responsibility.

2) *Guide*. In a collaborative classroom, the teacher must act as a guide - a complex and varied role that incorporates mediation, modeling, and coaching. When mediating student learning, the teacher frequently adjusts the level of and support based on students' needs and helps students to link new information to prior knowledge, refine their problem-solving strategies, and learn how to learn.
3) *Co-Learner and Co-Investigator*. Teachers and students participate in investigations with practicing professionals. Using this model, students explore new frontiers and become producers of knowledge in knowledge-building communities. Indeed, with the help of technology, students may become the teachers as teachers become the learners."

2.2.4 Self-directed Learning processes

Observed from the scenario, students take charge of their own learning. They define learning goals and problems that are meaningful to them. They decompose their overall learning goal into sub-goals and decide a course of actions to achieve those sub-goals. The actions are often scheduled ahead of time by the learning group itself. They allocate resources for each action. Students are required to allocate resources for each action and coordinate people, actions, and outcomes in order to accomplish goals on a predicted time schedule. Rather than all students learning the same content in the same classrooms in subject-based learning, obviously in the scenario, learners individually or in teams perform different learning actions and then share fully or partially their collected information. They take the active role to construct shared knowledge in meaningful ways according to the learning plan. On the way of implementing their own learning plan, learners must adjust their needs depending on their resources to resolve a problem, in turn, modify their plan. At the end of each action, the instructor gives feedback to each group before it can proceed with the next action. Generative instruction encourages learners to solve problems actively, conduct meaningful inquiry, engage in reflection, and build a repertoire of effective strategies for learning in diverse social contexts.

2.3 Major Implementation Requirements

As mentioned before, the goal of this research work is to establish a virtual environment for supporting collaborative PBL. Through the analysis of characteristics of PBL, implementation requirements can be identified. The remainder of this chapter discusses the major requirements for implementing a virtual collaborative PBL environment.

In order to support rich forms of social interaction, a virtual learning environment should be built like a real learning environment. When people enter the virtual learning environment, they feel the virtual learning environment still intimate. Therefore, they can exploit their experience that come from real learning environments in the virtual learning environment. In order to achieve this goal, the system should provide necessary metaphors of entities existing in a real learning environment. The relationship among these metaphors should be similar to the relationship of corresponding entities in the real learning environment. Consequently, users of the system can intuitively move in the virtual learning environment and use facilities to interact with each other in the virtual learning environment. Concretely speaking, the virtual learning environment should enable each user to know where s/he is, where s/he can go, where s/he can seek capable peers, where s/he can acquire needed information, where s/he can get and use tools (R1.1: social orientation). The virtual learning environment should enable each user to socially present her/him to others in the virtual learning environment, and to be aware of who are other users and what they are doing (R1.2: group awareness). The virtual learning environment should enable users to interact with each other in same/different time/place cooperation forms (R1.3: rich forms of social interaction). The virtual learning environment should enable users to customize their learning environments for performing different types of tasks (R1.4: customization of learning environments).

In order to support learners to solve ill structured problems and deal with a large amount of complex intelligent work, a meta-knowledge for representing the problem-oriented knowledge as a common schema is helpful for learners. A meta-knowledge can highlight essential thematic features and relationships within and among the information and knowledge around the problem to be solved. It can help to expose the inconsistency and ambiguities of the collected and constructed knowledge. It allows incremental, fine-grained representation and integration of problem-oriented knowledge. It can be used as a shared frame of reference that highlights similarities and differences between learners' points of view. It can be used as a communication tool for learners to pursue common understanding and shared knowledge. It can be used as a tool for inquiring and reasoning. It can be used as a tool for tutors to guide and coach learners. Concretely speaking, the virtual learning environment should allow the users to represent various types of information and their intention (R2.1: representation of various types of ideas). It should support users to represent the relations between the typed information (R2.2: representation of relations between ideas). It should enable users to connect and refer to related or detailed information from a piece of information (R2.3: provision of referential information). It is desired to allow users to represent different perspectives and to negotiate shared knowledge (R2.4: negotiation of shared knowledge).

In order to support situated roles, a virtual learning environment should support to explicitly specify responsibilities of collaborators and patterns of interaction to guide social interaction and reduce the contingencies. In the virtual learning environment, weak communication channels make informal coordination using social protocols difficult. That is, using social protocols in the virtual learning environment may lead to potentially unexpected interactions and unpredictable conflicts. Therefore, the system should provide a computational mechanism, which replaces the social protocol, to guide and control the behaviors of the teachers and learners. Concretely speaking, the virtual learning environment should explicitly define roles for users who have different responsibilities and obligations in a collaboration process (R3.1: definition of roles). The virtual learning environment should guide teachers and students to conduct PBL activities according a PBL strategy, in which distinct situations and transitions between these situations are specified (R3.2: provision of guidance according to PBL strategies). The virtual learning environment should synchronize collaborative activities so that users always do their focal task in each situation and unexpected behaviors are avoided (R3.3: synchronization of collaborative activities). It is desired to enable users to define their own learning strategies and to shift PBL strategies to fit changes (R3.4: shifting between PBL strategies).

In order to support self-directed learning processes, a virtual learning environment should enable learners to define their learning plans and to coordinate their actions by implementing their learning plans. However, making a good learning plan is not an easy task, because a lot of factors need to be considered, such as how to decompose the overall learning process and how to allocate resources. The system should provide support for users to do such a difficult task. Concretely speaking, the virtual learning environment should allow users to define actions and the relationships between these actions (R4.1: definition of action plans). The virtual learning environment should enable users to allocate resource (R4.2: allocation of resource). It is desired that the virtual learning environment can aid users to define their action plans (R4.3: release users' burden of making a learning plan). The virtual learning environment should enable users to coordinate their actions by executing the defined learning plans (R4.4: execution of learning plans).

This chapter identified requirements for technology support for virtual PBL environments. The next chapter will investigate whether existing virtual learning environments can meet these requirements.

3 Related Work

In the passed decade, a lot of systems for supporting education have been developed and used. Most of them were developed to realize teacher-centered learning as on-line services. That is, the knowledge to be transferred is prepared by teachers or experts before delivery. Some systems allow learners to access the prepared online information by browsing or following the predefined structure of the information. Examples are Computer-Based Training (CBT), Web-Based Training (WBT), and Education MUD/MOO. Some systems are designed with the goal to support co-authoring course materials for delivery. Some systems use artificial intelligent technology to develop intelligent agents that act as teachers or experts to teach students, such as in Intelligent Tutoring Systems (ITS). Some systems are developed based on computer-mediated communication (CMC) technology (e.g., e-mail and electronic bulletin board) and intend to replace traditional classrooms by conducting an asynchronous class, such as in the Virtual Classroom (VC) approach. Some systems enable on-line synchronous lectures by providing real-time communication channels (e.g., application sharing, chat room, shared whiteboard, and audio/video channels), such as in the Electronic Meeting System (EMS) approach. All these types of teaching and learning support systems didn't attempt to support learner-centered learning. Most requirements to support PBL can not be met by these types of systems. Therefore, these types of systems are not discussed in detail in this thesis.

This chapter provides a survey of the systems that support collaborative learning by using a learner-centered approach. In particular, the focus is on discussing computer-based learning systems for problem based learning.

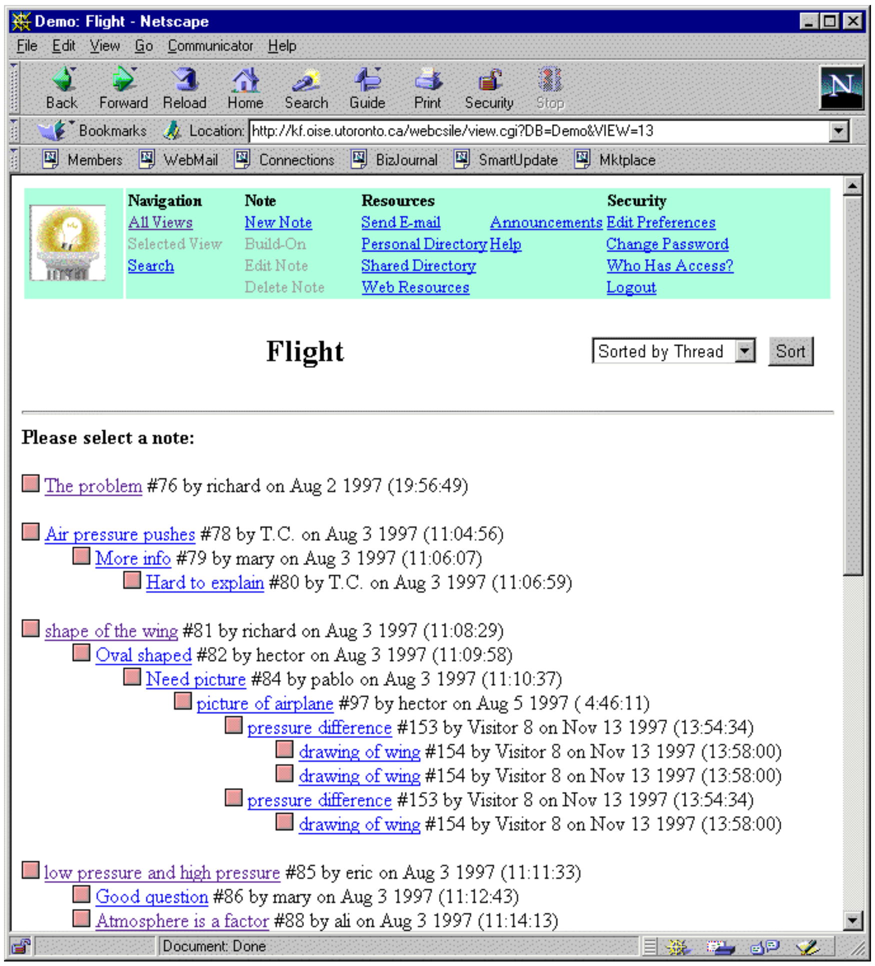
3.1 CCL

Collaborative Learning Laboratory (CCL) [Koschmann90] [Koschmann92] is a pilot PBL support system developed at the Southern Illinois University School of Medicine. This research group is also one of the pioneers of developing and practicing the PBL method. The system was developed by exploiting CSCW technology in PBL meetings. CCL is used to conduct co-located PBL classes. In a CCL class, each participant works on a networked workstation. A designated workstation is connected to a projection system so that the content of this screen is visible to all members of the group. Each participant therefore has a private screen and, in addition, all participants share a public screen, upon which to work. The projected screen serves the role of a blackboard, flip chart, and overhead projector in a more traditional face-to-face meeting. The private screen serves as an electronic desktop to be organized and used by the individual participant. The system helps learners to record, manage, retrieve information resources and maintain a shared context for PBL.

The functionality provided by CCL is essentially a shared text-based whiteboard. Except for support of synchronous collaboration, one among many forms of social interaction as required by R1.3, all other requirements identified in Chapter 2 can not be met in CLL.

3.2 CSILE

Computer Supported Intentional Learning Environments (CSILE) is a network-based system to provide across-the-curriculum support for collaborative learning and inquiry [Scardamalia89] [Scardamalia92] [Scardamalia96]. The project started in 1986 at the Ontario Institute for Studies in Education. CSILE is based on self-regulated learning (in the CSILE papers, they use the term "intentional learning") and constructivist's view of learning. Later on, new members of CSILE family have been developed such as WebCSILE, a web-based CSILE, and Knowledge Forum, a commercial version of CSILE. It emphasizes building a classroom culture supportive of active knowledge construction that can extend individual intentional learning to the group level. In CSILE, the class becomes a research team aimed at building knowledge through sustained, collaborative investigation. Information about CSILE described in this section is primarily taken from CSILE's Webpages [CSILE's Webpages].


Demo: Flight - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://kf.oise.utoronto.ca/webcsile/view.cgi?DB=Demo&VIEW=13
Members WebMail Connections BizJournal SmartUpdate Mktplace
Navigation All Views Selected View Search
Note New Note Build-On Edit Note Delete Note
Resources Send E-mail Announcements Personal Directory Help Shared Directory Web Resources
Security Edit Preferences Change Password Who Has Access? Logout
Sorted by Thread Sort
Document: Done

Flight

Please select a note:

The problem #76 by richard on Aug 2 1997 (19:56:49)

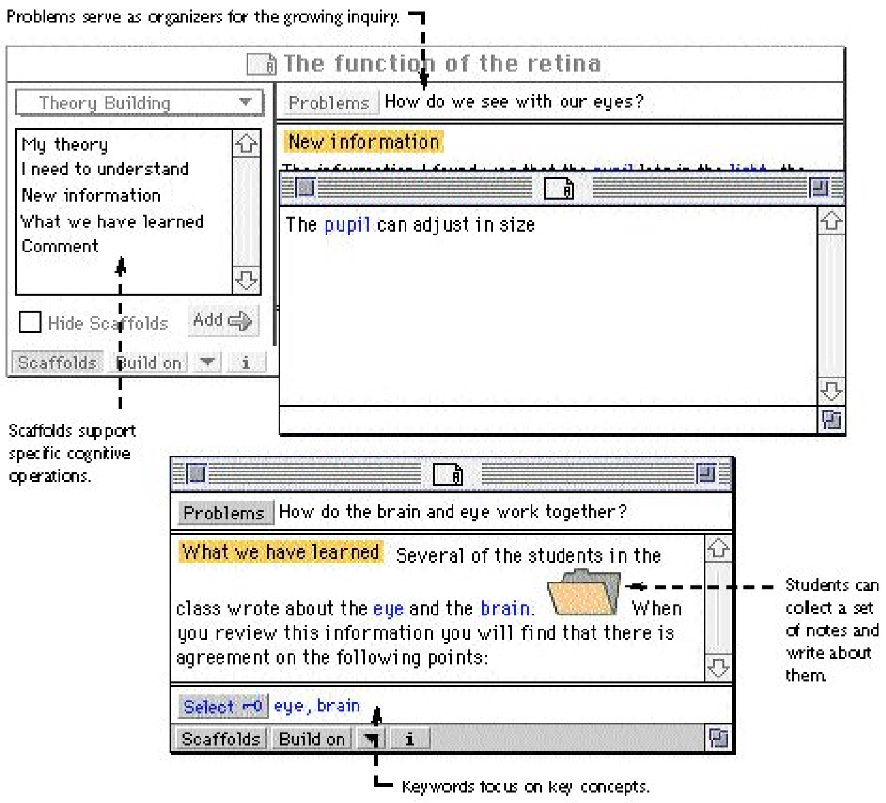
Air pressure pushes #78 by T.C. on Aug 3 1997 (11:04:56)
    More info #79 by mary on Aug 3 1997 (11:06:07)
        Hard to explain #80 by T.C. on Aug 3 1997 (11:06:59)

shape of the wing #81 by richard on Aug 3 1997 (11:08:29)
    Oval shaped #82 by hector on Aug 3 1997 (11:09:58)
        Need picture #84 by pablo on Aug 3 1997 (11:10:37)
            picture of airplane #97 by hector on Aug 5 1997 (4:46:11)
                pressure difference #153 by Visitor 8 on Nov 13 1997 (13:54:34)
                    drawing of wing #154 by Visitor 8 on Nov 13 1997 (13:58:00)
                    drawing of wing #154 by Visitor 8 on Nov 13 1997 (13:58:00)
                pressure difference #153 by Visitor 8 on Nov 13 1997 (13:54:34)
                    drawing of wing #154 by Visitor 8 on Nov 13 1997 (13:58:00)

low pressure and high pressure #85 by eric on Aug 3 1997 (11:11:33)
    Good question #86 by mary on Aug 3 1997 (11:12:43)
    Atmosphere is a factor #88 by ali on Aug 3 1997 (11:14:13)

Figure 3.1: An Example WebCSILE Public Forum
(taken from [CSILE's Webpages])

Through the use of the CSILE software, a communal database is created by students and their teachers. In a CSILE class, learners can select different communication modes (text and graphics) to generate "nodes" in a public forum (see Figure 3.1). These nodes contain ideas or information that are related to the topic under study. Nodes are available for others to comment on, leading to dialogues, and an accumulation of knowledge. A student can create links to specify a relation between any two nodes, so that the linked node becomes a reference, or a citation, in the other note [Burtis97]. All questions, theories, ideas, information, and discoveries are preserved on the database for the analysis of the entire class. CSILE provides a permanent record of the community's interactions. This eliminates the need for turn taking, allowing all learners to work on different nodes simultaneously. Specially designed scaffolds support social and cognitive operations that facilitate understanding. Students select different scaffolds to support processes such as reading difficult material, theory-building, and debate. For example, when a student creates a node for theory-building (see Figure 3.2), s/he can assign that note to one of the five predefined types called *thinking types: my theory, I need to understand, new information, what we have learned, comment* [Oshima96].


Problems serve as organizers for the growing inquiry.
The function of the retina
Theory Building
My theory
I need to understand
New information
What we have learned
Comment
Hide Scaffolds
Add
Scaffolds
Build on
i
Problems
New information
Scaffolds support specific cognitive operations.
Problems
What we have learned
Students can collect a set of notes and write about them.
Select
Scaffolds
Build on
i
Keywords focus on key concepts.

How do we see with our eyes?

The pupil can adjust in size

How do the brain and eye work together?

Several of the students in the class wrote about the eye and the brain. When you review this information you will find that there is agreement on the following points:

eye, brain

Figure 3.2: Progressive Inquiring by Using Theory-building Scaffold (taken from [CSILE's Webpages])

CSILE uses the classroom metaphor only for dividing the whole information space into sub-spaces. It can not support social orientation (R1.1 is not fulfilled). Users of CSILE can not present themselves to others in workspaces, except to input their names when creating a note. Group awareness can not be supported in CSILE (R1.2 is not fulfilled). The users of CSILE indirectly interact with each other through manipulating notes and links. Synchronous editing the same node can not be supported. Synchronous communication is allowed by providing an external chatboard. That is, CSILE support limited forms of social interaction (meeting R1.3 in a limited way). CSILE can not support end-users to customize their learning environments (R1.4 is not fulfilled). CSILE provides a scaffold with five thinking types (*my theory, I need to understand, new information, what we have learned, comment*) that allows learners to represent and categorize their intentions (R2.1 is met). Each learner can manipulate nodes and hyperlinks in the shared database (R2.3 is met), but the types of relation between nodes are not distinguished (R2.2 is not fulfilled). CSILE supports users to negotiate knowledge by creating comment nodes following discussion threads. However, it can not support users to represent different perspectives on the same information item (meeting R2.4 in a weak way). In CSILE, teacher roles and student roles are explicitly defined (R3.1 is met). The difference between these two roles is that students can only create nodes in the database on any topic that their teacher has created. However, all other requirements identied in chapter 2 can not be met in CSILE.

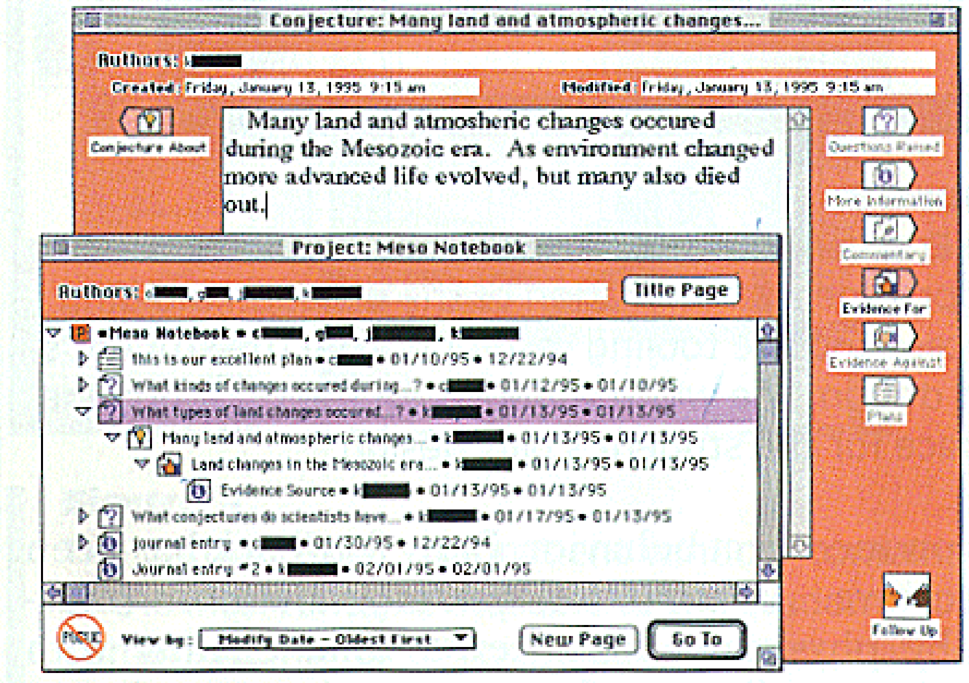
3.3 CALE

Computer Assisted Learning and Exploration (CALE) [Mahling95] is a collaborative environment to support problem based learning by discovery and exploration. According to [Mahling95], CALE acts as a multi-media repository for case materials and manages the structured group access to those documents and user generated information. CALE supports collaborative learning, exploration of medical simulated patient cases edited by facilitators, and access to reference materials. By using the system, students can access shared and individual notes by opening tools from their principal control panel. Text-boxes (text-based editor) are used by students to take notes, as well as to answer questions. The central blackboard acts as the coordination center for the students. This gives structure to discussion and allows individual students to connect observations made in an asynchronous session to the overall learning effort. CALE allows hyperlinks from the document where the observation was made to the central blackboard. Thus local information and central coordination are achieved. The blackboard follows the PBL example of dividing information into three separate classes: *observed facts, hypothesis*, and *need more information*. An item in the *need more information* category could be turned into an action item, which in turn would be assigned to a team member with a due date. CALE keeps track of these commitments and thus allows students to structure the learning task. With the case presentation shell being the same for all cases, new cases can easily be added. The case designer has control for the case material repository and specifies access control for the case material.

In CALE there is no place metaphor where people can navigate and present themselves to others (R1.1 and R1.2 are not fulfilled). People can use a blackboard tool to conduct collaborative learning activities and can use a Text-box to communicate with each other (meeting R1.3 in a limited way). Users can not customize the learning environment for performing different types of activities (R1.4 is not fulfilled). CALE supports students to create notes, to connect individual notes to the shared document, to explore case materials, and access reference materials following the document structure. The notes are categorized (R2.1 is fulfilled), but there exists only a referral relation between them (R2.2 is not fulfilled, but R2.3 is met). CALE supports negotiation of knowledge by providing discussion threads (meeting R2.4 in a weak way). In CALE two roles are defined (R3.1 is fulfilled). Teachers can edit case material and specify access control. Students can only read case material. CALE can not provide computational support for learning strategies (R3.2, R3.3, and R3.4 are not fulfilled). CALE helps students to create an action plan in which the action items are isolated (meeting R4.1 in a weak way). Each action can be assigned to a team member with a due date (meeting R4.2 in a weak way). CALE provides limited support to aid users to create action plans by turning each n*eed more information* item into an action item (meeting R4.3 in a limited way). However, the action plan is not a computational process description that can not be executed automatically (R4.4 is not fulfilled).

3.4 CNB

Collaboratory Notebook (CNB) [Edelson95] is a networked, multimedia tool. This software tool is created in the Collaborative Visualization (CoVis) project initiated at Northwestern University in Chicago. The objective of this project is to provide a distributed multimedia learning environment that supports learning-in-doing [Pea93] [Edelson96a]. The proposed system is intended to transform science learning to better resemble the authentic practice of science. Information about CNB described in this section is primarily taken from CoVis's Webpages [CoVis Webpages].



Conjecture: Many land and atmospheric changes...

Authors: k[redacted]

Created: Friday, January 13, 1995 9:15 am Modified: Friday, January 13, 1995 9:15 am

Many land and atmosheric changes occured during the Mesozoic era. As environment changed more advanced life evolved, but many also died out.

Project: Meso Notebook

Authors: c[redacted], g[redacted], j[redacted], k[redacted]

```
▽ •Meso Notebook • c[redacted], g[redacted], j[redacted], k[redacted]
  ▷ this is our excellent plan • c[redacted] • 01/10/95 • 12/22/94
  ▷ What kinds of changes occured during...? • c[redacted] • 01/12/95 • 01/10/95
  ▽ What types of land changes occured...? • k[redacted] • 01/13/95 • 01/13/95
    ▽ Many land and atmospheric changes... • k[redacted] • 01/13/95 • 01/13/95
      ▽ Land changes in the Mesozoic era... • k[redacted] • 01/13/95 • 01/13/95
          Evidence Source • k[redacted] • 01/13/95 • 01/13/95
  ▷ What conjectures do scientists have... • k[redacted] • 01/17/95 • 01/13/95
  ▷ Journal entry • c[redacted] • 01/30/95 • 12/22/94
    Journal entry #2 • k[redacted] • 02/01/95 • 02/01/95
```

View by: Modify Date – Oldest First

Figure 3.3: The Table of Contents of a Notebook and the Content of a Page (taken from [Edelson96b])

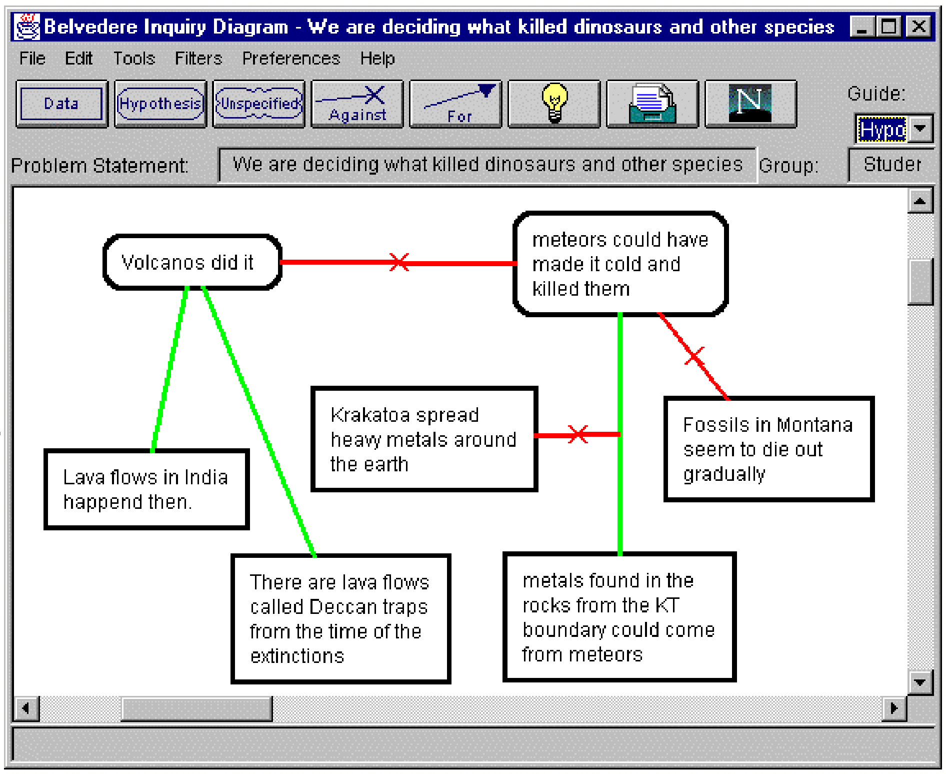
CNB has been designed to scaffold students as they learn to conduct open-ended inquires in a collaborative context across the boundaries of time and space. A primary function of CNB is to allow teachers to monitor and guide students' process of learning. It emphasizes the learning process instead of learning outcomes. The software is based on the metaphor of the scientist's laboratory notebook, with a bookshelf, notebooks and pages being the primary interface elements (see Figure 3.3). It extends this metaphor with facilities for collaborators anywhere on the Internet to share and co-author inquiry. In a notebook, each page has a type that provides a description of its content and a description of the relationships to other pages. The page types available are *question, conjecture, evidence for, evidence against, information, commentary, plan*, and *step in plan* [Edelson94] [Edelson96b].

CNB organizes the shared information space by using the bookshelf, notebook and page metaphors. Except to record users' names when the users create a notebook or a page, CNB does not provide support for social orientation, group awareness, and customization of learning environment (R1.1, R1.2, and R1.4 are not fulfilled). Users of CNB indirectly interact with each other through manipulating notebook and page. There is no other means that can be used to interact with each other (meeting R1.3 in a weak way). CNB provides eight page types (*question, conjecture, evidence for, evidence against, information, commentary, plan*, and *step in plan*) (R2.1 is met). As shown in the background window of the Figure 3.3, a page can be connected by hyperlinks to other pages that may have different page types. These pages are connected by hyperlinks forming a discussion structure (R2.2 is not met and R2.3 is met). CNB supports users to negotiate knowledge by creating different types of pages to represent their perspectives following discussion threads (meeting R2.4 in a weak way). Users can manipulate notebooks and pages jointly if they get permission from the owners of the pages. There are no explicitly defined roles (R3.1 is not met). There is no computational support for learning strategies in CNB (R3.2, R3.3, and R3.4 are not fulfilled). In addition, CNB allows users to define a plan or a step in a plan as a page, but the plan and the step of a plan serve as a common understanding about what actions should be done in an inquiring process (meeting R4.1 in a weak way). No support is provided to help users to assign resources to actions (R4.2 is not met), to define and modify learning plans (R4.3 is not met), and to coordinate actions by executing the defined learning plan (R4.4 is not met).

3.5 Belvedere

The Belvedere software [Suthers95] [Suthers97] was developed at the Laboratory for Interactive Learning Technologies at the University of Hawai'i at Manoa. It is designed to support problem-based collaborative learning scenarios in which middle school and high-school students learn critical inquiry skills that they can apply in everyday life as well as in science. Information about Belvedere described in this section is primarily taken from Belvedere's Webpages [Belvedere Webpages]. According to [Belvedere Webpages], Belvedere's core functionality is a shared workspace for constructing "inquiry diagrams," which relate data and hypotheses by evidential relations (consistency and inconsistency). The software also includes artificial intelligence coaches that provide advice, a "chat" facility for unstructured discussions and facilities for integrated use with Web browsers.

Belvedere Inquiry Diagrams [Suthers97] are designed to help students express graphically how ideas are connected. These ideas can come from scientific articles, or they can come from students' own knowledge, experiments, and research. Students can construct their own diagrams by using a number of shapes for representing different types of statements and links for representing different kinds of relationships between these statements. Figure 3.4 shows the user interface of the tool and illustrates an example of inquiry diagrams. When students solve a problem or explain something, Belvedere helps them keep track of the ideas by displaying them graphically. If students investigate a scientific question, Belvedere can use their diagram to give them ideas about what to do next. An intelligent agent, called a Coach, provides students with suggestions on how to use the software through five "phases of inquiry" (explore, hypothesize, investigate, evaluate, and report). It checks the diagram to investigate what have been found out so far. Then it makes suggestions for what to consider next. For example, if no empirical data has been offered in support of a hypothesis, the Coach will highlight the hypothesis and asks whether the students can find a way to support it or show that it predicts or explains the phenomenon under discussion. If only one hypothesis has been stated in the discussion, the Coach will point out that scientists compare alternative explanations, and asks whether another hypothesis might explain the same data [Suthers99b].



We are deciding what killed dinosaurs and other species

Volcanos did it

meteors could have made it cold and killed them

Krakatoa spread heavy metals around the earth

Fossils in Montana seem to die out gradually

Lava flows in India happend then.

There are lava flows called Deccan traps from the time of the extinctions

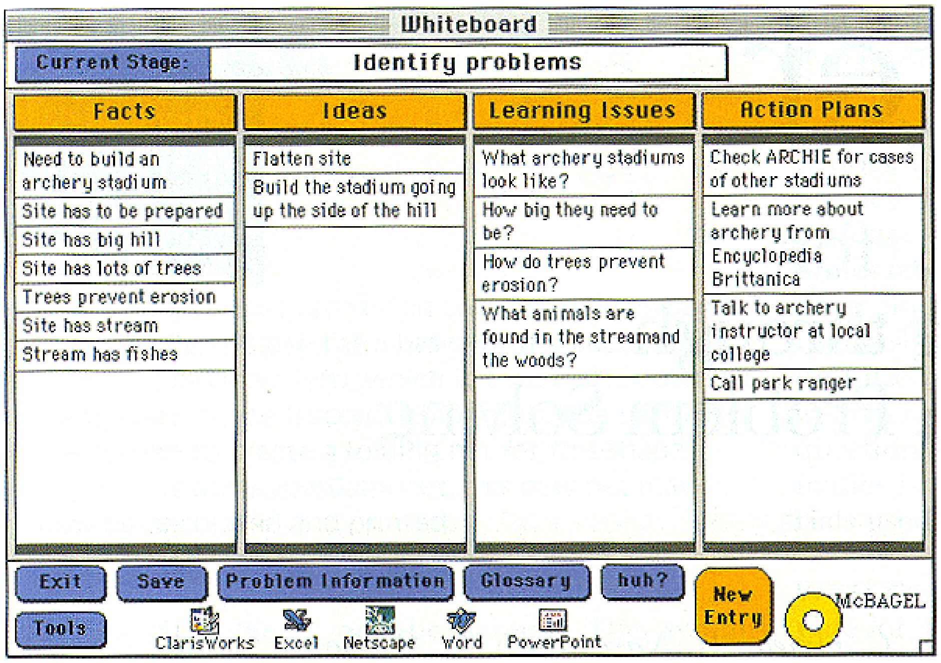
metals found in the rocks from the KT boundary could come from meteors

Figure 3.4: Belvedere Inquiry Diagram
(taken from [Belvedere Webpages])

In Belvedere, social orientation and group awareness can not be supported (R1.1 and R1.2 are not fulfilled). Students can not socially present themselves to others in the learning environment. Belvedere support students to construct inquiry diagrams synchronously or asynchronously. The chat tool can be used to communicate with each other in the form of unstructured text-based discussion (meeting R1.3 in a limited way). Users can not customize the learning environment for performing different types of activities (R1.4 is not fulfilled). Students can create different shapes for representing different types of ideas such as *data* and *hypothesis* (R2.1 is fulfilled). Students can also create different types of links to represent the relationship between their ideas such as *against* and *for*. However, the types of relation between ideas are not sufficient to support the whole process of PBL (meeting R2.2 in a weak way). In addition, more detail information about an idea can not be provided by using the tool (R2.3 is not fulfilled). Different perspectives can be represented as separate ideas (meeting R2.4 in a weak way). Belvedere did not explicitly define different roles (R3.1 is not fulfilled). Five phases of inquiry are distinguished and what activities should be done in each phase is suggested according to the current state of the diagram constructed (R3.2 is fulfilled). However, the tool does not support to synchronize collaborative learning activities and to avoid unexpected operations in any state (R3.3 is not fulfilled). Belvedere's learning strategy is defined by the software developers, and PBL strategies can not be defined and modified by end-users to fit their situations. Therefore, it is impossible to shift learning strategies (R3.4 is not fulfilled). Belvedere does not provide any mechanisms for students to define their own action plans and to execute action plans (R4.1, R4.2, R4.3, and R4.4 are not fulfilled).

3.6 McBAGEL

The McBAGEL system [Narayanan95] [Guzdial96] was developed at the EduTech Institute at Georgia Institute of Technology, which is a multi-disciplinary research organization committed to enhancing science, math and design education through innovative uses of technology. Their research efforts are aimed at creating environments for learning, both embodied and virtual, that reflect the knowledge of cognition behind learning, complex problem solving and understanding.



Identify problems

Facts	Ideas	Learning Issues	Action Plans
Need to build an archery stadium	Flatten site	What archery stadiums look like?	Check ARCHIE for cases of other stadiums
Site has to be prepared	Build the stadium going up the side of the hill	How big they need to be?	Learn more about archery from Encyclopedia Brittanica
Site has big hill		How do trees prevent erosion?	Talk to archery instructor at local college
Site has lots of trees		What animals are found in the streamand the woods?	Call park ranger
Trees prevent erosion			
Site has stream			
Stream has fishes			

Figure 3.5: The McBAGEL's Whiteboard with Four Sub-spaces
(taken from [Guzdial96])

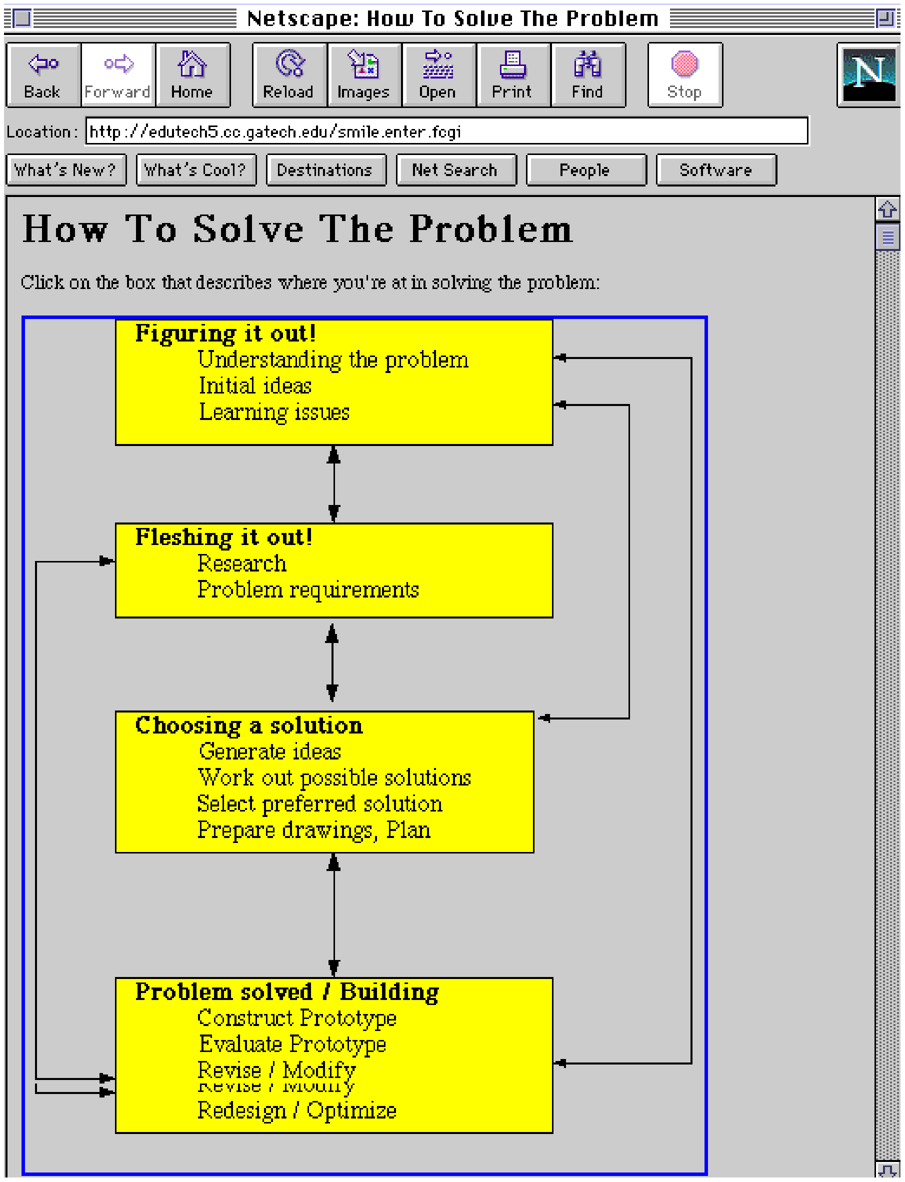
Information about McBAGEL described in this section is primarily taken from EduTech's Webpages. According to [EduTech Webpages], McBAGEL arose out of the synthesis of PBL and Case-based reasoning (CBR) [Kolodner93]. The PBL whiteboard of McBAGEL helps scaffold the students' problem solving by communicating the PBL process as well as serving as an external memory aid. This software provides an electronic workspace that is split into four regions (see Figure 3.5). Students, working in small groups, enter the environment at their shared electronic workspace. They are provided with relevant information on the design problem they need to solve via the button "problem information." As students are initially formulating and understanding the problem, they will be encouraged to identify data relevant to the problem from the information they have been provided with, and to articulate this by recording those in the "facts" space. Similarly, as they consider alternative solutions, they will make use of the "ideas" space. The problem-based learning methodology that this environment embodies explicitly prepares students for self-directed learning by requiring them to identify their knowledge deficiencies in the "learning issues" space and the actions they plan to take to remedy those deficiencies in the "action plan" space. Several buttons are found on the bottom of the screen, which provide access to different tools (e.g., case libraries) that they will need to solve the problem. The column titles offer pull-down help to provide examples of the kinds of information that would be entered into each column.

McBAGEL can not support social orientation (R1.1 is not fulfilled) and group awareness (R1.2 is not fulfilled). It enables students to manipulate different items synchronously (meeting R1.3 in a limited way). Users can not customize the learning environment for performing different types of activities (R1.4 is not fulfilled). McBAGEL supports students to manipulate information items in the shared whiteboard. The information items are categorized (R2.1 is fulfilled), but there exists neither any relation between them (R2.2 is not fulfilled) nor additional information pages connected with these items (R2.3 is not fulfilled). McBAGEL does not support negotiation of knowledge, but it enables users to represent different perspectives by formulating different information items (meeting R2.4 in a weak way). McBAGEL does not define any role (R3.1 is not fulfilled). McBAGEL identifies different phases and enables users to view the current phase (meeting R3.2 in a weak way). Users should behave appropriately to fit the current situation. However, the system does not provide computational support to define, execute and change learning strategies (R3.3 and R3.4 are not fulfilled). McBAGEL allows students to propose actions, but the proposed actions are represented as a list of isolated items and serve to establish mutual understanding of their future work (meeting R4.1 in a weak way). It is impossible to assign resources to an action in a computational form (R4.2 is not fulfilled). McBAGEL does not provide any help for users to make action plans (R4.3 is not fulfilled). Furthermore, the action list is not a computational process description and thus can not be executed automatically (R4.4 is not fulfilled).

3.7 Web-SMILE

Web-SMILE [Guzdial97] [Kolodner98] is a software tool for the learning-by-design curriculum of middle school projects. This software tool was developed at the EduTech Institute at Georgia Institute of Technology as well. Information about Web-SMILE described in this section is primarily taken from EduTech's Webpages

[EduTech Webpages]. Web-SMILE's support of problem based learning evolved from McBAGEL and its support of asynchronous collaboration inherited from CaMILE [Hmelo95] (stands for "Collaborative and Multimedia Interactive Learning Environment"). In Web-SMILE, the McBAGEL 's action column has been removed. Like CaMILE, Web-SMILE can also scaffold collaboration by providing a structure (asynchronous threaded discussions, where notes directly commenting upon another are displayed as related). Web-SMILE supports tight integration with multimedia. Students can link a variety of forms of media into their notes on the Web. This enables the creation of single-click access from a Web page anchor (e.g., a design report to discuss or an exam review question) to a thread for discussion of that anchor. In addition, Web-SMILE explicitly used the steps in the problem solving process to guide the use of the integrated tools (see Figure 3.6).


Netscape: How To Solve The Problem

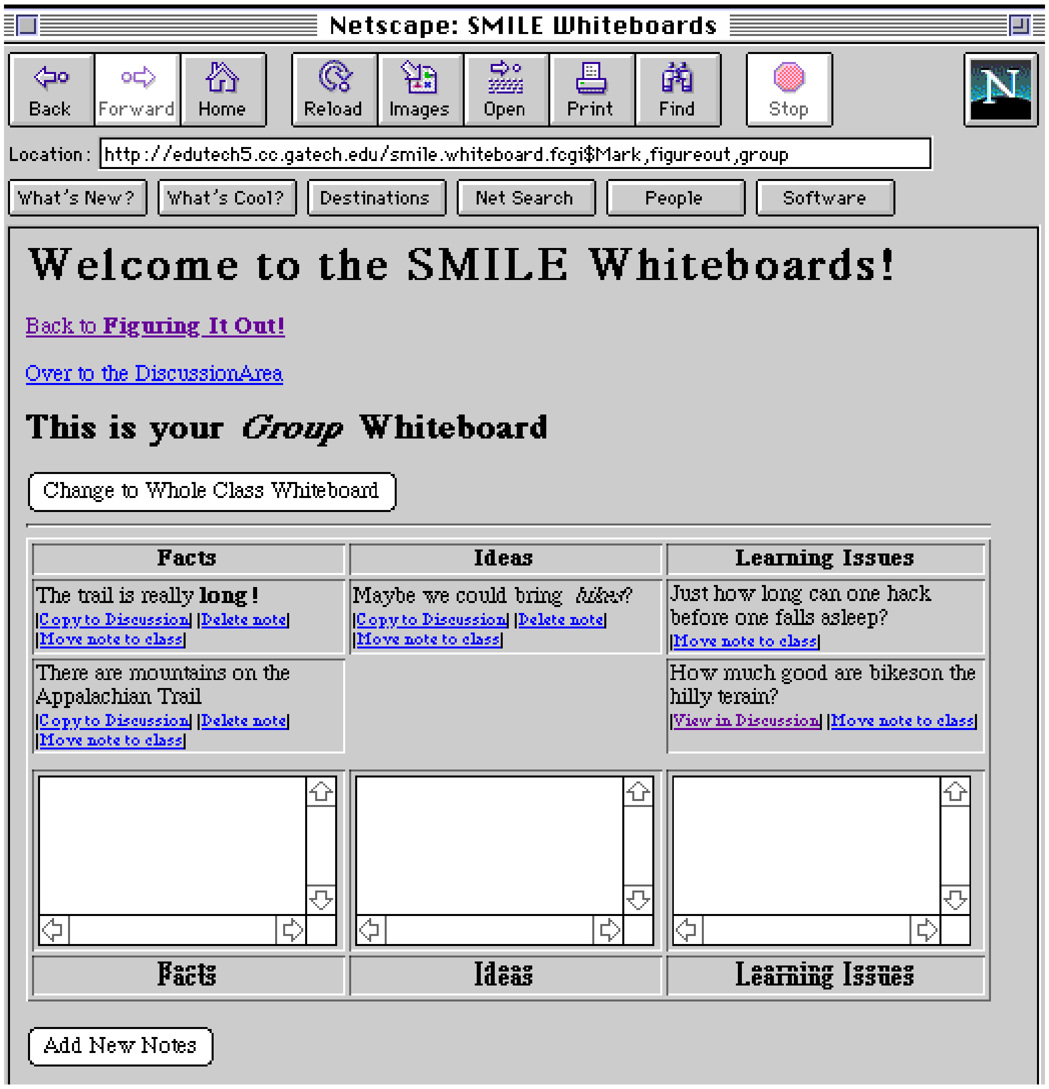
Location: http://edutech5.cc.gatech.edu/smile.enter.fcgi

How To Solve The Problem

Click on the box that describes where you're at in solving the problem:

Figure 3.6: The Problem-solving Flowchart in Web-SMILE
(taken from [EduTech Webpages])

The students are able to access the "whiteboard" to record their facts, ideas and learning issues, and the "discussion area" to share the results of research or discuss a solution (see Figure 3.7).



http://edutech5.cc.gatech.edu/smile.whiteboard.fcgi$Mark,figureout,group

Welcome to the SMILE Whiteboards!

Back to **Figuring It Out!**

Over to the DiscussionArea

This is your *Group* Whiteboard

Change to Whole Class Whiteboard

Facts	Ideas	Learning Issues
The trail is really **long!** |Copy to Discussion| |Delete note| |Move note to class|	Maybe we could bring *bikes?* |Copy to Discussion| |Delete note| |Move note to class|	Just how long can one hack before one falls asleep? |Move note to class|
There are mountains on the Appalachian Trail |Copy to Discussion| |Delete note| |Move note to class|		How much good are bikeson the hilly terain? |View in Discussion| |Move note to class|

Figure 3.7: Web-SMILE's Whiteboard
(taken from [EduTech Webpages])

There is no need to copy data recorded in one area to be pasted in another; the student can simply click the "move" option to place a whiteboard idea into the discussion area. This integration of tools is part of the process-oriented view of student support in Web-SMILE. Instead of prompting for a tool's use, the environment asks, "Where are you in solving the problem?" Based on the selected step of the process, activities are suggested, with the tools accessible by a single click. There is also procedural facilitation of note types and their associated starter text for that type note (see Figure 3.8).

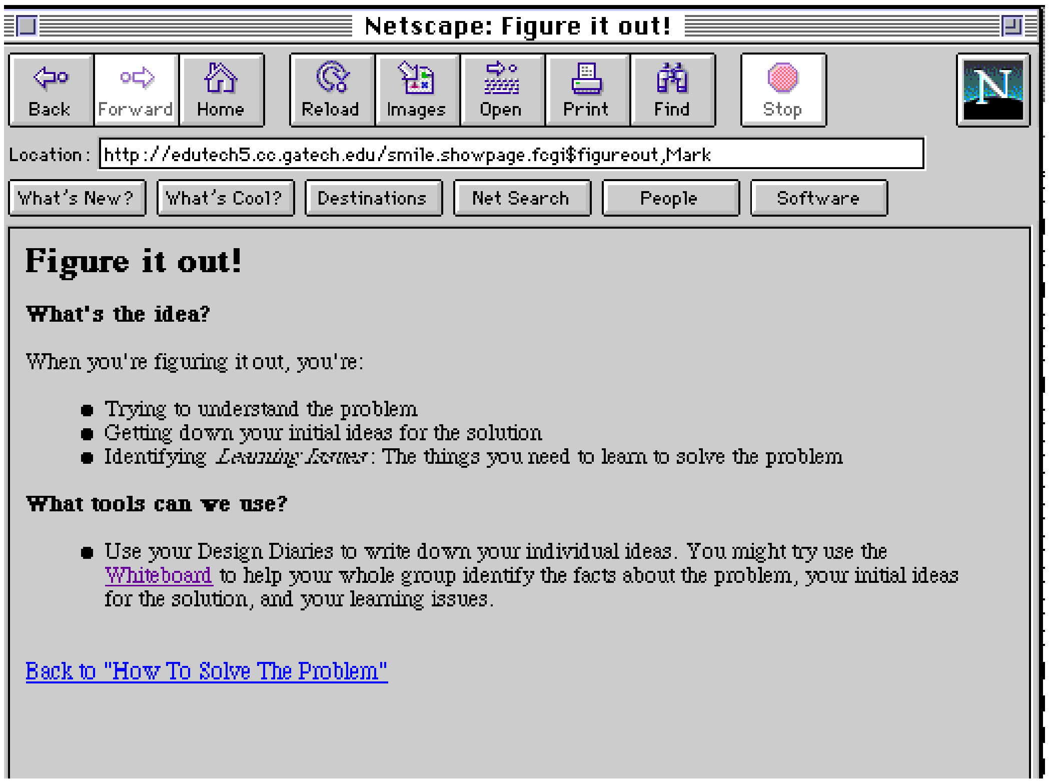


Figure it out!

What's the idea?

When you're figuring it out, you're:

- Trying to understand the problem
- Getting down your initial ideas for the solution
- Identifying *Learning Issues*: The things you need to learn to solve the problem

What tools can we use?

- Use your Design Diaries to write down your individual ideas. You might try use the Whiteboard to help your whole group identify the facts about the problem, your initial ideas for the solution, and your learning issues.

Back to "How To Solve The Problem"

Figure 3.8: Guidance Information for a Stage of the Process
(taken from [EduTech Webpages])

Web-SMILE does not support social orientation and group awareness (R1.1 and R1.2 are not fulfilled). It supports synchronous collaboration in a shared whiteboard with multiple sub-spaces and asynchronous collaboration in consequent threads of discussion (meeting R1.3 in a limited way). It does not support customization of learning environment (R1.4 is not fulfilled). Web-SMILE supports students to create three types of information items as notes (R2.1 is fulfilled). A note is directly addressable as a hyperlink on the Web and more detailed information can be provided on the Web page connected by the hyperlink (R2.3 is fulfilled). However, the relation between these types of information items is not explicitly defined (R2.2 is not fulfilled). Different perspectives can be represented as separate items in the shared whiteboard or as separate notes in the threads of discussion (meeting R2.4 in a weak way). Web-SMILE does not take care of users with different roles (R3.1 is not fulfilled). In Web-SMILE, scaffolding enables users to select a step of the process and then provides guideline (e.g., suggesting what activities should be performed in each step) (R3.2 is fulfilled). However, scaffolding can not be used to control social interaction and users can work in different steps at the same point of time (R3.3 is not fulfilled). Web-SMILE provides a unique learning strategy to perform PBL and can not support to define and shift learning strategies (R3.4 is not fulfilled). Users of Web-SMILE can not define action plans (R4.1 is not fulfilled) and assign resources to actions (R4.2 is not fulfilled). There is no support to aid users to make a plan (R4.3 is not fulfilled) and to coordinate these actions by executing the defined plan (R4.4 is not fulfilled).

3.8 Analysis of the State of the Art

One can observe that all these systems were developed by adopting an information-sharing approach. Thus, they didn't consider the social orientation problem (R1.1). The user of the systems can not present herself/himself to others in the virtual learning environments. The user can not know with whom s/he collaborates directly from these systems (R1.2). In fact, most of these systems (e.g., CALE, CSILE, CNB, and Web-SMILE) purposely support large and loosely organized user groups such as a Web user community. In this case, intentional and scheduled collaborative learning activities can not be supported by the systems. Learners collaborate with each other by chance when they manipulate the same note or page. Other systems (e.g., CCL, McBAGEL, and Belvedere) are normally used in a co-located collaborative mode. In this case, social orientation and group awareness are supported outside systems. Users only need to concentrate on substantive work. When conducting problem based learning like in the scenario described in the last chapter by geographically distributed people, social orientation and group awareness is a serious problem when using these systems. All systems can support communication and collaboration by providing some tools. However, rich forms of communication and collaboration required in the scenario can not be fully supported by these systems (R1.3). In these systems, functionality and the user interfaces are fixed and the users of these systems have to use them in the way that system developers designed. Thus, the learning environments can not be customized by users to fit different learning activities (R1.4).

All systems except for CCL enable users to categorize their ideas when they collaboratively construct their shared knowledge (R2.1). Only Belvedere allows users to represent relationships between the ideas, but the types of ideas and the types of relation between these ideas are not sufficient to support the whole process of PBL (R2.2). Other systems support users to organize their ideas by using discussion threads. Some systems (e.g., CALE, CSILE, CNB, and Web-SMILE) allow users to provide detail information and connect it to ideas, while other systems do not support this (R2.3). All systems support negotiation of knowledge to some extent. Representing different perspectives is supported in these systems by creating separate statements (R2.4).

Most of these systems have explicitly defined teacher role and student role. However, the membership of a role and its responsibilities are defined rigidly. That is, when a user with a given role registers in the system, s/he can not change his/her role and the responsibilities of the role can not be changed in different situations (R3.1). Among these systems, McBAGEL, Web-SMILE and Belvedere define distinct phases and provide guidance to perform the focal task in each phase (R3.2). However, these systems can not control social interaction according to the current working phase (R3.3). Users can follow the guidance or not. In fact, these systems do not attempt to support a synchronized group activity. In other words, different users can work in different phases at the same point of time. Changing a phase of a user does not influence other users' working phase. In addition, each system has a unique collaboration strategy and it is not allowed for end-users to define, modify, and change learning strategies (R3.4).

Some systems (e.g., CNB, CALE and McBAGEL) support users to define actions as a set of isolated items, but the purpose of defining actions in these systems is to support mutual understanding about future work (R4.1). In these systems action items are simply defined as a statement or a commitment. A lot of information necessary for triggering actions is not specified, such as conditions for starting and finishing actions and allocated resources (R4.2). Therefore, the action plan defined in this way can not be executed by the system (R4.4). CALE can help users to define actions by turning "need more information" items into action items. However, such primitive support is not sufficient for developing an executable learning plan (R4.4). The reason why these systems provide insufficient support for planning is that these systems are designed mainly for supporting science inquiry in middle- or high schools. The main intention of this thesis work is to support adult learners mostly in professional training.

We conclude the discussion by comparing these systems respectively with respect to support for the requirements identified in the last chapter. The identified requirements are summarized as below.

(R1.1): support social orientation
(R1.2): support group awareness
(R1.3): support rich forms of social interaction
(R1.4): support customization of learning environments

(R2.1): support representation of various types of ideas
(R2.2): support representation of relations between ideas
(R2.3): support provision of referential information
(R2.4): support negotiation of shared knowledge

(R3.1): support definition of roles
(R3.2): provision of guidance according to PBL strategies
(R3.3): support synchronization of collaborative activities
(R3.4): support shifting between PBL strategies

(R4.1): support definition of action plans
(R4.2): support allocation of resources.
(R4.3): release users' burden to make action plans
(R4.4): support execution of action plans

	CCL	CSILE	CALE	CNB	Belvedere	McBAGEL	Web-SMILE
R1.1	∅	∅	∅	∅	∅	∅	∅
R1.2	∅	∅	∅	∅	∅	∅	∅
R1.3	-	-	-	-	-	-	-
R1.4	∅	∅	∅	∅	∅	∅	∅
R2.1	∅	+	+	+	+	+	+
R2.2	∅	∅	∅	∅	-	∅	∅
R2.3	∅	+	+	+	∅	∅	+
R2.4	∅	-	-	-	-	-	-
R3.1	∅	+	+	∅	∅	∅	∅
R3.2	∅	∅	∅	∅	+	-	+
R3.3	∅	∅	∅	∅	∅	∅	∅
R3.4	∅	∅	∅	∅	∅	∅	∅

R4.1	∅	∅	-	-	∅	-	∅
R4.2	∅	∅	-	∅	∅	∅	∅
R4.3	∅	∅	-	∅	∅	∅	∅
R4.4	∅	∅	∅	∅	∅	∅	∅

Table 3.1: Comparison of Existing PBL Support Systems

Table 3.1 summarizes to what degree these systems fulfill the requirements presented in the last chapter. The notion "∅" in the table 3.1 indicates "no support". The notion "+" indicates "full support". The notion "-" means "partial fulfill" or "weak support".

3.9 Summary

This chapter described related work and analyzed the features of existing virtual PBL environments according to the requirements identified in the last chapter. We summarize deficits of existing virtual PBL environments regarding the objectives of the thesis.

There are 6 requirements not addressed by existing systems. They are R1.1: support social orientation; R1.2: support group awareness; R1.4: support customization of learning environments; R3.3: support synchronization of collaborative activities; R3.4: support shifting between PBL strategies; and R4.4: support execution of action plans. The main reason why existing systems have not addressed these requirements is that most of them aim at supporting asynchronous collaboration among distributed users while other systems aim at supporting co-located synchronous collaborative learning. This thesis aims at supporting geographically distributed and co-located people to conduct collaborative PBL activities in a series of organized synchronous and asynchronous sessions. In order to achieve this goal, this thesis work will contribute solutions to provide computerized context of collaborative learning and computational coordination mechanisms.

There are 6 requirements where existing systems offer only weak or limited support. They are R1.3: support rich forms of social interaction; R2.2: support representation of relations between ideas; R2.4: support negotiation of shared knowledge; R4.1: support definition of action plans; R4.2: support allocation of resources; R4.3: release users' burden to make action plans. This thesis tries to suggest better and more integrated solutions to meet these requirements.

The other requirements are sufficiently supported by one or many existing systems. This thesis will adopt or improve the existing solutions to meet these requirements, too.

4 Approach

This Chapter presents the approach adopted in this thesis. Based on activity theory, a conceptual framework for the design of virtual PBL environments is developed. According to this conceptual framework, a conceptual architecture of a virtual PBL environment is designed. The conceptual architecture has four modules. The detailed design of these four modules is described informally and specified formally by means of Z language in this Chapter.

4.1 Overview of the Approach

This section outlines the general approach adopted in this thesis to develop a computer supported collaborative problem-based learning environment. In this chapter, first of all, a brief introduction to activity theory is presented. Activity theory emphasizes the mediational role of cultural factors (artifacts, tools, and language) and social factors (conventions, division of labor and established procedures) in an activity system. Viewed from the perspective of activity theory, a problem-based learning activity is an activity system in which the cultural and social characteristics are different from those of traditional subject based learning. Such distinction should be reflected in a computer-based learning environment for PBL. Based on this view, a conceptual framework for the design of virtual problem based learning environments is proposed. When presenting this conceptual framework, important design issues and possible design choices are discussed. Following the guideline of this conceptual framework, a virtual PBL environment is designed, in which cultural and social mediation in the PBL activity is supported by

1) the *virtual institute metaphor* (for inheriting the culture of real learning environments),
2) the *PBL-net* (for mediating the construction of shared knowledge),
3) the *PBL-protocol* (for mediating the interaction between the community members), and
4) the *PBL-plan* (for mediating the contributions made by different members).

4.1.1 Perspective of Activity Theory

The activity theory was formulated in the 1930's by a group of Russian psychologists. It is a philosophical framework that allows the study of different forms of human activities. In its simplest forms, an activity is defined as the engagement of a subject toward a certain goal or objective. As the founder of this theory, Vygotsky created the idea of mediation and claimed that human activities are mediated by instruments such as tools and language [Vygotsky78]. Instruments are created by people and effect control over behavior. They have an associated culture and history and permanence that exists across time and space. Leontiev further suggested that activities are also mediated by other human beings and social relations [Leontiev47]. The activity of the individual is not viewed in isolation, but is tied to the social context.

Engestroem [Engestroem87] proposed the structure of human activity illustrated in Figure 4.1. In this model, the *subject* refers to the individual or sub-group whose agency is chosen as the point of view in the analysis. The *object* refers to the 'raw material' or 'problem space' at which the activity is directed and which is molded and transformed into *outcomes*. The *instrument* refers to all the means, which the subjects have at their disposal for influencing the object and for achieving the goals. Instruments include both tools (e.g., hammer and pen) and signs (e.g., language and symbol). The *community* comprises multiple individuals and/or sub-groups who share the same general object. The *rules* refer to the explicit and implicit society- and community level laws, standards, norms, policies, and strategies that constrain actions and interactions within the activity system. The *division of labor* refers to both the horizontal division of tasks between the members of the community and to the vertical division of power and status.

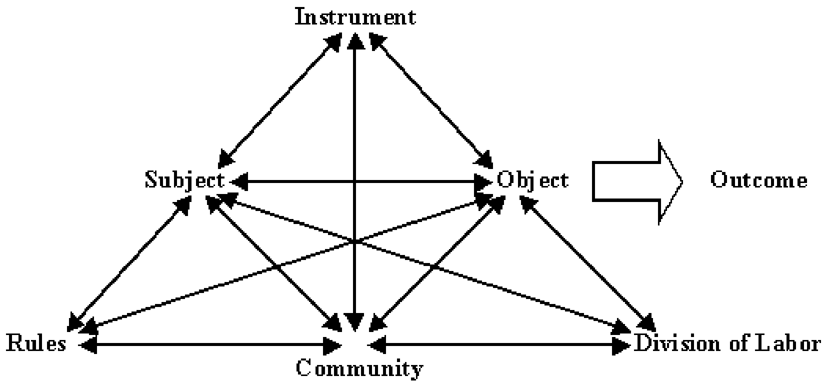
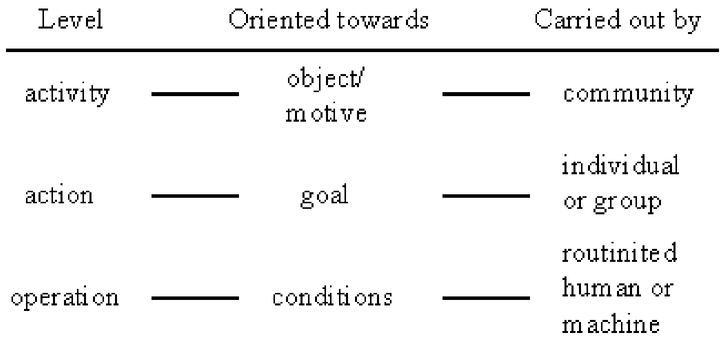


Figure 4.1: The Structure of Human Activity
(taken from [Engestroem87])

Human activity can be described as a hierarchy structure. The three-level structure of activity proposed by Leontiev is depicted in Figure 4.2. Human activity is driven by an object-related motive and carried out by a community. The activity consists of actions or chains of actions. An action is driven by a conscious goal and performed by individual (or group). Actions are realized through operations, which are driven by the conditions of the concrete situation and is related to routinized behaviors performed automatically.



Level	Oriented towards	Carried out by
activity	object/ motive	community
action	goal	individual or group
operation	conditions	routinited human or machine

Figure 4.2: The Hierarchical Structure of Activity
(taken from [The Activity System])

Activity theory provides a number of useful concepts that can be used to analyze problem based learning activities and to create a conceptual framework for the design of virtual PBL environments.

4.1.2 An Analysis of Problem Based Learning according to Activity Theory

In this section, the PBL activity is analyzed from the perspective of activity theory.

PBL is different from traditional instructional methods. In traditional subject based learning methods, the focus is on the content. For each content area, experts and teachers divide the topics into small, manageable bundles, and transfer those to students according to a prescribed lesson plan. Students passively receive knowledge piece by piece individually. In PBL, the focus is on the learner and authentic problems [Norman96]. Guided by tutors, who just take a facilitative role, learners engage in active and meaningful learning, normally in cooperative forms. The subject of a PBL activity is a group of learners, rather than teachers. The object of a PBL activity is a problem under study. The expected outcome of a PBL activity is (1) acquiring knowledge and skills that can be transferred to solve similar problems on an individual level, and (2) constructing a shared knowledge and promoting mutual understanding on the group level. From the scenario described in the Chapter 2, we can observe that the instruments used in a PBL activity are tools (such as blackboards and domain-specific tools like experimental instruments), places (such as discussing rooms, library, and laboratory), documents (such as learning materials and learning records), and jargon that facilitates pursuing mutual understanding and constructing shared knowledge. The community of a PBL activity is broad and consists of the people who are involved in or have influence on the activity in some forms. These people may have some expertise, may have similar learning interests, may be able to provide assistance, and may learn or work in the same organization. In PBL, learners may have different expertise and different learning interests and the problem to be solved may be complex. Not all learners study the same topic together and to the same degree. A division of labor within the learning group is necessary to achieve a shared goal. Tasks are assigned to different learners who will have different responsibilities for performing various tasks. Meanwhile, different members of the community including tutors and experts also divide responsibility in defining and influencing the object of the PBL activity. PBL-specific rules are used to regulate the behaviors in the interaction among learners, tutors and experts, to use instruments, and to measure outcomes.

The PBL activity comprises various forms of actions, such as defining problem, identifying learning issue, searching and collecting learning resources, interviewing experts, discussing and reasoning, generating and evaluating hypothesis and solution, etc. These actions are implemented mainly by learners individually or collaboratively within or outside of classrooms. Each action is goal-oriented and has expected outcome. That is, the motive of the overall activity can be decomposed into a set of goals. These goals are achieved through performing actions, which results in intermediate outcome, such as that the problem is defined, that necessary information is collected, that a preliminary solution is generated, etc. The intermediate outcome becomes parts of conditions for further actions. These conditions will trigger operations of actors on the object by using instruments at hand.

4.1.3 A Conceptual Framework for the Design of Virtual PBL Environments

Based on this conceptualization of human activity, some components of virtual PBL environments can be identified from the analysis of problem based learning. In the paragraphs below, a conceptual framework for the design of virtual PBL environments is proposed [Miao00e]. It can be used to guide the design of virtual PBL environments. This conceptual framework distinguishes eight related components (see Figure 4.3) that do not coincide exactly with the components described in the structure of an activity system.

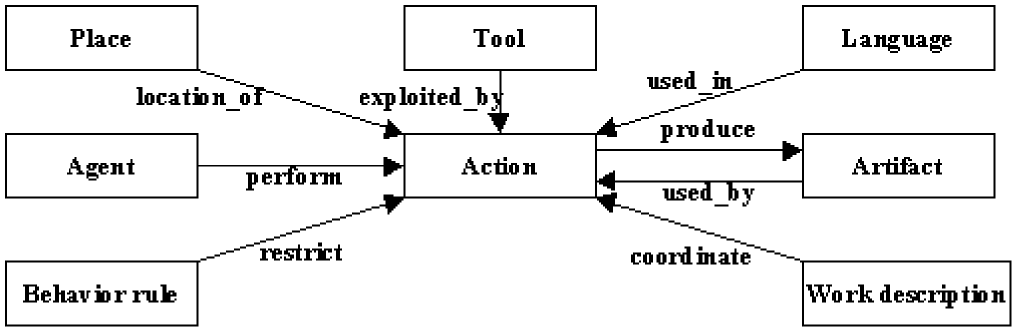


Figure 4.3: The Conceptual Framework

Because more than one PBL activity may be carried out simultaneously in a virtual PBL environment, a user of the system may be a subject of one activity and a member of community of another activity simultaneously. Therefore, the concepts of subject and community have to be combined into the component of *agent* that is used to represent an individual or a group in a computerized form. An individual can, as an actor, be described by some attributes such as name, age, sex, abilities, experience, education, personality, and so on. A community can be defined as a group by specifying a set of actors. A group is a general notion that can refer to all members of a school, a department, a class, a course, a project, a role, etc. A group may consist of other sub-groups and may have a hierarchical structure. Different actor models and group models can be built for different purposes.

The concept of instrument can be decomposed into several components. In PBL, the most frequently used instruments fall into one of four categories: *place*, *tool*, *document*, and *language*. A place is a computational space in which actors can stay and move and actions take place. A system can have a unique place or multiple places. Multiple place systems can have a set of isolated or connected places and a place can be decomposed into a hierarchical structure. A document is a logical unit of information that will be handled (e.g., created, stored, moved, open, and destroyed) as a whole in the system. Information carried in a document can be represented in a text-based form or in a multimedia form. Documents may be kept separate from each other or be connected as a hyper-document. A tool is a kind of system function, with which the actors can perform certain operations to deal with documents, to interact with other actors, and so on. Some tools may have certain relations such as position relation (e.g., book in bookshelf), connected relation (e.g., telephone), and so on. The component of language used here does not refer to natural language or programming language. It is a PBL-specific knowledge representation language that is used to structure knowledge. This language can be defined in a simple form that just provides a category of types of knowledge, or in a complex form that has semantics and structure. It can be used in a text-based form or in a visualized hypertext form.

The concepts of object, outcome, and condition can be partially implemented into the document component, because the PBL activity is somehow a knowledge processing activity. The constructed knowledge is represented in and carried by documents. The object, outcome, and condition of the activity can be measured by investigating the status of the documents. Meanwhile, the concepts of object, outcome, condition, division of labor, action, motive, and goal are combined into the components of *action* and *work-description*. The action component can be divided into pre-hoc part and post-hoc part. The pre-hoc part of the action component is used to represent a scheduled action with information such as the goal, participants, start and terminated condition, place, needed documents, expected outcome, and so on. The post-hoc part of action component refers to a set of operations performed by participants of the action. The work-description component is used to organize the work. A work-description consists of a set of isolated or coordinated actions and is a pre-defined representation of the overall activity. It may take the forms of ToDo lists, plans, checklists, schedules, work programs etc. It can be used to simply provide a common understanding about the ongoing work and to clarify the responsibility of the involved agents. It can also be used as a control mechanism to support automatic execution of learning processes. The concept of rule can be modeled as a *rule* component. The conventions can come in the form of a set of behavior rules that must be followed in social interaction. The rules can be implemented in systems as guidance or help function. It can also be realized in systems as a computational mechanism to force users to behave appropriately.

The relations between these eight components are depicted in Figure 4.3 as well. The center component of the conceptual framework is the action component. An actor or a group of actors perform an action. An action produces document as the outcome of the action and some documents are used as input of the action. An action is carried out in a place. Tools are exploited in an action and some languages are used during the execution of an action. Rules guide and control the social interaction by constraining the operations. The work-description is used to coordinate the contributions made by the participants of actions.

This conceptual framework can be used as a basis to analyze existing virtual PBL environments and as a guideline to design a virtual PBL environment. In order to design a virtual PBL environment, the designers have to make design decision by choosing some of the eight components and by generating solutions to construct each selected component. The next subsection briefly presents a conceptual architecture of a virtual PBL environment developed according to this conceptual framework.

4.1.4 The Conceptual Architecture of a Virtual PBL Environment

According to the conceptual framework described above, the author of the thesis designed a virtual PBL environment. The conceptual architecture of the virtual PBL environment is depicted in Figure 4.4. This conceptual architecture consists of four modules. This section briefly describes these four modules.

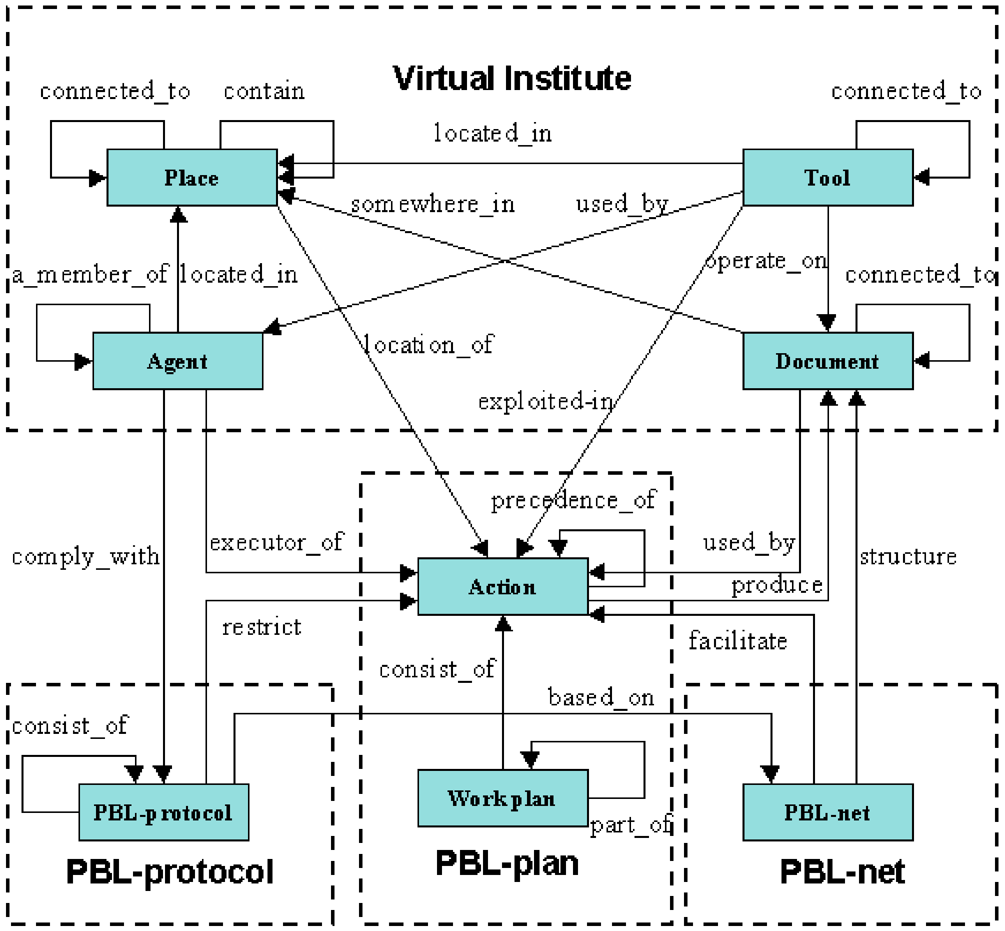


Figure 4.4: The Conceptual Architecture of a Virtual Learning Environment

The core module takes the metaphor of a *virtual institute* that consists of agent, place, tool, and document components. An agent can be an actor or a group. A group can consist of other agents that may be actors and other groups. An actor or a group can be a member of more than one group. That is, the agent has a hierarchical structure and the type of relation between agents is a many-to-many relationship. A place is a computational space. An actor can be present only in one place at a point in time and many actors can be present in the same place at the same time. Places have composite relations and connected relations. The composite relation in the virtual institute is a kind of one-to-many relationship. The root place of a virtual institute is a campus that contains several functional buildings such as administrational building, dormitory, library, and instructional buildings, which, in turn, consist of various types of rooms such as homes and public rooms. Places can be connected in a way that actors can navigate from one place to other places or come back to the place from other places through the doors of this place. Tools (e.g., whiteboard, computer, bookshelf, chat board, telephone, speaker, document search engine, etc) with different functions are available in different types of places. Tools may have a connected relationship. For example, many virtual computers in different places can be connected to a certain virtual computer. Documents are organized in the form of a hyper-document that consists of a set of connected documents. Each document consists of information units in the form of text, table, graphics, images and even hyperlinks to other documents. Meanwhile, many documents can have hyperlinks to refer to a given document. A document can be stored in a bookshelf, or can be opened on whiteboards, in computers, or on private editors. An opened document displayed on a whiteboard or a computer can be edited by means of the edit function provided in editor. The virtual institute inherits part of the culture of real learning environments.

The second module is the *PBL-net* that coincides with the language component. The PBL-net provides support to explicitly represent information types and the relationships between these information types and to guide collaborative problem based learning processes by providing PBL-specific operations to represent, explore, and negotiate shared knowledge. The PBL-net takes the role of PBL-specific cultural mediation in the PBL activity.

The *PBL-protocol* module coincides with the rule component. A PBL-protocol is a computational description of a PBL policy or strategy and a protocol can contain sub-protocols. It represents how learners, tutors, and experts are expected to behave during the learning process. Concretely speaking, a protocol specifies under which condition a learner or a tutor can operate on which information type defined in the PBL-net. More than just a representation, a PBL-protocol can be initiated and the initiated protocol actually forces the learners and tutors to behave appropriately by restricting which behaviors are allowed. In terms of activity theory, a PBL-protocol provides coordination support at the operation level based on the distribution of the subject (roles) in the community.

The action and work-description components are realized in the *PBL-plan* module. A PBL-plan is defined by specifying a set of coordinated actions. An action is defined by specifying the goal, the executors, a location, the input and output document, active condition and terminal condition. The active condition and terminal condition are expressed by one or a combination of conditions at a point in time. Examples are the state of a document, the state of an action, and so on. This module provides a learning plan definition tool, so that the participants of a PBL activity can collaboratively define their own learning plan by specifying the scheduled actions and the relations between these actions. This module also has an enactment mechanism with which a defined learning plan can be initiated, monitored, modified, and executed. When participants carry out an action in the place arranged for the action, they can exploit tools in the place or add new tools, and they can initiate or terminate PBL-protocols. If the terminal condition of an action is met, the system will trigger the next actions according to the definition of the PBL-plan. The documents produced in this action will be transferred to other places as the input documents of the subsequent actions, according to the definition of the executed PBL-plan. In term of activity theory, the PBL-plan takes the role of social mediation in PBL activities at the action level based on the distribution of the objects (goals and tasks) in the community.

4.1.5 Summary

In this section, based on an analysis of problem based learning from the perspective of the activity theory, a conceptual framework for the design of virtual PBL environments was developed. The conceptual framework consists of eight components: agent, place, tool, language, document, action, work description, and behavior rule. This conceptual framework can be used as a basis to analyze existing virtual PBL environments and as a guideline to design a virtual PBL environment. In order to design a virtual PBL environment, the designers should address the mediation role of cultural factors and social factors. Concretely speaking, these eight components should be modeled in virtual PBL environments appropriately. As an example, a virtual PBL environment was described, in which the roles of the cultural and social mediation in the PBL activity are reflected in the system by four modules: the virtual institute metaphor, the PBL-net, the PBL-protocol, and the PBL-plan.

4.2 Notation of Z

The specification language used in this thesis is the Z language. The Z language is a notation for formal specification based on set theory and first order predicate logic. It has been developed at the Programming Research Group at the Oxford University Computing Laboratory and elsewhere since the late 1970s. It is used by industry as part of the software development process in Europe, USA and elsewhere. The objective of using the Z language in this thesis is to present the main design ideas and important behaviors of the system precisely. It is not intended to specify the whole system design completely.

There are a number of reasons to choose the Z language. Firstly, Z has the advantage that it is able to specify a system accurately and unambiguously (unlike semi-formal specification languages). Secondly, the functional specification can be used to express design ideas at an abstract level, rather than describing the design from a mass of detailed program code or pseudo-code. Thirdly, a Z specification can act as a clear statement of design. At times a specification is complex, and understanding the full richness of its behavior may be hard. Its behavior under certain constraints can be deduced as a property of the specification. These 'partial behavior properties' can be used to understand and check a complex behavior. Fourthly, Z can be used to describe a specification of a large system by breaking it down into a number of subsystems, each of which can be specified in a separate document. Fifthly, Z's *schema* can be used to partition the system specification into local and global concerns. By splitting a specification into schemas, the specification can be presented piece by piece accompanied by informal explanation.

This section briefly introduces the basic knowledge about the Z language and the Z notation used in this thesis. Rather than to be a tutorial, the intention of this section is to help readers to recall the Z language and to refer to the Z notation conveniently. The description in this section is primarily taken from two books [Spivey89] [Barden94]. Detailed information about the Z language can be obtained from these books.

4.2.1 Basic Knowledge

Z specifications are mathematical: the variables that appear in them range over mathematical objects, and they express mathematical models of information systems. This subsection contains a description of the world of mathematical objects in which Z specifications have their meaning: it describes what objects there are, and how relationships between them may be made into specifications.

4.2.1.1 Objects and Types

Every mathematical expression that appears in a Z specification is given a *type*: this determines a set known to contain the value of the expression. Each variable is given a type by its declaration, and there are rules for deriving the type of each kind of compound expression from the types of its sub-expression. Every Z specification begins with certain objects that play a part in the specification, but have no internal structure of interest. These atomic objects are the members of the basic types or given sets of the specification. From these atomic objects, composite objects can be put together in various ways. These composite types are the members of composite types put together with the type constructors of Z. There are three kinds of composite types: set types, Cartesian product types, and schema types (see the next subsection). The type constructors can be applied repeatedly to obtain more and more complex types, whose members have a more and more complex internal structure.

4.2.1.2 Schema

Let us discuss schema types in detail. If $p_1, \ldots, p_n$ are distinct identifiers and $x_1, \ldots, x_n$ are objects of types $t_1, \ldots, t_n$ respectively, then there is a binding z with components $z.p_i = x_i$ for each i with $1 \leq i \leq n$. This binding is an object with the schema type

$$<| \, p_1 : t_1; \ldots; p_n : t_n \, |>$$

The binding is equal to another binding w of the same type exactly if $z.p_i = w.p_i$ for each i with $1 \leq i \leq n$. Two schema types are regarded as identical if they differ only in the order in which the components are listed.

A *signature* is a collection of variable names, each with a type. Signatures are created by declarations, and they provide a vocabulary for making mathematical statements, which are expressed by *predicates*. Given a signature, we can think of various situations, in which the variables take different values drawn from their types. A property over the signature is characterized by the situations in which it is true. A predicate expresses a property, and by extension we say a predicate is true in a situation if the property it expresses is true in that situation. A *schema* is a signature together with a property over the signature; the schema *Aleph* with the signature and property might be written

```
┌─ Aleph ──────────────────────
│ x, y : ℤ
├──────────────────────────
│ x < y
└─────────────────────────────
```

or

$$\text{Aleph} \triangleq [x, y : \mathbb{Z} \mid x < y]$$

We call x and y the *components* of *Aleph*.

A fundamental operation on schemas is *systematic decoration*. If **S** is a schema, then **S'** is the same as **S**, except that all the component names have been suffixed with the decoration '. The signature of **S'** contains a component x' for each component x of **S**, and the type of x' in **S'** is the same as the type of x in **S**. The property of **S'** is true in a situation exactly if the property of **S** is true when each component x takes the value taken by x' in that situation.

4.2.1.3 States and Operations

An abstract data type consists of a set of *states*, called the *state space*, a non-empty set of *initial states*, and a number of *operations*. Each operation has certain input and output variables, and is specified by a relationship between the input and output variables and a pair of states, one representing the state before the execution of the operation, and the other representing the state after execution.

In Z, the set of states of an abstract data type is specified by a schema, which is conventionally given the same names as the abstract data type itself. By convention, none of the components of the state space schema has any decoration. The set of initial states of an abstract data type is specified by another schema with the same signatures as the state space schema. The abstract data type may start in any one of the initial states; often there is only one of them. The operations of an abstract data type are specified by schemas which have all the components of both *State* and *State*', where *State* is the schema describing the state space. The state of the abstract data type before the operation is modeled by the undashed components of its schema, and the state afterwards is modeled by the components decorated with a dash. Before and after execution, operations often have inputs and outputs. The inputs are modeled by components of the schema decorated with ?, and the outputs by components decorated with !.

Operations on data types are specified by schemas which have two copies of the state variables among their components: an undecorated set corresponding to the state of the data type before the operation, and a dashed set corresponding to the state after the operation. To make it more convenient to declare these variables, there is a convention that whenever a schema State is introduced as the state space of an abstract data type, the schema **Δ***State* is implicitly defined as the combination of *State* and *State*'. With this definition, each operation on the data type can be specified by extending **Δ***State* with decorations of the inputs and outputs of the operation and predicates giving the pre-condition and post-condition. Many data types have operations that access information in the state without changing the state at all. In this case, it is convenient to have a special schema **Ξ***State* on which these access operations can be built.

4.2.2 The Z Notations Used in This Thesis

4.2.2.1 Syntactic Conventions

The syntactic description of Z constructs given in this subsection is intended as a guide to the way the constructs looks on paper: it treats each construct in isolation, and does not properly respect the relative binding powers of connectives and quantifiers, for example. A full grammar for Z is given in [Spivey89]. A few extensions to BNF are used to make the syntax descriptions more readable. The notation **S; …; S** stands for one or more instances of syntactic class **S**, separated by semicolons; similarly, the notation **S, …, S** stands for one or more **S**'s separated by commas. Slanted square brackets *[ ]* enclose items which are optional. Lists of items that may be empty are indicated by combining these two notations.

A word (**Word**) is the simplest kind of name in a Z specification: it is either a non-empty sequence of upper and lower case letters, digits, and underscores beginning with a letter, or a special symbol. Words are used as the names of schemas. An identifier (**Ident**) is a word followed by a decoration (**Decoration**), which is a possibly empty sequence of ', ?, or ! characters:

Ident ::= Word Decoration

If a word is used in a specification as the name of a schema, it is called a schema name and is no longer available for use as in an ordinary identifier. Schemas are named with words rather than identifiers to allow systematic decoration: if A is a schema and we write A', this means a copy of A in which all the component names have been decorated with '. Some words are given the special status of prefix, infix, or postfix symbols (e.g., ¬, +, and *).

4.2.2.2 Specifications

A Z specification document consists of interleaved passages of formal, mathematical text, and informal prose explanation. The formal text consists of a sequence of paragraphs that gradually introduce the schemas, global variables and basic types of the specification, each paragraph building on the ones that come before it. Except in the case of free type definitions, recursion is not allowed. Several kinds of paragraph are introduced below.

1). Basic type definitions

Paragraph ::= [**Ident, …, Ident**]

A basic type definition introduces one or more basic types.

2). Axiomatic descriptions

```
                     │ Declaration
Paragraph ::= [      ├─────────────────────────────
                     │ Predicate; …; Predicate              ]
```

An axiomatic description introduces one or more global variables, and optionally specifies a constraint on their values.

3). Constraints

Paragraph ::= Predicate

A predicate may appear on its own as a paragraph; it specifies a constraint on the values of previously declared global variables.

4). Abbreviation definitions

Paragraph ::= Ident = = Expression

An abbreviation definition introduces a new global constant.

5). Free type

Paragraph ::= Ident = = Branch | … | Branch

Branch ::= Ident *[*≪**Expression**≫*]*

A free type definition is a way of introducing a given set, together with some additional information. The branch names are injective functions, so we follow the convention for variables when naming them. Care should taken with the intuitive meaning associated with the name given to the accompanying injections.

6). Schema definitions

```
                       ┌── Schema-Name ──────────────────────────
                       │ Declaration
Paragraph ::=   [      ├─────────────────────────────
                       │ Predicate ;…; Predicate                 ]
                       └─────────────────────────────────────
```

or

Paragraph ::= Schema-Name ≙ Schema-Exp

These forms introduce a new schema name. The word heading the box or appearing on the left of the definition sign becomes associated with the schema that is the contents of the box or appears to the right of the definition sign.

Schema-Exp ::= $\forall$ **Schema-Text • Schema-Exp**
| $\exists$ **Schema-Text • Schema-Exp**
| $\exists_1$ **Schema-Text • Schema-Exp**
| **Schema-Exp-1**

Schema-Exp-1 ::= [Schema-Text]
| **Schema-Ref**
| $\neg$ **Schema-Exp-1**
| **pre Schema-Exp-1**
| **Schema-Exp-1** $\wedge$ **Schema-Exp-1**
| **Schema-Exp-1** $\vee$ **Schema-Exp-1**
| **Schema-Exp-1** $\Rightarrow$ **Schema-Exp-1**
| **Schema-Exp-1** $\Leftrightarrow$ **Schema-Exp-1**
| **Schema-Exp-1** $\upharpoonright$ **Schema-Exp-1**
| **Schema-Exp-1 \ (Decl-Name, …, Decl-Name)**
| **Schema-Exp-1** $\fatsemi$ **Schema-Exp-1**
| **(Schema-Exp)**

4.2.2.3 Schema References

When a schema name has been defined as described above, it can be used in a schema reference to refer to the schema. A schema reference can be used as a declaration, an expression, or a predicate, and it forms a basic element of schema expressions.

Schema-Ref ::= Schema-Name Decoration

A schema reference consists of a schema name followed by a decoration (which may be empty).

4.2.2.4 Declarations

Variables are introduced and associated with types by *declarations*. As explained above, a declaration may also require that the values of the variables satisfy a certain property, which we call the constraint of the declaration. There are two kinds of declaration in Z:

Basic-Decl ::= Ident, …, Ident : Expression
| **Schema-Ref**

The first kind introduces an explicitly-listed collection of variables. When a schema reference is used as a declaration, it introduces the components of the schema as variables, with the same types as they have in the schema, and constrains their values to satisfy its property.

In every context where a single declaration is allowed, a sequence of declarations may also appear:

Declaration ::= Basic-Decl; …; Basic-Decl

This declaration introduces all the variables introduced by each of its constituent basic declarations, with the same types.

A set-comprehension expression has the form

{ **Declaration | Predicate ;…; Predicate • Expression** }

And its value is the set of values taken by the expression when the variables introduced by the declaration take all values that satisfy both the constraints of the declaration and predicates. The expression part may be omitted, and the default is then the characteristic tuple of the declaration.

4.2.2.5 Schema Text

A schema text consists of a declaration and an optional list of predicates. Most Z constructs that introduce variables allow a schema text rather than simply a declaration, so that a relationship between the values of the variables can be described. Schema texts appear in vertical form in axiomatic descriptions and schema definitions, but they also have a horizontal form:

Schema-Text ::= Declaration *[* **| Predicate ;…; Predicate** *]*

4.2.2.6 Mathematical Symbols

The syntex of predicate and expression and some components of the Z language are omitted in this thesis, because of the limitation of the space. A complete syntext of the Z language can be obtained in the Spivey's book [Spivey89]. Note that it is a convention to keep the use of parentheses to the minimum required if their absence would not be confusing. For example, given a function is defined as

wordCount : Document → ℕ
aThesis : Document

the number of words in the thesis is represented as

wordCount (aThesis)

However, it can be represented as

wordCount aThesis

Some symbols used to represent expression and predicate are listed below:

(…)	Tuple
{…}	Set display
< …>	Sequences
$\langle\!\mid \ldots \mid\!\rangle$	Binding
$\mathbb{P}$	Power set
$\times$	Cartesian product
$\{ \mid \bullet \}$	Set comprehension. $\{S \mid P \bullet V\}$ means for all members of S, which fulfill P then V must be valid
λ	Lambda-expression: The expression $\lambda \boldsymbol{S} \bullet \boldsymbol{E}$ denotes a function that takes arguments of a shape determined by ***S***, and returns the result ***E***
μ	Mu-expression: The expression $\mu \boldsymbol{S} \bullet \boldsymbol{E}$ is defined only if there is a unique way of giving values to the variables introduced by ***S*** that makes the property of ***S*** true; if this is so, then its value is the value of ***E*** when the variables introduced by ***S*** take these values
$\boldsymbol{\theta}$	Binding formation: In the expression $\boldsymbol{\theta S}$’, in which the symbol ’ stands for an optional decoration, let the components of ***S*** be $x_1, \ldots, x_n$. The variables $x'_1, \ldots, x'_n$ must be in scope: let their types be $t_1, \ldots, t_n$. The type of the expression is the schema type $\langle\!\mid x_1 : t_1; \ldots; x_n : t_n \mid\!\rangle$. The value of the expression $\boldsymbol{\theta S}$’ is a binding z with the schema type shown above; for each i with $1 \leq i \leq n$, the component $z.x_i$ is the value of the variable x_i in that situation.
.	Selection: the notation for selecting a component from a binding
\	Schema hiding
$\leftrightarrow$	Binary relations
$\mapsto$	Maplet
$\fatsemi$	Relational composition and sequential composition
$\lhd$	Domain restriction
$\rhd$	Range restriction
$⩥$	Domain anti-restriction
$⩤$	Range anti-restriction
$\_^{\sim}$	Relational inversion
$\_^{+}$	Transitive closure
$\_^{*}$	Reflexive-transitive closure
$\_(\!\mid\_\mid\!)$	Relational image
$\nrightarrow$	Partial functions
$\rightarrow$	Total functions
$⤔$	Partial injections
$\rightarrowtail$	Total injections
$⤀$	Partial surjections
$\twoheadrightarrow$	Total surjections
$⤖$	Bijections
$\oplus$	Functional overriding

#	Number of members of a set
$\upharpoonright$	Filter
$\frown$	Concatenation
•	Order properties of set operation, Monotonic operations, or Relational operations on functions and sequences
dom	Domain of a relation
ran	Range of a relation
first	Projection function splitting ordered pairs into the first co-ordinates
second	Projection function splitting ordered pairs into the second co-ordinates
id	Identity relation
min	Minimum of a set of numbers
max	Maximum of a set of numbers
pre	Pre-condition
seq	Finite sequences
seq$_1$	Non-empty finite sequences
iseq	Injective sequences
head	The first element of a non-empty sequence
last	The last element of a non-empty sequence
disjoint	Disjointness
partition	Partition

4.3 Virtual Institute Metaphor: A Context-based Virtual Learning Environment

Collaborative learning is a process of social interactions and social construction of knowledge. Theories from education promote an understanding of how learner's knowledge structures, motivations, and interpersonal interactions interact with learning environments. The goal of building a virtual collaborative learning environment is to provide an environment so that geographically distributed people can interact with each other and construct knowledge collaboratively as they would do in conventional co-located learning environments. Especially, for supporting problem-based learning, it is critical for a successful system to provide rich context for social interactions and social construction of knowledge.

Designing a virtual collaborative learning support system requires a process of abstraction, which focuses only on the essential elements of conventional learning environments. In virtual learning environments, more or less details of collaborative learning may be ignored intentionally or because of the limitations of technology. Ignoring some aspects of collaborative learning is dangerous, because rich forms of social interactions may become impossible. Limited forms of social interactions, in turn, may make collaborative PBL processes very difficult. According to our state of the art analysis, all existing PBL support systems omit some crucial aspects.

The first section of this Chapter has presented a conceptual framework for the design of virtual PBL environments and the conceptual architecture of a virtual PBL environment. It suggested that a virtual learning environment should reflect the major culture existing in real learning environments. However, how to inherit the culture of real learning environments is unclear. The approach taken in this thesis is to use the metaphor of a virtual institute. This section is organized as follows. It begins by an introduction of the theory of situated learning, which emphasize the importance of the context in which the student works, and the importance of social interactions. Following the guidelines of the theory of situated learning, the requirements for design of virtual learning environments are identified. The main body of this section describes the basic concepts of a context-based virtual learning environment and an approach to develop a context-based virtual learning environment. The context-based virtual learning environment is formally described to demonstrate how to support construction and maintenance of learning contexts, how to support awareness of learning contexts, and how to support social interaction. Finally, we summarize this section by comparing our approach with other approaches.

4.3.1 Theoretical Background

The theory of *situated learning* formulated by Lave and Wenger considers that social interaction is a critical component of situated learning. It is impossible to separate cognitive tasks from social tasks, because all cognitive tasks have a social component [Perret93]. Collins et al. developed the concept of cognitive apprenticeship [Collins89]. Brown et al. believed: "cognitive apprenticeship supports learning in a domain by enabling students to acquire, develop and use cognitive tools in authentic domain activity. … In essence, cognitive apprenticeship attempts to promote learning within the nexus of activity, tools, and culture that they have described. Learning, both outside and inside school, advances through collaborative social interaction and the social construction of knowledge. … So the term apprenticeship helps to emphasize the centrality of activity in learning and knowledge and highlights the inherently context-dependent, situated, and enculturating nature of learning" [Brown89]. Lave et al. believed that collaboration can lead to articulation of learning strategies that can then be discussed, which, in turn, can enhance generalization grounded in students' situated understandings [Lave91a]. Greeno et al. suggested: “we need to organize learning environments and activities that include opportunities for acquiring basic skills, knowledge, and conceptual understanding, not as isolated dimensions of intellectual activity, but as contributions to students’ development of strong identities as individual learners and as more effective participants in the meaningful social practices of their learning communities in school and elsewhere in their lives” [Greeno98]. “Our community, and each of us, creates networks of connections (and disconnections) among texts, situations and activities… These networks of connections that we make, and that are made in the self-organizing activity of the larger systems to which we belong, extend backwards in time as well as outwards into the social-material world” [Lemke97]. Wenger [Wenger98] uses *Communities of Practice* (CoP) to describe the impact of social learning. A community of practice is defined by McDermott as “… a group that shares knowledge, learns together, and creates common practices” [McDermott99]. He wrote: “Community members frequently help each other solve problems, give each other advice, and develop new approaches or tools for their field. Regularly helping each other makes it easier for community members to show their weak spots and learn together in the ‘public space’ of the community” [McDermott99].

Trilling and Hood emphasized the role of context in learning: "the environmental conditions for learning (objects, people, symbols, and their relationships) are much more influential than we've previously thought, and that the transfer of knowledge from one context to another is not often successful. The demand for more 'authentic' learning tasks that match real-world conditions comes directly from these findings, as well as the desire to have rich learning environments that offer a wide variety of contextualized opportunities for discovery, inquiry, design, practice, instruction, and constructive exploration. This approach coincides with the need to become proficient in solving real-world problems and to exercise critical thinking-and-doing in the Knowledge Age" [Trilling99]. Suchman [Suchman87] argued: "action such as learning, understanding and remembering is situated. Because of the situated nature of action, communication must include both an awareness of the local context and a mechanism to solve problems in understanding." In addition, "situations might be said to co-produce knowledge through activity" [Brown89]. Learning environments will be changed by learning groups during learning processes.

According to Wolfson et al. [Wolfson], learning results from four components:

1) "situated contexts: communities of practice, artifacts as mediating devices, multiple resources,
2) authentic contexts: authentic projects, problem solving scenarios, intrinsic motivation and student responsibility,
3) collaborative contexts: small group interactions, skilled peer guidance, community expert guidance, and
4) reflective contexts: goal setting, formative assessment, teacher modeling & scaffolding, cognitive apprenticeship."

4.3.2 Requirements

Based on the guideline of the theories described above, we can develop the concept of learning context. A *learning context* can be regarded as a situation for learning [Miao99d]. A learning context consists of a set of structured places in which tools and learning materials are available and constructed knowledge is recorded for communities of practice. A formal definition of learning context will be given in this section. The nature of learning contexts is dynamical, and they evolve in time. It is important to note that the members of communities of practice are active agents involved in collaborative learning activities and they become a part of learning contexts. In this subsection, we analyze the main requirements for building a comprehensive collaborative learning environment to support construction of learning contexts, awareness of learning context, and social interaction in learning contexts. These requirements are consistent with those derived from the scenario in Chapter 2. This subsection presents the requirements in a systematical way.

4.3.2.1 Support for the Construction of Learning Contexts

A learning context is formed while a group of people shares a commitment to some form of collective learning activities, e.g., conducting a problem based learning course. They need to socially present themselves to other people. Increasing possibilities of co-presence of people in the same place at the same time will increase the likelihood of social interactions. In the discourse of social interaction, they use tools and learning materials at hand. As a learning process progresses, they will change the learning environment in the way that new places are arranged, new tools are installed and exploited, documents are produced or introduced, new members are introduced, and so on. A virtual learning environment should allow users to configure and reconfigure learning contexts on demand. It should provide suitable tools and relevant documents for establishing different learning contexts and to maintain learning contexts so that social interactions can be carried out at low cost. That is, users do not need to make a lot of efforts to seeking suitable partners and to search for necessary documents and tools in the discourse of interactions.

4.3.2.2 Support for Awareness of Learning Contexts

Awareness of learning context is a precondition for situated learning. When a user has a clear idea about the current situation, s/he can rapidly involve her/himself in the situation, take proper roles, and initiate or join learning activities. A virtual learning environment should support user's awareness of existence and state of tangible entities (e.g., people, documents, and tools, places, and their relations) and even intangible entities (e.g., activity) in different degrees through different ways.

4.3.2.3 Support for Social Interaction in Learning Contexts

In learning processes such as the scenario described in the chapter 2, a variety of forms of social interaction take place. A virtual learning environment should support rich forms of social interaction. No matter whether users are geographically distributed or co-located, they can enter the same computational place (virtual place) or can be located in different virtual places. Therefore, a virtual learning environment should be able to support synchronous and asynchronous communication conducted in the same virtual place and in different virtual places by using tools. The system should support social construction of knowledge at same time and at different time, at the same virtual place and at different virtual places, jointly or in sub-groups. A virtual learning environment should support users to make use of shared resources effectively.

4.3.3 Design of a Context-based Virtual Learning Environment

This subsection describes the design of a virtual learning environment. Adopting this approach, a virtual learning environment is developed by using a set of metaphors. These metaphors can be flexibly combined to form different learning contexts for support different collaborative learning activities. As an example system, a set of metaphors is used and organized in a way to form a metaphor of institute, called as a

Virtual Institute. As illustrated in Figure 4.5, a *Virtual Institute* consists of an institute space, a community, a hyperdocument base, and a tool base. The institute consists of virtual places that are connected by doors. Each door can be approached through two door views that belong to the places connected by the door. The community consists of actors with a group structure. Both actors and groups are agents. The hyperdocument base contains a set of documents. A document may have references that serve as hyperlinks to other documents. The tool base contains a set of tools that may be document editor, bookshelf, message-box, and so on. Actors, documents, and tools will be located in a virtual place. A document may be edited by actors using a tool. A *Virtual Institute* provides the overall learning context in which all learning-related activities occur. Each virtual place with the objects it contains forms a learning context.

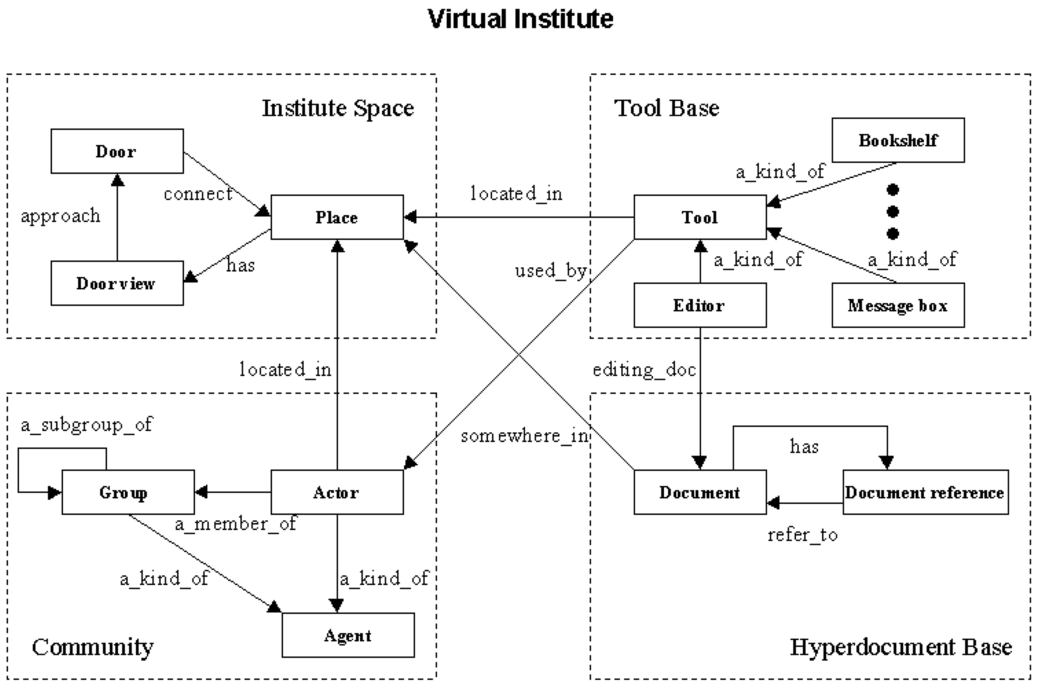


Figure 4.5: The Conceptual Architecture of a Virtual Institute

In this subsection, the design of a context-based virtual learning environment is described formally.

4.3.3.1 Basic Concepts of the Context-Based Virtual Learning Environment

First of all, we define the basic types:

[$\mathbb{N}$ $\mathbb{N}_1$ $\mathbb{Z}$ $\mathbb{R}$ BOOLEAN CHAR STRING TEXT IMAGE TABLE]

These notations represent the types of natural number, integer, real number, Boolean, character, string, text, image, and table respectively. In order to ease discussion, we don’t consider the internal structure of some data types such as text in our model. These data types are introduced as given sets. In this thesis, it is sufficient to model aspects of time using integer such that time can be calculated.

TIME = = $\mathbb{N}$

Then we formally describe the four parts of a virtual institute: community, institute space, hyperdocument base, and tool base respectively.

4.3.3.1.1 Community

Definition (Actor): An *actor* represents a user of the system in a computational form. An *actor* has attributes to identify and characterize a user such as name, picture, phone number, email address, expertise, and learning interests.

— Actor —

name : STRING
picture : IMAGE

phoneNumber : $\mathbb{N}_1$
emailAddr : STRING

expertises : $\mathbb{P}$ STRING

learningInterests : $\mathbb{P}$ STRING

Definition (Group): A *group* refers to a general notation for a department, a class, a project, a role, etc. We simply model it by using a name attribute.

— Group —

name : STRING

Definition (Agent): *An agent* represents a general notation for an *actor* or a *group*.

Agent ::= actor <<Actor >>
| group <<Group>>

Definition (Community): A *community* consists of a set of agents with structural relations among them. The set of actors and the set of groups are distinguished, and these two sets partition the set of agent. Within a community, an actor may be a member of a group and a group may be a sub-group of another group. An actor can be a member of multiple groups at the same time and a group can have multiple members. A group can be a sub-group of multiple groups at the same time and a group can have multiple sub-groups as well. However, the sub-group relations within a community can not form a loop. In other words, a group can not be a sub-group of another group that is directly or indirectly a sub-group of the first group. It is important to note that the term of community used here denotes all potential users of the system and the possible formal relations (e.g., aMemberOf and aSubGroupOf). It does not represent a *community of practice* defined in situated learning. A community of practice is a natural, informal, and domain specific group. The members of a community of practice share their experiences and learn from each other on a regular basis with respect to a background of shared practices [Wenger98]. A community of practice is neither a community nor a group. However, a community of practice is formed and developed within a community. The seed of a community of practice may be a formally defined group. The members of a community of practice may usually meet in a place to do something together. There may be multiple communities of practice within a community.

Community

actors : $\mathbb{P}$ Actor
groups : $\mathbb{P}$ Group
agents : $\mathbb{P}$ Agent
aMemberOf : Actor $\leftrightarrow$ Group
aSubGroupOf : Group $\leftrightarrow$ Group

ran actor = actors $\wedge$ **ran** group = groups
< **ran** actors, **ran** groups > **partition** agents
dom aMemberOf $\subseteq$ actors
ran aMemberOf $\subseteq$ groups
(**dom** aSubGroupOf $\cup$ **ran** aSubGroupOf) $\subseteq$ groups
disjoint < aSubGroupOf $^{+}$, **id** Group >

4.3.3.1.2 Institute space

Definition (Place): A *place* represents a virtual space in which objects such as actors can exist and move and actions can take place. There are several types of place: campus, administrational building, library, dormitory, instructional building, home, public room, specific room. Each place has a name, a type, and an owner that can be an actor or a group. There are relationships between places. The data type GoTo is defined to represent a navigational relation from the source place to the destination place directly.

PlaceType ::= campus
| administrationalBuilding | library
| dormitory | instructionalBuilding
| home | publicRoom | specificRoom

Place

name : STRING
placeType : PlaceType
owner : Agent

GoTo = = Place × Place

Definition (Door): A *door* represents a gateway between two places. A door may be open or closed. In this model, there are two types of doors. The first type of door is a *concrete* door. This kind of door is used to connect all places in a virtual institute into a tree. The second type of door is a *virtual* door that doesn't exist in the real world. It provides a (one-way) shortcut gateway from one place to another.

DoorType ::= concrete | virtual

```
┌─ Door ──────────────────────────────────────
│ status : BOOLEAN
│ doorType : DoorType
└─────────────────────────────────────────────
```

Definition (Door View): *A door view* denotes a view of a door in a user interface (UI) for interaction purposes. A door view has a name and an image.

```
┌─ DoorView ──────────────────────────────────
│ name : STRING
│ view : IMAGE
└─────────────────────────────────────────────
```

Definition (Institute Space): An *institute space* consists of a set of places with certain relations. In order to describe operations easily, the *campus* place and the *dormitory* place are distinguished from other places. A place may contain other places, e.g., a campus contains several buildings and a building contains several rooms. Furthermore, a room can contain several smaller rooms. Within an institute space, there is a set of such relationships between places. Each place has a set of door views. Each door view refers a door. Each door refers to a GoTo relation between two places.

The campus and the dormitory are two special places and may not be identical. The campus contains certain places of type administrational building, library, dormitory, and instructional building. An administrational building contains places with the type of specific room. An instructional building contains places with the type of public room. A dormitory contains places with the type of home.

A place can contain other places and can be contained by another place. However, not all places within an institute space can contain or can be contained in other places. The campus can not be contained in any place in the institute space. It is not allowed that a place contains itself directly or indirectly. Within an institute space, there is no isolated place, i.e., every place is reachable via contain relationships from the campus place.

Each concrete door has two door views that are visible from the two places connected by the door respectively. The name of a door view in one side is exactly equal to the name of the place on the other side. A virtual door has only one door view that can be seen in the source place and the name of this view is equal to the name of the destination place.

InstituteSpace

$\text{places} : \mathbb{P}\ \text{Place}$
$\text{campus, dormitory} : \text{Place}$

$\text{contains} : \mathbb{P}\ \text{GoTo}$

$\text{hasDoorView} : \text{Place} \rightarrow \mathbb{P}\ \text{DoorView}$

$\text{approach} : \text{DoorView} \rightarrow \text{Door}$

$\text{connect} : \text{Door} \rightarrow \text{GoTo}$

$\text{campus} \in \text{places} \wedge \text{dormitory} \in \text{places} \wedge \text{campus} \neq \text{dormitory}$

$\mathbf{ran}\ (\{\text{campus}\} \lhd \text{contains}) = \{ p : \text{Place} \mid$
$\quad p.\text{placeType} = \text{administrationalBuilding} \vee$
$\quad p.\text{placeType} = \text{library} \vee$
$\quad p.\text{placeType} = \text{domitory} \vee$
$\quad p.\text{placeType} = \text{instructionalBuilding} \}$

$\mathbf{ran}\ (\{ p : \text{Place} \mid p.\text{placeType} = \text{administrationBuilding} \vee$
$\quad p.\text{placeType} = \text{library} \} \lhd \text{contains}) = \{ p : \text{Place} \mid p.\text{placeType} = \text{specificRoom}\}$

$\mathbf{ran}\ (\{ p : \text{Place} \mid$
$\quad p.\text{placeType} = \text{instructionalBuilding} \vee p.\text{placeType} = \text{publicRoom} \} \lhd \text{contains})$
$\quad = \{ p : \text{Place} \mid p.\text{placeType} = \text{publicRoom}\}$

$\mathbf{ran}\ (\{ p : \text{Place} \mid p.\text{placeType} = \text{dormitory} \} \lhd \text{contains}) =$
$\quad \{ p : \text{Place} \mid p.\text{placeType} = \text{home} \}$

$\mathbf{dom}\ \text{contains} \subset \text{places}$
$\mathbf{ran}\ \text{contains} \subset \text{places}$
$\text{campus} \notin \mathbf{ran}\ \text{contains}$
$\text{contains} \in \text{Place} \twoheadrightarrow \text{Place}$

$\mathbf{disjoint} < \text{contains}^{+}, \mathbf{id}\ \text{Place} >$
$(\text{contains} \cup \text{contains}^{\sim})^{*} = \text{places} \times \text{places}$
$\mathbf{ran}\ \text{approach} = \mathbf{dom}\ \text{connect}$

$\forall\ \text{ref} : \text{DoorView}; d: \text{Door} \mid (\text{ref} \mapsto d) \in \text{approach} \wedge d.\text{doorType} = \text{concrete} \bullet$
$\quad \#\ \mathbf{dom}\ (\text{approach} \rhd \{d\}) = 2 \wedge$
$\quad (\text{ref} \in \mathbf{ran}\ ((\mathbf{first}\ (\text{connect}\ d)) \lhd \text{hasDoorView}) \vee$
$\quad\quad \text{ref} \in \mathbf{ran}\ ((\mathbf{second}\ (\text{connect}\ d)) \lhd \text{hasDoorView})) \wedge$
$\quad (\text{ref} \in \mathbf{ran}\ ((\mathbf{first}\ (\text{connect}\ d)) \lhd \text{hasDoorView})$
$\quad\quad \text{ref.name} = (\mathbf{second}\ (\text{connect}\ d)).\text{name}) \wedge$
$\quad (\text{ref} \in \mathbf{ran}\ ((\mathbf{second}\ (\text{connect}\ d)) \lhd \text{hasDoorView})$
$\quad\quad \text{ref.name} = (\mathbf{first}\ (\text{connect}\ d)).\text{name})$

$\forall$ ref : DoorView; d: Door | (ref $\mapsto$ d) $\in$ approach $\wedge$ d.doorType = virtual $\bullet$
  # **dom** (approach $\rhd$ {d}) = 1 $\wedge$
  (ref $\in$ **ran** ((**first** (connect d)) $\lhd$ hasDoorView) $\wedge$
    ref.name = (**second** (connect d)).name)

It is important to note that this institute space model is a specific hypertext model. In terms of hypertext systems, a place in this model is a node and a door in this model is a hyperlink. According to the categories of Conklin [Conklin87b], a concrete door in this model represents a bi-directional *organizational link* and a *virtual door* can be regarded as an uni-directional *referential link.*

When a door is closed on one side, one can not move to the other side of the door. If an actor don't want to be disturbed by others when s/he is doing something, s/he can simply close the concrete door inside of the place. The door concept provides a flexible navigation control mechanism for users. By using virtual doors, a place can be connected to any other place that is not contained by this place. For example, a public room created for studying frogs can be connected to an instructional building for learning biology and can also be connected to an instructional building for a problem-based learning course about deformed frogs. A virtual door can be created between a home in a dormitory and a classroom in an instructional building. It provides a shortcut for actors to navigate from one place to another.

4.3.3.1.3 Hyperdocument Base

Definition (Document): A *document* represents a logical unit of information. A document will be handled (e.g., created, stored, moved, and destroyed) as a whole. Each document has a title and information about the topic of this document, the current status, and the creator of the document. A document has a content that may contain a collection of information items. An information item can be described by using different media objects such as text, image, and table. In this thesis, the nature of a media object is not further considered.

MediaObject ::= TEXT | TABLE | IMAGE

— Document —
title : STRING
owner : Actor
topic : STRING
status : STRING

content : $\mathbb{P}$ MediaObject

Definition (Document Reference): A *document reference* serves as a link to another document. Each document reference has a name.

```
┌─ DocumentReference ─────────────────────────
│ name : STRING
└─────────────────────────────────────────────
```

Definition (Hyperdocument Base): A *hyperdocument base* contains a set of documents usually called nodes. Within a hyperdocument base, a document may have a set of document references. Each document reference within a document refers to another document in the hyperdocument base. And the name of a document reference is equal to the title of the document to be referred.

```
┌─ HyperDocumentBase ─────────────────────────
│ documents : ℙ Document
│ documentRefs : Document → ℙ DocumentReference
│ referTo : DocumentReference → Document
├──────────────────────────────
│ dom documentRefs ⊆ documents
│ ran referTo ⊆  documents
│
│ ∀ r : referTo | (first r).name = (second r).title
└─────────────────────────────────────────────
```

4.3.3.1.4 Tool Base

Definition (Document Editor): A *document editor* is used to browse and edit hyperdocuments. There are three types of document editors: whiteboard, computer, and private editor. Each document editor has a type and a history of navigation in the hyperdocuments.

EditorType ::= whiteboard | computer | privateEditor

```
┌─ DocumentEditor ────────────────────────────
│ history  : seq Document
│ type  : EditorType
└─────────────────────────────────────────────
```

Definition (Bookshelf): A *bookshelf* is used to store documents.

[Bookshelf]

Definition (Message Box): A *message box* is used to transfer documents between places. Actors can use it to send documents to other places by giving the name of destination places. The actors in the destination places can take the received documents from the message box installed in these places. The system uses it to transfer documents automatically from one place to another according to learning plans.

[MessageBox]

Definition (Calendar): A *calendar* is used to manage scheduled actions. Each actor has a private calendar that in installed in home of the actor. A calendar installed in a public room is used to manage past, currently executed, and future actions that are related to this public room.

[Calendar]

Definition (Specific Tool): A *specific tool* is a general notion for some special kinds of tools: document search engine, group definition tool, knowledge structure definition tool, collaboration protocol definition tool, and session-based collaborative process definition tool. In order to concentrate on the major features of the model, the document search engine and the group definition tool will not be described formally. They will be mentioned briefly when describing some operations. The knowledge structure definition tool, collaboration protocol definition tool, and session-based collaborative process definition tool will be described in detail in the subsequent chapters. In order to support specific learning tasks, more task-specific tools can be developed and installed in different places to form different learning contexts. The system is open for integrating new tools.

[DocumentSearchEngine GroupDefinitionTool]

[KnowledgeStructureDefinitionTool CollaborationProtocolDefinitionTool SCPDefinitionTool]

```
SpecificTool ::= documentSearchEngine << DocumentSearchEngine >>
   | groupDefinitionTool << GroupDefinitionTool >>
   | knowlegeStructureDefinitionTool << KnowlegeStructureDefinitionTool >>
   | collaborationProtocolDefinitionTool <<CollaborationProtocolDefinitionTool >>
   | scpDefinitionTool << SCPDefinitionTool >>
```

Definition (Tool): A *tool* represents a general notion of ways that can be used to deal with documents, to interact with other actors, and so on. A tool may be a document editor, a bookshelf, a message box, a chatboard, a phone, a speaker, a conversation tool, a suitcase, a calendar, and a specific tool. Note that some types of tools such as chatboard, phone, speaker, conversation tool, and suitcase will not be formally described in this thesis. Some specific types of tools will be discussed in the subsequent sections in detail.

```
Tool ::= documentEditor << DocumentEditor >>
        | bookshelf << Bookshelf >>
        | messageBox << MessageBox >>
        | chatboard << Chatboard >>
        | calendar << Calendar >>
        | phone << Phone >>
        | speaker << Speaker >>
        | conversationTool << ConversationTool >>
        | suitcase << Suitcase >>
        | specificTool << SpecificTool >>
```

Definition (Tool Base): A *tool base* contains all tools and their relationships. Editor with computer type can connect to and be connected by other computers.

ToolBase

$\mathit{editors} : \mathbb{P}\ \mathit{DocumentEditor}$

$\mathit{bookshelves} : \mathbb{P}\ \mathit{Bookshelf}$

$\mathit{messageBoxes} : \mathbb{P}\ \mathit{MessageBox}$

$\mathit{calendars} : \mathbb{P}\ \mathit{Calendar}$

$\mathit{specificTools} : \mathbb{P}\ \mathit{SpecificTool}$
$\mathit{connectedTo} : \mathit{DocumentEditor} \twoheadrightarrow \mathit{DocumentEditor}$

$\forall\ x, y : \mathit{DocumentEditor} \mid (x, y) \in \mathit{connectedTo} \bullet$
$(\ x.\mathit{type} = \mathit{computer} \wedge y.\mathit{type} = \mathit{computer}\ )$

4.3.3.1.5 Virtual Institute and Learning Context

Definition (Virtual Institute): A *virtual institute* is defined as an abstract state that consists of four parts and their relations. An actor is located only in one place or is not present in the institute. Each editor is located in a place and has a document to be currently viewed and edited. An editor can be used by multiple users and a user can work on multiple editors. The number of users of a private editor is limited to one at a point in time. Each bookshelf is located in a place and two bookshelves can not be located in the same place. Some documents are stored in bookshelves. Each message box is located in a place and two message boxes can not be located in the same place. Some documents are stored in message boxes. A document must be in a place. Each calendar is located in a place and two calendars are not allowed to be located in the same place. Each specific tool is located in a place. If a document is somewhere in a place, it may be in the bookshelf installed in this place, or may be currently edited in an editor installed in this place, or may be in the message box installed in this place. However, it must be in one and only one type of tool. A private editor can be used at maximum one user at a point in time.

VirtualInstitute

InstituteSpace
Community
HyperdocumentBase
ToolBase
$\mathit{actorLocation} : \mathit{Actor} \twoheadrightarrow \mathit{Place}$

$\mathit{editorLocation} : \mathit{DocumentEditor} \rightarrow \mathit{Place}$

$\mathit{currentDoc} : \mathit{DocumentEditor} \rightarrow \mathit{Document}$

$\mathit{usedBy} : \mathit{DocumentEditor} \leftrightarrow \mathit{Actor}$
$\mathit{bookshelfLocation} : \mathit{Bookshelf} \rightarrowtail \mathit{Place}$
$\mathit{storedIn} : \mathit{Document} \twoheadrightarrow \mathit{Bookshelf}$
$\mathit{messageBoxLocation} : \mathit{MessageBox} \rightarrowtail \mathit{Place}$
$\mathit{inMessageBox} : \mathit{Document} \twoheadrightarrow \mathit{MessageBox}$

$\mathit{somewhereIn} : \mathit{Document} \rightarrow \mathit{Place}$
$\mathit{calendarLocation} : \mathit{Calendar} \rightarrowtail \mathit{Place}$
$\mathit{specificToolLocation} : \mathit{SpecificTool} \rightarrow \mathit{Place}$

$\mathbf{dom}\ \mathit{actorLocation} \subseteq \mathit{actors} \wedge \mathbf{ran}\ \mathit{actorLocation} \subseteq \mathit{places}$
$\mathbf{dom}\ \mathit{somewhereIn} = \mathit{documents} \wedge \mathbf{ran}\ \mathit{somewhereIn} \subseteq \mathit{places}$
$\mathbf{dom}\ \mathit{usedBy} \subseteq \mathbf{dom}\ \mathit{currentDoc} = \mathbf{dom}\ \mathit{editorLocation} = \mathit{editors}$
$\mathbf{ran}\ \mathit{storedIn} \subseteq \mathbf{dom}\ \mathit{bookshelfLocation} = \mathit{bookshelves}$
$\mathbf{ran}\ \mathit{inMessageBox} \subseteq \mathbf{dom}\ \mathit{messageBoxLocation} = \mathit{messageBoxes}$

$$\begin{aligned}
&\forall\, p : \mathit{Place};\ d : \mathit{Document} \mid (d \mapsto p) \in \mathit{somewhereIn} \bullet \\
&\qquad d \in \mathbf{dom}\ (\mathit{storedIn} \rhd \mathbf{dom}\ (\mathit{bookshelfLocation} \rhd \{p\})) \vee \\
&\qquad d \in \mathbf{ran}\ (\mathbf{dom}\ (\mathit{editorLocation} \rhd \{p\}) \lhd \mathit{currentDoc}) \vee \\
&\qquad d \in \mathbf{dom}\ (\mathit{inMessageBox} \rhd \mathbf{dom}\ (\mathit{messageBoxLocation} \rhd \{p\}))
\end{aligned}$$

$$\begin{aligned}
&\forall\, p : \mathit{Place} \mid p \in \mathit{places} \bullet \langle \mathbf{dom}\ (\mathit{storedIn} \rhd \mathbf{dom}\ (\mathit{bookshelfLocation} \rhd \{p\})), \\
&\qquad \mathbf{ran}\ (\mathbf{dom}\ (\mathit{editorLocation} \rhd \{p\}) \lhd \mathit{currentDoc}), \\
&\qquad \mathbf{dom}\ (\mathit{inMessageBox} \rhd \mathbf{dom}\ (\mathit{messageBoxLocation} \rhd \{p\}))\rangle \\
&\qquad\qquad \mathbf{partition}\ (\mathbf{dom}\ (\mathit{somewhereIn} \rhd \{p\})
\end{aligned}$$

$\forall\, p : \mathit{DocumentEditor} \mid p.\mathit{type} = \mathit{privateEditor} \bullet \#\,(p \lhd \mathit{usedBy}) \leq 1$

Definition (Learning Context): A *learning context* is defined by a root place and other places contained directly and indirectly by the root place including the entities (e.g., actors, document, and tools) currently existing in these places. A learning context specifies a situation where necessary resources are provided or prepared for performing certain kinds of learning actions. The learning context is dynamic and evolves over time.

A virtual institute is the overall learning context in which all learning-related activities occur. A building with agents, documents, and tools is regarded as a learning context too, in which some specific actions will be carried out. For example, the system's functionality to store, search, borrow, and return documents is distributed across different rooms which are grouped as a building, called library. Instructional buildings can be used to organize relevant rooms that may be created for a course, for the actors who have the same learning interest, for solving a given problem, for a project, etc. A room is also created for a certain purpose so that necessary tools and relevant documents are arranged in the room and it is assigned to relevant agents. Such a room is also regarded as a learning context, in which specific actions such as lecture, discussing, chatting, designing, inquiring, and doing homework can be carried out.

— LearningContext —
$\mathit{VirtualInstitute}$
$\mathit{root} : \mathit{Place}$

$root \in places$

$\{ p : Place \mid (root \mapsto p) \in contains^{+} \bullet p \}$

$actorLocation \rhd \{ p : Place \mid (root \mapsto p) \in contains^{+} \bullet p \}$

$somewhereIn \rhd \{ p : Place \mid (root? \mapsto p) \in contains^{+} \bullet p \}$

$editorLocation \rhd \{ p : Place \mid (root? \mapsto p) \in contains^{+} \bullet p \}$

$bookshelfLocation \rhd \{ p : Place \mid (root? \mapsto p) \in contains^{+} \bullet p \}$

$messageBoxLocation \rhd \{ p : Place \mid (root? \mapsto p) \in contains^{+} \bullet p \}$

$calendarLocation \rhd \{ p : Place \mid (root? \mapsto p) \in contains^{+} \bullet p \}$

There are two kinds of relationship between contexts: *nested context* and *connected context*. If a learning context contains other learning contexts, these learning contexts are called nested contexts. Connected contexts denote two or more learning contexts, which are connected in certain ways so that people can navigate among them. For example, the library and instructional buildings are connected contexts. People can go to the library and borrow documents and then go to an instructional building to take a course.

$\_isNested\_ : LearningContext \leftrightarrow LearningContext$
$\_isConnected\_ : LearningContext \leftrightarrow LearningContext$

$$\begin{aligned}
\_isNested\_ = \{ & \forall c_1, c_2 : LearningContexts \mid \\
& c_1.root \subseteq c_2.root \bullet \\
& c_1 \mapsto c_2 \}
\end{aligned}$$

$$\begin{aligned}
\_isConnected\_ = \{ & \forall c_1, c_2 : LearningContexts \mid \\
& (\exists commonParent : places \mid \\
& (commonParent \mapsto c_1.root) \notin contains \wedge \\
& (commonParent \mapsto c_2.root) \notin contains) \bullet \\
& c_1 \mapsto c_2 \}
\end{aligned}$$

Up to now the major data types and abstract states are defined. We can now start defining the various operations that make up a virtual institute.

4.3.3.2 Construction and Change of Learning Contexts

Starting from this subsection, the operations on the defined abstract states are specified formally. This specification is not complete. We ignore error conditions, so that each operation is described as a partial operation. The preconditions of the partial operations are described as predicates that relate the input and output variables. In addition, some operations are ignored as well, because they are not important or are similar to another operation described. A learning context is a part of the virtual institute abstract state and the whole virtual institute itself is a learning context.

Therefore, we only describe the operations on a virtual institute. First of all, we describe the initial state of a virtual institute.

4.3.3.2.1 Initial State of A Virtual Institute

The initial state of a virtual institute is specified by the following schema.

```
┌─ InitVirtualInstitute ─────────────────────────────
│ VirtualInstitute'
├──────────────────────────────
│ dw_1 = = ( μ DoorView | name = 'admin.')
│ dw_2 = = ( μ DoorView | name = 'library')
│ dw_3 = = ( μ DoorView | name = 'dormitory')
│ dw_4 = = ( μ DoorView | name = 'instruct.')
│ dw_5 = = ( μ DoorView | name = 'campus')
│ dw_6 = = ( μ DoorView | name = 'campus')
│ dw_7 = = ( μ DoorView | name = 'campus')
│ dw_8 = = ( μ DoorView | name = 'campus')
│ dw_9 = = ( μ DoorView | name = 'room for definition tools')
│ dw_10 = = ( μ DoorView | name = 'room for searching document')
│ dw_11 = = ( μ DoorView | name = 'admin.')
│ dw_12 = = ( μ DoorView | name = 'library' )
│
│ c = = (μ Place | name = 'campus' ∧
│                  placeType = campus ∧
│                  owner = ∅)
│ a = = (μ Place | name = 'admin.' ∧
│                  placeType = administrationalBuilding ∧
│                  owner = ∅ )
│ l = = (μ Place | name = 'library' ∧
│                  placeType = library ∧
│                  owner = ∅)
│ d = = (μ Place | name = 'dormitory' ∧
│                  placeType = dormitory ∧
│                  owner = ∅)
│ i = = (μ Place | name = 'instruct.' ∧
│                  placeType = instructionalBuilding ∧
│                  owner = ∅)
│ dr = = (μ Place | name = 'room for definition tools' ∧
│                  placeType = specificRoom ∧
│                  owner = ∅)
│ sr = = (μ Place | name = 'room for searching document' ∧
│                  placeType = instructionalBuilding ∧
│                  owner = ∅)
│
```

$d_1 == (\mu \text{ Door} \mid status = true \wedge doorType = concrete)$
$d_2 == (\mu \text{ Door} \mid status = true \wedge doorType = concrete)$
$d_3 == (\mu \text{ Door} \mid status = true \wedge doorType = concrete)$
$d_4 == (\mu \text{ Door} \mid status = true \wedge doorType = concrete)$
$d_5 == (\mu \text{ Door} \mid status = true \wedge doorType = concrete)$
$d_6 == (\mu \text{ Door} \mid status = true \wedge doorType = concrete)$

places' = {c, a, l, d, i, dr, sr }
campus' = c
dormitory' = d

$contains' = \{c \mapsto a, c \mapsto l, c \mapsto d, c \mapsto i, a \mapsto dr, l \mapsto sr\}$

$hasDoorView' = \{c \mapsto \{ dw_1, dw_2, dw_3, dw_4 \},$
$a \mapsto \{ dw_5, dw_9 \},$
$l \mapsto \{ dw_6, dw_{10} \},$
$d \mapsto \{ dw_7 \},$
$i \mapsto \{ dw_8 \},$
$dr \mapsto \{ dw_{11} \},$
$sr \mapsto \{ dw_{12} \}\}$

$approach' = \{ dw_1 \mapsto d_1, dw_5 \mapsto d_1, dw_2 \mapsto d_2, dw_6 \mapsto d_2,$
$dw_3 \mapsto d_3, dw_7 \mapsto d_3, dw_4 \mapsto d_4, dw_8 \mapsto d_4,$
$dw_9 \mapsto d_5, dw_{11} \mapsto d_5, dw_{10} \mapsto d_6, dw_{12} \mapsto d_6\}$

$connect' = \{(d_1, c \mapsto a), (d_2, c \mapsto l), (d_3, c \mapsto d),$
$(d_4, c \mapsto l), (d_5, a \mapsto dr), (d_6, l \mapsto sr) \}$

aGroupDefinitionTool = = (μ GroupDefinitionTool)
aKnowledgeStructureDefinitionTool = = (μ KnowledgeStructureDefinitionTool)
aCollaborationProtocolDefinitionTool = = (μ CollaborationProtocolDefinitionTool)
aDocumentSearchEngine = = (μ DocumentSearchEngine)
aSCPDefinitionTool = = (μ SCPDefinitionTool)

$specificToolLocation' = \{aGroupDefinitionTool \mapsto dr,$
$aKnowledgeStructureDefinitionTool \mapsto dr,$
$aCollaborationProtocolDefinitionTool \mapsto dr,$
$aSCPDefinitionTool \mapsto dr,$
$aDocumentSearchEngine \mapsto sr \}$

agent' = ∅
document' = ∅
actorLocation' = ∅
editorLocation' = ∅
bookshelfLocation' = ∅
messageBoxLocation' = ∅
calendarLocation' = ∅

somewhereIn’ = $\emptyset$
currentDoc’ = $\emptyset$
usedBy’ = $\emptyset$
storedIn’ = $\emptyset$
inMessageBox’ = $\emptyset$
connectedTo’ = $\emptyset$

The initial state of a virtual institute is the state generated by applying the initialization operation when creating a new virtual institute. The system creates a set of door views, a set of places including campus and dormitory, a set of doors, a set of specific tools. The relations among these entities are also generated. However, the set of agents and the set of document are still empty. Editor, bookshelf, message box, and calendar are not installed in any place.

4.3.3.2.2 Login/logout, Movement, and Construction of Institute Space

When a user login in a virtual institute, the system will check whether a user with that name login in the virtual institute for the first time by using a predicate $\forall$ a : Actor | a $\in$ actors $\bullet$ userName? $\neq$ a.name. If the predicate is true, this user will be regarded as a newcomer. Otherwise, the user is already a member of actors in this virtual institute. Therefore two alternative operations are described for the login operation. That is,

Login $\triangleq$ LoginFirstTime $\vee$ LoginAgain.

When a user logs in to a virtual institute for the first time, the user will be registered in the virtual institute as a new actor and a home will be created for the actor. Some variables change such as actors (adding the newcomer), places (adding the newly created home), bookshelfLocation (a new bookshelf is installed in the home). Some variables keep unchanged such as groups.

— LoginFirstTime —
Δ VirtualInstitute
userName? : STRING

$\forall$ a : Actor | a $\in$ actors $\bullet$ userName? $\neq$ a.name

newcomer = = (μ Actor | name = userName?)
homeEntrance = = (μ DoorView | name = userName?)
homeExit = = (μ DoorView | name = ‘dormitory’)
userHome = = (μ Place | name = userName? $\wedge$
placeType = home $\wedge$
owner = newcomer)
homeDoor = = (μ Door | status = true $\wedge$ doorType = concrete)
aBookshelf = = (μ Bookshelf)
aCalendar = = (μ Calendar)
aMessageBox = = (μ MessageBox)

actors’ = actors $\cup$ { newcomer }
groups’ = groups
agents’ = agents $\cup$ { newcomer }
actorLocation’ = actorLocation $\cup$ { a $\mapsto$ campus }
places’ = places $\cup$ { userHome }
contains’ = contains $\cup$ { dormitory $\mapsto$ userHome }
hasDoorView’ = hasDoorView $\cup$ { dormitory $\mapsto$
(**second** ({dormitory} $\lhd$ hasDoorView) $\cup$ { homeEntraince }),
userHome $\mapsto$ { homeExit }}
approach’ = approach $\cup$
{ homeEntraince $\mapsto$ userHome, homeExit $\mapsto$ dormitory }
connect’ = connect $\cup${(homeDoor, dormitory $\mapsto$ userHome)}
bookselfLocation’ = bookshelfLocation $\cup$ { aBookshelf $\mapsto$ userHome }
calendarLocation’ = calendarLocation $\cup$ { aCalendar $\mapsto$ userHome }
messageBoxLocation’ = messageBoxLocation $\cup$ { aMessageBox $\mapsto$ userHome }
editorLocation’ = editorLocation
somewhereIn’ = somewhereIn
currentDoc’ = currentDoc
usedBy’ = usedBy
storedIn’ = storedIn
inMessageBox’ = inMessageBox
specificToolLocation’ = specificToolLocation
connectedTo’ = connectedTo
documents’ = documents
documentRefs’ = documentRefs
referTo’ = referTo

When a user who has registered in the virtual institute logins again, only the function actorLocation adds a new member that represents the actor is in the campus. In order to shorten the description of this operation, a schema is defined.

Δ ActorChangeLocation $\triangleq$ **Ξ** VirtualInstitute \ (actorLocation, actorLocation’) $\wedge$
Δ VirtualInstitute

Here, hiding a particular before and after component in **Ξ** VirtualInstitute \ (actorLocation, actorLocation’) gives a before and after state that does not have actorLocation and actorLocation’, but still has all the other components, unchanged. Conjoining this with **Δ** VirtualInstitute reintroduces the declaration of actorLocation and actorLocation’, and any predicate involving them, but does not include the predicate actorLocation = actorLocation’. Hence **Ξ** VirtualInstitute \ (actorLocation, actorLocation’) $\wedge$ **Δ** VirtualInstitute is a schema describing a before and after state of VirtualInstitute that includes all the constructs on VirtualInstitute and VirtualInstitute’, and in addition has all the components, except actorLocation, unchanged.

```
┌─ LoginAgain ──────────────────────────────────
│ Δ ActorChangeLocation
│ userName? : STRING
├──────────────────────────
│ {∃ a : Actor | a ∈ actors • userName? = a.name }
│
│ actorLocation' = actorLocation ∪ { a ↦ campus }
└───────────────────────────────────────────────
```

An actor can move from one place to another place by choosing a door view. For a successful movement, the door view should belong to the place in which the actor is currently located and the status of the door referred by the door view should be 'open'. The movement also depends on the type of the door. If the predicate is not met, an operation will handle the error. The combination of these two operations is the complete description for the move operation. That is,

$\text{Move} \triangleq \text{MoveOK} \lor \text{MoveError}$

Here, only the successful move operation is described.

```
┌─ MoveOK ──────────────────────────────────────
│ Δ ActorChangeLocation
│ a? : Actor
│ doorView? : DoorView
├──────────────────────────
│ {a?} ◁ actorLocation ≠ ∅ ∧ doorView? ∈ hasDoorView (|{a?} ◁ actorLocation |)
│ ∧ (approach doorView?).status = true
│
│ actorLocation' = actorLocation ⊕ { a? ↦
│   if (approach doorView?).doorType = concrete
│   then if actorLocation a? = first (approach ⨾ connect doorView?)
│        then second (approach ⨾ connect doorView?)
│        else first (approach ⨾ connect doorView?)
│   else second (approach ⨾ connect doorView?) }
│ places' = places
│ contains' = contains
│ hasDoorView' =  hasDoorView
│ approach' = approach
│ connect' = connect
│ actorLocation' = actorLocation
└───────────────────────────────────────────────
```

When an actor leaves the virtual institute in which the actor logged in, s/he will leave from the place in which the actor is located and the editors in which the actor currently works.

Logout
Δ ActorChangeLocation
a? : Actor

$\{a?\} \lhd actorLocation \neq \emptyset$

$actorLocation' = \{ a \} \ntriangleleft actorLocation$
$usedBy' = usedBy \ntriangleright \{ a \}$
$editorLocation' = editorLocation$

It is allowed for an actor to create a new instructional building in the campus, or to create a public room in an instructional building.

Δ InstituteSpaceChange $\triangleq$ [Δ VirtualInstitute; Ξ ToolBase; Ξ Community; Ξ HyperdocumentBase]

CreateInstructionalBuildingOK
Δ InstituteSpaceChange
a? : Actor
buildingName? : STRING

$\{a?\} \lhd actorLocation \neq \emptyset \wedge actorLocation\ a? = campus$

$buildingEntrance == (\mu\ DoorView \mid name = buildingName?)$
$buildingExit == (\mu\ DoorView \mid name = (actorLocation\ a?).name)$
$newBuilding == (\mu\ Place \mid name = buildingName? \wedge$
$placeType = instructionalBuilding \wedge$
$owner = a?)$
$buildingDoor == (\mu\ Door \mid status = true \wedge doorType = concrete)$

$places' = places \cup \{ userHome \}$
$contains' = contains \cup \{ campus \mapsto newBuilding \}$
$hasDoorView' = hasDoorView \cup \{ campus \mapsto$
$(\mathbf{second}\ (\{campus\} \lhd hasDoorView) \cup \{ buildingEntrance \}),$
$newBuilding \mapsto \{ buildingExit \}\}$
$approach' = approach \cup$
$\{ buildingEntrance \mapsto newBuilding, buildingExit \mapsto campus \}$
$connect' = connect \cup \{(buildingDoor, (actorLocation\ a?) \mapsto newBuilding)\}$
$actorLocation' = actorLocation$
$bookselfLocation' = bookshelfLocation$
$calendarLocation' = calendarLocation$
$messageBoxLocation' = messageBoxLocation$
$editorLocation' = editorLocation$
$somewhereIn' = somewhereIn$
$currentDoc' = currentDoc$
$usedBy' = usedBy$

storedIn’ = storedIn
inMessageBox’ = inMessageBox
specificToolLocation’ = specificToolLocation
connectedTo’ = connectedTo

CreatePublicRoomOK

Δ InstituteSpaceChange
a? : Actor
roomName? : STRING

$\{a?\} \lhd$ actorLocation $\neq \emptyset \wedge$
    ((actorLocation a?).placeType = instructionalBuilding
    $\vee$ (actorLocation a?).placeType = publicRoom)

roomEntrance = = (μ DoorView | name = roomName?)
roomExit = = (μ DoorView | name = (actorLocation a?).name)
newRoom = = (μ Place | name = roomName? $\wedge$
                placeType = publicRoom
                owner = ?)
roomDoor = = (μ Door | status = true $\wedge$ doorType = concrete)
aBookshelf = = (μ Bookshelf)
aCalendar = = (μ Calendar)
aMessageBox = = (μ MessageBox)

places’ = places $\cup$ { newRoom }
contains’ = contains $\cup$ { (actorLocation a?) $\mapsto$ newRoom }
hasDoorView’ = hasDoorView $\cup$ { (actorLocation a?) $\mapsto$
    (**second** ({actorLocation a?} $\lhd$ hasDoorView) $\cup$ { roomEntrance }),
        newRoom $\mapsto$ { roomExit },
approach’ = approach $\cup$
    { roomEntraince $\mapsto$ newRoom, roomExit $\mapsto$ (actorLocation a?) }
connect’ = connect $\cup$ {(roomDoor, (actorLocation a?) $\mapsto$ newRoom)}
bookselfLocation’ = bookshelfLocation $\cup$ { aBookshelf $\mapsto$ newRoom }
calendarLocation’ = calendarLocation $\cup$ { aCalendar $\mapsto$ newRoom }
messageBoxLocation’ = messageBoxLocation $\cup$ { aMessageBox $\mapsto$ newRoom }
actorLocation’ = actorLocation
bookselfLocation’ = bookshelfLocation
calendarLocation’ = calendarLocation
messageBoxLocation’ = messageBoxLocation
editorLocation’ = editorLocation
somewhereIn’ = somewhereIn
currentDoc’ = currentDoc
usedBy’ = usedBy
storedIn’ = storedIn
inMessageBox’ = inMessageBox

specificToolLocation' = specificToolLocation
connectedTo' = connectedTo

All doors are open when being created. An actor can close a door and open again if the actor is the owner of the place of the inside door. Here is the successful operation for closing a door

CloseDoorOK
Δ InstituteSpaceChange
a? : Actor
doorView? : DoorView

$\{a?\} \lhd$ actorLocation $\neq \varnothing \wedge$
doorView? $\in$ **ran** ({actorLocation a?} $\lhd$ hasDoorView) $\wedge$
(approach doorView?).status = true $\wedge$
((approach doorView?).doorType = concrete $\wedge$
(**second** (approach $\fatsemi$ connect doorView?)).owner = a?) $\vee$
(approach doorView?).doorType = virtual)

(approach doorView').status = false
places' = places
contains' = contains
hasDoorView' = hasDoorView
approach' = approach
connect' = connect
actorLocation' = actorLocation
bookselfLocation' = bookshelfLocation
calendarLocation' = calendarLocation
messageBoxLocation' = messageBoxLocation
editorLocation' = editorLocation
somewhereIn' = somewhereIn
currentDoc' = currentDoc
usedBy' = usedBy
storedIn' = storedIn
inMessageBox' = inMessageBox
specificToolLocation' = specificToolLocation
connectedTo' = connectedTo

Some operations such as creating a virtual door, moving forward and backward, and removing a place are ignored, because the specification of these operations is similar to those for operating a concrete door, except for being only an uni-directed connection.

4.3.3.2.3 Community Operations

Actors can define the community structure of a virtual institute by using the group definition tool. The community structure defined in this way represents the formal structure among agents, such as organizations, project teams, and roles. These groups do not refer to the concept of communities of practice defined in situated learning theory. Communities of practice are formed and developed by actors informally in a virtual institute. The groups defined here will be used to define PBL protocols and PBL-plan (see section 4.5 and 4.6).

The operation of *CreateGroup* will add a new group in the community.

```
── CreateGroupOK ──────────────────────────
Δ Community
name? : STRING
───────────────────────────────────────────
∀ g : Group | g ∈ groups • g.name ≠ name?

let newGroup = = ( μ Group | name = name? ) •
        groups' = groups ∪{newGroup}∧ agents' = agents ∪{newGroup}
aMemberOf' = aMemberOf
aSubGroupOf' = aSubGroupOf
actors' = actors
───────────────────────────────────────────
```

Defining a sub-group of a group is done by performing a JoinGroup operation. It is important to note that the sub-group to be defined can not be a group that is a parent directly or indirectly of the other group.

```
── JoinGroupOK ────────────────────────────
Δ Community
childGroup?, parentGroup? : Group
───────────────────────────────────────────
childGroup? ∈ groups ∧ parentGroup? ∈ groups
∧ (childGroup? ↦ parentGroup?) ∉ aSubGroupOf
∧ (parentGroup? ↦ childGroup?) ∉ (aSubGroupOf ˜ )⁺

actors' = actors
groups' = groups
agents' = agents
aMemberOf' = aMemberOf
aSubGroupOf' = aSubGroupOf ∪{childGroup? ↦ parentGroup?}
───────────────────────────────────────────
```

A LeaveGroupOK operation describes the situation in which a sub-group of another group is no longer a sub-group of the other group.

```
── LeaveGroupOK ───────────────────────────
Δ Community
```

childGroup?, parentGroup? : Group

childGroup? $\in$ groups $\wedge$ parentGroup? $\in$ groups
$\wedge$ (childGroup? $\mapsto$ parentGroup?) $\in$ aSubGroupOf

actors' = actors
groups' = groups
agents' = agents
aMemberOf' = aMemberOf
aSubGroupOf' = aSubGroupOf \ {childGroup? $\mapsto$parentGroup?}

The DissolveGroupOK operation is used to remove a group. For example, if a project is finished, then the project team will dissolve.

DissolveGroupOK
Δ Community
g? : Group

g? $\in$ groups

actors' = actors
groups' = groups \ {g?}
agents' = agents \ {g?}
aMemberOf' = aMemberOf ⩥{g?}
aSubGroupOf' = {g?} ⩤ aSubGroupOf ⩥{g?}

When an actor participates in a group or ends participation in a group, the operations ParticipateInGroupOK and EndParticipationInGroupOK are used.

ParticipateInGroupOK
Δ Community
a? : Actor
g? : Group

a? $\in$ actors $\wedge$ g? $\in$ groups $\wedge$ (a? $\mapsto$ g?) $\notin$ isMemberOf
aMemberOf' = aMemberOf $\cup$ {a? $\mapsto$ g?}
aSubGroupOf' = aSubGroupOf
actors' = actors
groups' = groups
agents' = agents

EndParticipationInGroupOK
Δ Community
a? : Actor
g? : Group

$a? \in actors \land g? \in groups \land (a? \mapsto g?) \in isMemberOf$

$aMemberOf' = aMemberOf \setminus \{a? \mapsto g?\}$
$aSubGroupOf' = aSubGroupOf$
$actors' = actors$
$groups' = groups$
$agents' = agents$

4.3.3.2.4 Tool Operation

Actors can change a learning context by installing or removing tools. Installing any types of tools leads to a similar change of states. Hence, only the operations to install and remove a document editor are described.

Most of the operations will change the state of the virtual institute. However, some operations change only one of components such as Editors or Bookshelves. Hence the following schemas are useful to shorten the description.

Δ ToolAndDocumentChange $\triangleq$ [Δ VirtualInstitute; Ξ InstituteSpace; Ξ Community]

CreateEditorOK
Δ ToolAndDocumentChange
$a? : Actor$
$e? : DocumentEditor$
$type? : EditorType$

$e? \notin \textbf{dom}\ editorLocation \land$
$\{a?\} \lhd actorLocation \neq \emptyset \land actorLocation\ a? \neq campus$

$newDocument == (\mu\ Document \mid title = \emptyset \land$
$owner = a? \land$
$topic = \emptyset \land$
$texts = \emptyset \land$
$tables = \emptyset \land$
$images = \emptyset\ )$

$e'.type = type?$
$e'.history = \{1 \mapsto newDocument\}$

$editorLocation' = editorLocation \cup \{ e? \mapsto actorLocation\ a? \}$
$currentDoc' = currentDoc \cup \{ e? \mapsto newDocument \}$
$usedBy' = usedBy \cup \{ e? \mapsto a? \}$
$somewhereIn' = somewhereIn \cup \{ newDocument \mapsto actorLocation\ a? \}$
$documents' = documents \cup \{ newDocument \}$

documentRefs’ = documentRefs $\cup$ { newDocument $\mapsto \emptyset$}
referTo’ = referTo
actorLocation’ = actorLocation
bookselfLocation’ = bookshelfLocation
calendarLocation’ = calendarLocation
messageBoxLocation’ = messageBoxLocation
storedIn’ = storedIn
inMessageBox’ = inMessageBox
specificToolLocation’ = specificToolLocation
connectedTo’ = connectedTo

RemoveEditorOK
Δ ToolAndDocumentChange
a? : Actor
e? : DocumentEditor

e? $\in$ **dom** editorLocation $\wedge$ ({e?} $\lhd$ usedBy) $\rhd$ {a?} = $\emptyset$ $\wedge$
((e?.type = computer $\wedge$ e? $\notin$ **ran** connectedTo) $\vee$ e?.type $\neq$ computer)

editorLocation’ = editorLocation \ { e? $\mapsto$ actorLocation a? }
currentDoc’ = currentDoc \ { e? $\mapsto$ currentDoc e?}
usedBy’ = usedBy \ { e? $\mapsto$ a? }
storedIn’ = storedIn $\cup$
{ currentDoc e? $\mapsto$ **first** (bookshelfLocation $\rhd$ {actorLocation a?})}
actorLocation’ = actorLocation
bookselfLocation’ = bookshelfLocation
calendarLocation’ = calendarLocation
messageBoxLocation’ = messageBoxLocation
specificToolLocation’ = specificToolLocation
somewhereIn’ = somewhereIn
inMessageBox’ = inMessageBox
connectedTo’ = connectedTo
documents’ = documents
documentRefs’ = documentRefs
referTo’ = referTo

4.3.3.2.5 Handling Documents

Documents are created by using document editors. As described above, creating a new document editor will create a document. Another way to create a new document is to create a reference to the new document in the currently edited document of the editor. Documents in a virtual institute are connected and form hyperdocuments. A document reference and the referred document are created together.

CreateReferenceAndDocumentOK

Δ ToolAndDocumentChange
a? : Actor
editor? : DocumentEditor
referenceName? : STRING
topic? : STRING

$\{editor?\} \lhd usedBy \rhd \{a?\} \neq \emptyset$

$docRef == (\mu\ DocumentReference \mid name = referenceName?)$
$newDoc == (\mu\ Document \mid title = referenceName?$
$\quad topic = topic? \wedge$
$\quad owner = a? \wedge$
$\quad text = \emptyset \wedge$
$\quad tables = \emptyset \wedge$
$\quad images = \emptyset)$
$documents' = documents \cup \{ newDoc \}$
$documentRefs' = documentRefs \oplus \{ (currentDoc\ editor?) \mapsto$
$\quad$ **second** $(\{currentDoc\ editor?\} \lhd documentRefs) \cup \{ docRef \} \}$
$referTo' = referTo \cup \{ docRef \mapsto newDoc \}$
$actorLocation' = actorLocation$
$editorLocation' = editorLocation$
$bookshelfLocation' = bookshelfLocation$
$calendarLocation' = calendarLocation$
$messageBoxLocation' = messageBoxLocation$
$currentDoc' = currentDoc$
$usedBy' = usedBy$
$somewhereIn' = somewhereIn \cup \{ newDoc \mapsto actorLocation\ a? \}$
$storedIn' = storedIn$
$inMessageBox' = inMessageBox$
$specificToolLocation' = specificToolLocation$
$connectedTo' = connectedTo$

Users can jointly edit a document by adding, removing, or modifying media objects in the content of the document. We take the specification of operation to add a media object as an example for all these operations.

CreateAMediaObjectInDocumentOK

Δ ToolAndDocumentChange
a? : Actor
editor? : DocumentEditor
doc? : Document
object? : MediaObject

$\{editor?\} \lhd usedBy \rhd \{a?\} \neq \emptyset$

$doc'.content = doc?.content \cup \{object?\}$

documents’ = documents
documentRefs’ = documentRefs
referTo’ = referTo
actorLocation’ = actorLocation
editorLocation’ = editorLocation
bookshelfLocation’ = bookshelfLocation
calendarLocation’ = calendarLocation
messageBoxLocation’ = messageBoxLocation
currentDoc’ = currentDoc
usedBy’ = usedBy
somewhereIn’ = somewhereIn
storedIn’ = storedIn
inMessageBox’ = inMessageBox
specificToolLocation’ = specificToolLocation
connectedTo’ = connectedTo

Actors who are using a document editor can navigate in the hyperdocument base by following a link indicated by a document reference. The precondition of this operation is that the actor is a current user of the editor and the document reference connects to a document and can be found on the current editing page of the editor. After performing this operation, the currently edited document of the editor will change to the document referred to. The editing history will add a new item in the history queue. The former currently edited document will be put back to the bookshelf installed in the place in which the editor exists.

FollowReferenceOK
Δ ToolAndDocumentChange
a? : Actor
editor? : DocumentEditor
docReference? : DocumentReference

$\{editor?\} \lhd usedBy \rhd \{a?\} \neq \emptyset \wedge docReference? \in$ **dom** referTo $\wedge$
  $docReference? \in$ **second** $(documentRefs \rhd \{currentDoc\ editor?\})$

$currentDoc' = currentDoc \oplus \{ editor? \mapsto referTo\ docReference?\}$
$storedIn' = storedIn \cup \{ (currentDoc\ editor?) \mapsto$
      first $(bookshelfLocation \rhd \{actorLocation\ a?\})\}$
    $\setminus \{ (referTo\ docReference?) \mapsto$
      first $(bookshelfLocation \rhd \{actorLocation\ a?\})\}$
$editor?.history' = editor?.history \frown \langle referTo\ docReference?\rangle$
actorLocation’ = actorLocation
editorLocation’ = editorLocation
bookshelfLocation’ = bookshelfLocation
calendarLocation’ = calendarLocation
messageBoxLocation’ = messageBoxLocation
usedBy’ = usedBy

somewhereIn’ = somewhereIn
inMessageBox’ = inMessageBox
specificToolLocation’ = specificToolLocation
connectedTo’ = connectedTo
documents’ = documents
documentRefs’ = documentRefs
referTo’ = referTo

As described above, if an actor removes a document editor or navigates to another document, the previously edited document of the document editor will be put in the bookshelf. A document stored in a bookshelf can be opened in two ways.

One way is to take a document from a bookshelf and open it in a new private document editor. After this operation, a new editor is created and the document is the currently edited document of the editor. The document should be removed from the bookshelf.

```
── TakeDocumentFromBookshelfOK ──────────────────
Δ ToolAndDocumentChange
a? : Actor
d? : Document
├──────────────────────────────
```

$\{a?\} \lhd actorLocation \neq \emptyset \wedge$

$d? \in \mathbf{dom}\ storedIn \rhd \mathbf{dom}\ bookshelfLocation \rhd \{actorLocation\ a?\}$

$\mathbf{let}\ e == (\mu : DocumentEditor \mid type = privateEditor \wedge history = \{1 \mapsto d?\})$

$editorLocation' = editorLocation \cup \{e \mapsto actorLocation\ a?\}$

$currentDoc' = currentDoc \cup \{e \mapsto d?\}$

$usedBy' = usedBy \cup \{e \mapsto a?\}$

$storedIn' = storedIn \setminus \{d? \mapsto \mathbf{first}\ (bookshelfLocation \rhd \{actorLocation\ a?\})\}$
actorLocation’ = actorLocation
bookshelfLocation’ = bookshelfLocation
calendarLocation’ = calendarLocation
messageBoxLocation’ = messageBoxLocation
specificToolLocation’ = specificToolLocation
somewhereIn’ = somewhereIn
inMessageBox’ = inMessageBox
connectedTo’ = connectedTo
documents’ = documents
documentRefs’ = documentRefs
referTo’ = referTo

Another way is to take a document from a bookshelf and put it on an existing document editor, which takes the document as the currently edited document. The document should be removed in the bookshelf, but the previous editing document of the editor should be put into the bookshelf.

DragDocumentDropInEditorOK

Δ ToolAndDocumentChange
a? : Actor
d? : Document
e? : DocumentEditor

$\{a?\} \lhd \text{actorLocation} \neq \emptyset \land$

$d? \in \textbf{dom}\ (\text{storedIn} \rhd \textbf{dom}\ (\text{bookshelfLocation} \rhd \{\text{actorLocation } a?\})) \land$

$e? \in \textbf{dom}\ (\text{editorLocation} \rhd \{\text{actorLocation } a?\})$

editorLocation' = editorLocation

$\text{currentDoc}' = \text{currentDoc} \cup \{\ e \mapsto d?\ \}$

$\text{storedIn}' = \text{storedIn} \setminus \{\ d? \mapsto \textbf{first}\ (\text{bookshelfLocation} \rhd \{\text{actorLocation } a?\})\}$

$\qquad \cup \{\ (\text{currentDoc } e?) \mapsto$

$\qquad\qquad \textbf{first}\ (\text{bookshelfLocation} \rhd \{\text{actorLocation } a?\})\}$

$e?.\text{history} = e?.\text{history} \frown \langle d? \rangle$
actorLocation' = actorLocation
usedBy' = usedBy
somewhereIn' = somewhereIn
bookshelfLocation' = bookshelfLocation
calendarLocation' = calendarLocation
messageBoxLocation' = messageBoxLocation
inMessageBox' = inMessageBox
specificToolLocation' = specificToolLocation
connectedTo' = connectedTo
documents' = documents
documentRefs' = documentRefs
referTo' = referTo

Operations to take a document from and to put it into a message box are similar to the operations described above, except that it is needed to choose a place or an actor as the receiver of the document when sending it through the message box.

So far, the data types and some operations of a context-based virtual learning system are formally described.

4.3.3.3 Awareness of Learning Context

The pre-condition of social interaction is that users are able to be aware of the situation in which they are. As mentioned above, a virtual institute is designed as a hypermedia structure, in which a place is modeled as a hypermedia node and actors, documents, and tools located in the place are modeled as content elements of this node. These elements are symbolically presented on the node. Displaying a hypermedia node in the user interface visualizes a learning context. The advantage of this approach is that users of the system can intuitively be aware of and then interact with the learning context. In addition, in such a hypermedia structure, doors are created to connect places. These doors are established by placing 'door views' as navigation buttons on places. Compared to a list of isolated places, these door views enable users to be aware of closely related places and to access to these places conveniently. That is, hypermedia provides navigation facilities and guided tours, which sequence access in ways that make the places and their contents more meaningful. It also provides flexibility for users to reorganize places to form new learning contexts.

Some awareness information can be computed according to the information associated with the place in which an actor is located. These include:

1) Actors in the same place:

$$\begin{aligned}\text{actorsInSamePlace} == \{\ & a : \text{actors} \mid \{a\} \lhd \text{actorLocation} \neq \emptyset \bullet \\ & \textbf{dom}\ (\text{actorLocation} \rhd \{\text{actorLocation}\ a\}\}\end{aligned}$$

2) Tools available in the place:

$$\begin{aligned}\text{toolsInSamePlace} == \{\ & a : \text{actors} \mid \{a\} \lhd \text{actorLocation} \neq \emptyset \bullet \\ & \textbf{dom}\ (\text{editorLocation} \rhd \{\text{actorLocation}\ a\}) \cup \\ & \textbf{first}\ (\text{chatboardLocation} \rhd \{\text{actorLocation}\ a\}) \cup \\ & \textbf{dom}\ (\text{phoneLocation} \rhd \{\text{actorLocation}\ a\}) \cup \\ & \textbf{first}\ (\text{calendarLocation} \rhd \{\text{actorLocation}\ a\}) \cup \\ & \textbf{first}\ (\text{messageBoxLocation} \rhd \{\text{actorLocation}\ a\}) \cup \\ & \textbf{first}\ (\text{bookshelfLocation} \rhd \{\text{actorLocation}\ a\})\ \}\end{aligned}$$

3) Who is working in which editor:

$$\begin{aligned}\text{whoWorksOnEditor} == \{\ & a : \text{actors} \mid \{a\} \lhd \text{actorLocation} \neq \emptyset \bullet \\ & (\ \textbf{dom}\ (\text{editorLocation} \rhd \{\text{actorLocation}\ a\}) \\ & \lhd\ \text{usedBy}\ \}\end{aligned}$$

Some information can be obtained by using simple enquiry operations. Two examples are given below:

1) Which documents stored in the bookshelf installed in an actor's location can be computed by:

$$\begin{aligned}\text{documentListInBookshelf} == \{\ & a : \text{actors} \mid \{a\} \lhd \text{actorLocation} \neq \emptyset \bullet \\ & \textbf{dom}\ (\text{storedIn} \rhd \textbf{first}\ (\text{bookshelfLocation} \rhd \{\text{actorLocation}\ a\}))\}\end{aligned}$$

2) The email address of another actor who is located in the same place can be obtained by computing:

emailAddrOfActor = = (a, another : actors | {a} ◁ actorLocation ≠ ∅ ∧
another ∈ **dom** (actorLocation ▷ {actorLocation a} •
another.emailAddr)

Some indirect enquiry operations can be defined. Two examples are given below:

1) When an actor is reading a document and has questions about the content of the document, s/he may want to ask someone who may be able to provide help. Potential candidates can be computed by:

whoCanHelpMe = = (λ VirtualInstitute; d : Document •
{ Actor | ***θ*** Actor ∈ actors ∧ d.topic ∈ expertise • name })

2) The place in which the creator of a document is located now can be computed by:

```
whereIsOwnerOfDocument = = (λ VirtualInstitute; d : Document •
                              if {d.owner} ◁ actorLocation ≠ ∅
                                  then first ({d.owner} ◁ actorLocation)
                                  else ∅ )
```

A lot of information about a document can be inquired by using a search engine in the library. For examples, where is a given document? Who is the owner or the current users of a given document? What are the documents stored in a given place or in the suitcase of given actor? Which documents are related to a given topic?

Some awareness information is displayed as graphical elements in the user interface. For example, an arrow with a label "talk" between two pictures of actors indicates that these two actors are talking by means of a conversation tool. Some awareness information is retrieved by clicking the graphical unit of the inquired entity and being displayed in pop-up windows. For example, clicking on a picture of an actor leads to poping up a window, which contains personal information of the actor such as name, email address, telephone, expertises, and so on.

4.3.3.4 Social Interaction in a Learning Context

As Wenger [Wenger98] points out, members of a community are informally bound by what they do together and what they learned through their mutual engagement in these activities. Social interaction occurs in all stages of development of CoPs. At a point in time, the users of a virtual learning environment may be distributed in multiple virtual places of a virtual institute. Some of them work individually and others may work in teams. It is possible that some users are not logged in the virtual institute at this time, but they need interaction with each other as well. This subsection describes how to support social interaction when the users are at the same/different time present at the same/different virtual places. Note that geographically co-locate/distributed users can work in the same or in different virtual places in a virtual institute.

4.3.3.4.1 Synchronous Interaction in the Same Virtual Place

By using the speaker tool installed in a virtual place, an actor can listen to and talk to other people in the same virtual place. When an actor wants to make a private conversation with somebody in the same place, s/he can start a conversation tool that supports text-based communication between two conversation partners.

A whiteboard can be used to establish a synchronous session for all users in the same place. Documents dragged onto the whiteboard can be edited collaboratively by using a hypermedia document editor. Any change to the hypermedia document will be propagated to other users' windows, and all users navigate through the hypermedia document together. As they do this, they share the same view of the document; specifically, all users share one scrollbar and change pages simultaneously. This is a pure WYSIWIS (What You See Is What I See) collaboration mode [Stefik86].

Furthermore, if users in the same virtual place want to split into two or more sub-groups such that each sub-group works on different documents in a pure WYSIWIS collaboration mode, then they can simply create a whiteboard for each sub-group and drag the corresponding documents onto the whiteboard. If all sub-groups want to work on the same document, but on different positions of the document, they can create a whiteboard for each sub-group and drag a reference copy of the document onto their whiteboards. In this case, users in the same sub-group work in a pure WYSIWIS collaboration mode and people in different sub-groups work in a relaxed WYSIWIS collaboration mode [Stefik86], but they all work on the same document.

4.3.3.4.2 Synchronous Interaction in different Virtual Places

There are a number of ways to communicate with somebody located in different virtual places. By using a phone tool, an actor can establish audio channels with a partner in a given virtual place.

A virtual computer is used as perceptual metaphor to establish a session for people in different virtual places [Miao99b]. Users working in different virtual places can view and edit the same document in a pure WYSIWIS collaboration mode without leaving their current virtual places by connecting their virtual computers. When the common activity is finished, they can cut the logical connection and then the mode of coupling and sharing will recover to the mode before connection. Therefore, rich and flexible forms of sharing information and of coupling of user interfaces are provided by using virtual computers. The formal descriptions of the connection of computers and of following a document reference in a virtual computer are given below.

```
┌─ ConnectComputersOK ──────────────────────────────
│ Δ ToolAndDocumentChange
│ a? : Actor
│ c? : DocumentEditor
│ another? : DocumentEditor
│ p? : Place
```

{a?} ◁ actorLocation ≠ ∅ ∧ p? ∈ places ∧ actorLocation a? ≠ p? ∧
c? ∈ **dom** (editorLocation ▷ { actorLocation a? }) ∧ c.type = computer ∧
another? ∈ **dom** (editorLocation ▷ { p? }) ∧ another?.type = computer ∧
(c? ↦ another?) ∉ connectedTo

connectedTo’ = connectedTo ∪ { c? ↦ another? }
c?.history’ = c?.history ⁀ < currentDoc another?>
currentDoc’ = currentDoc ⊕ { c? ↦ currentDoc another? }
usedBy’ = usedBy
actorLocation’ = actorLocation
editorLocation’ = editorLocation
bookshelfLocation’ = bookshelfLocation
calendarLocation’ = calendarLocation
messageBoxLocation’ = messageBoxLocation
specificToolLocation’ = specificToolLocation
inMessageBox’ = inMessageBox
somewhereIn’ = somewhereIn
documents’ = documents
documentRefs’ = documentRefs
referTo’ = referTo

— FollowReferenceInComputersOK —
Δ ToolAndDocumentChange
a? : Actor
c? : DocumentEditor
docReference? : DocumentReference

{a?} ◁ actorLocation ≠ ∅ ∧ c? ∈ **dom** (editorLocation ▷ { actorLocation a? }) ∧
c?.type = computer ∧ c? ∉ **dom** connectedTo ∪ **ran** connectedTo
{c?} ◁ usedBy ▷ {a?} ≠ ∅ ∧ docReference? ∈ **dom** referTo ∧
docReference? ∈ **second** (documentRefs ▷ {currentDoc editor?})

c?.history = c?.history ⁀ < referTo docReference?>
currentDoc’ = currentDoc ⊕ { c? ↦ referTo docReference?}

(∀ computer : DocumentEditor |
    (c? ↦ computer) ∈ $(\text{connectedTo}^{\sim})^{+} \cup \text{connectedTo}^{+}$ •
        computer.history = computer.history ⁀ < referTo docReference?> ∧
        currentDoc’ = currentDoc ⊕ { computer ↦ referTo docReference?}

editorLocation’ = editorLocation
usedBy’ = usedBy
connectedTo’ = connectedTo

```
actorLocation’ = actorLocation
bookshelfLocation’ = bookshelfLocation
calendarLocation’ = calendarLocation
messageBoxLocation’ = messageBoxLocation
specificToolLocation’ = specificToolLocation
inMessageBox’ = inMessageBox
somewhereIn’ = somewhereIn
documents’ = documents
documentRefs’ = documentRefs
referTo’ = referTo
```

4.3.3.4.3 Asynchronous Interaction in the Same Virtual Place

The chatboard tool can also support asynchronous communication among users in the same virtual place. A chatboard records all contributions of users of this tool as permanent information, which can be retrieved at any time.

Another possiblity is to use a shared whiteboard. As mentioned above, a session can be established by creating a shared whiteboard in a place. An actor can join the session by clicking on the whiteboard icon, and a local window of the shared whiteboard will open for the actor. Actors in this place can view and edit the same document in the session via local window of the whiteboard. A user can leave a session by closing his/her local window of the shared whiteboard. Even in the case that all users leave the session, the session is still active. When someone joins the session by openning a local window of the shared whiteboard later, s/he will find that the working context keeps unchanged and can work continually. The session ends when the shared whiteboard is removed. By using whiteboards, actors can asynchronously collaborate in the same place.

4.3.3.4.4 Asynchronous Interaction in different Virtual Places

Message boxes can be used to transfer documents from one virtual place to another. After being transferred, a document can be taken and used later on. Virtual computers can also be used to establish an asynchronous session for users working in different virtual places. A user can connect his/her virtual computer to a virtual computer located in another place. The user of that virtual computer perhaps has already left the session. However, the user can work in this session, and his change are recorded for later use.

4.3.4 Related Work and Discussion

In the literature we find three categories of virtual learning environments: document-based, conferencing-based, and room-based systems. In the following, we look briefly at each of these systems and compare them to our approach of a context-based virtual learning environment.

Document-based learning systems primarily serve as a repository for documents and control the access to documents. The users of this kind of systems can interact only indirectly with each other by navigating through the information space, and by viewing and manipulating information items in the shared database (e.g., CSILE [Scardamalia94] and Collaboratory Notebook [Edelson94]). Unlike context-based systems, these systems do not attempt to provide a very sophisticated context, but one which is based mainly on a document structure and which supports limited social interaction.

Conferencing-based systems support real-time learning activities such as CCL [Koschmann90]. They support typical class and seminar style activities such as discussion and lecture. For example, a PBL environment described in [Cameron99] is based on Microsoft NetMeeting [Summers99]. Such a kind of systems are developed by adopting conferencing-based approach. Unlike context-based systems, these systems provide limited persistence of documents and do not support asynchronous activities.

Room-based systems are often based on a number of isolated virtual rooms, which often include a fixed set of embedded tools such as whiteboard, audio/video tools and so on. Users in the same room can view and edit information items in a shared workspace simultaneously, and they can talk to others and see others in the same room (e.g., TeamRooms [Roseman96] and VITAL [Pfister98a]). It is important to note that such systems are general-purpose virtual learning environments. Although there is no PBL-specific support, they can be used to conduct PBL activities. Context-based systems are derived from room-based systems, but both types of systems differ in several aspects: Firstly, in room-based systems, the organization of people, and the structure of documents (if any) rely on the structure of rooms, and tools belong to certain rooms. In context-based systems, the organization of people and documents is dynamic and flexible, rather than being fixed in a particular place. Some tools can belong to actors and be carried around. Secondly, in room-based systems the room structure and the internal structure of a room are rigid. In context-based systems, the place structure and the internal structure of a place can be reorganized and customized to establish a set of rich, dynamic, purpose-specific environments for situated learning. For example, a driving school may consist of virtual lecture hall, virtual practice grounds, and so on. Thirdly, in room-based systems, social interaction is limited within the scope of certain rooms. In context-based systems, a context may be formed beyond the boundaries of certain places. Thus, social interaction can be carried out across places.

4.3.5 Summary

Based on the theory of situated learning, a context-based virtual learning environment has been designed. The virtual learning environment enables the learners themselves to create and modify their learning environments. They therefore provide a customized learning context in which learning processes and communications between learners can be situated. The characteristics of this approach are: the use of a set of perceptual metaphors, the flexible combination of these metaphors within the learning environment, and the support for awareness of the learning context and the social interaction within it. In this section, the design of a context-based learning environment was specified formally.

The virtual institute metaphor allows users to reuse the culture existing in real learning environments. The users of the context-based virtual learning environment can intuitively use basic educational tools and learning resources as they do in real learning environments. It is important to note that the virtual learning environment described in this chapter is a general-purpose learning environment. That is, domain-specific support hasn't be mentioned. In fact, tools for acquiring and applying domain-specific knowledge (such as experimental instruments for science learning) are very important for supporting the idea of a context-based learning environment. These tools should be provided when developing a domain-specific learning support system.

4.4 PBL-net: An Activity-oriented, Graphical Knowledge Representation Language for PBL

In section 4.1, it is suggested that the role of cultural mediation for the PBL activity should be addressed when designing a virtual learning environment. In the last section, a context-based approach for a virtual learning environment is presented. In such a virtual learning environment a major part of the culture used in real learning environments can be applied. This virtual learning environment can support many forms of learning and many knowledge domains. However, in order to support problem based learning, PBL-specific culture should be supported in a virtual learning environment. Language is one of the important cultural factors. Thus, in this section we discuss a knowledge representation language for PBL.

In order to perform a problem based learning activity collaboratively, participants have to represent their ideas to other collaborators and understand others' ideas so that they can construct consistent, shared knowledge. A PBL-specific language is helpful to exchange ideas and information within PBL communities. When performing problem based learning in a virtual learning environment, rich communication channels are lost. Exchanging ideas and information mainly relies on a shared database in which the participants of a PBL activity externally represent their knowledge and transfer information in an electronic form. A knowledge representation method, which is used to organize ideas and information in such a shared database, is crucial to facilitate mutual understanding and to construct shared knowledge. There are some methods of knowledge representation available in the education area such as Concept Maps [Jonassen93] and RESRA [Wan94a] [Wan94b]. However, these methods are designed to support traditional subject based learning. According to [Woods96], PBL forces the learners to acquire knowledge in the context of needing it to solve a problem. Consequently, knowledge is acquired in formats different from subject based learning. In such traditional settings, knowledge is well structured and learning materials are prepared in advance by the teachers before transferring it in the form of a lecture presentation. In contrast, in PBL, knowledge is ill-structured and constructed collaboratively by learners in the course of the learning process. Some computer-supported PBL environments made initial efforts to provide mechanisms to facilitate knowledge representation (e.g., Web-SMILE [Guzdial97] and Belvedere [Suthers97]). However, a systematic approach to develop a knowledge representation method for PBL and a PBL-specific knowledge representation language are still missing.

In this section, the main principles of constructivist and situated learning are described. According to these principles, a model of collaborative learning is developed. This model is used to derive requirements for the design of a graphical knowledge representation method for collaborative learning. The main body of this chapter describes an activity-oriented approach to knowledge representation for learning. The approach is applied specifically to problem-based learning. Then we compare our approach with other graphical knowledge representation methods for education and with other problem-based learning support systems described in Chapter 3. The last subsection summarizes the work described in this section.

4.4.1 Theoretical Background and a Conceptual Model of Collaborative Learning

The theoretical background of the proposed model is based on two paradigms: firstly, the constructivist paradigm, whereby learners actively construct their own knowledge as they interact with the environment. They learn through conflicts and by socially negotiating with others. The second paradigm is situated learning, which also stresses the importance of the environment in which learning is carried out through purposeful activities. In the following subsection the main principles of these two paradigms are described.

4.4.1.1 The Constructivist Learning Perspective

From a constructivist view, learning is the process of constructing knowledge - not merely acquiring it - in social environments [Brooks93]. Knowledge is socially constructed and taken-to-be-shared within communities of learners [Roth92]. "Knowledge is a dialectical process the essence of which is that individuals have opportunities to test their constructed ideas on others, persuade others of the virtue of their thinking, and be persuaded" [Cognition and Technology Group at Vanderbilt 91]. “The role of education in a constructivist view is to show students how to construct knowledge, to promote collaboration with others to show the multiple perspectives that can be brought to bear on a particular problem, and to arrive at self-chosen positions (emphasis added) to which they can commit themselves, while realizing the basis of other views with which they may disagree" [Cunningham93].

John R. Savery and Thomas M. Duffy [Savery95] characterize the philosophical view of constructivism in terms of three primary propositions that are listed as follows:

“1. *Understanding is in our interactions with the environment*. This is the core concept of constructivism. We cannot talk about what is learned separately from how it is learned, as if a variety of experiences all lead to the same understanding. Rather, what we understand is a function of the content, the context, the activity of the learner, and, perhaps most importantly, the goals of the learner. Since understanding is an individual construction, we cannot share understandings but rather we can test the degree to which our individual understandings are compatible. An implication of this proposition is that cognition is not just within the individual but rather it is a part of the entire context, i.e., cognition is distributed.

2. *Cognitive conflict or puzzlement is the stimulus for learning and determines the organization and nature of what is learned*. When we are in a learning environment, there is some stimulus or goal for learning -- the learner has a purpose for being there. That goal is not only the stimulus for learning, but it is a primary factor in determining what the learner attends to, what prior experience the learner brings to bear in constructing an understanding, and, basically, what under standing is eventually constructed. In Dewey's terms it is the ‘problematic’ that leads to and is the organizer for learning [Dewey38b] [Roschelle92]. For Piaget it is the need for accommodation when current experience cannot be assimilated in existing schema [Piaget77] [vonGlaserfeld89]. We prefer to talk about the learner's ‘puzzlement’ as being the stimulus and organizer for learning since this more readily suggests both intellectual and pragmatic goals for learning. The important point, however, is that it is the goal of the learner that is central in considering what is learned.

3. *Knowledge evolves through social negotiation and through the evaluation of the viability of individual understandings*. The social environment is critical to the development of our individual understanding as well as to the development of the body of propositions we call knowledge. At the individual level, other individuals are a primary mechanism for testing our understanding. Collaborative groups are important because we can test our own understanding and examine the understanding of others as a mechanism for enriching, interweaving, and expanding our understanding of particular issues or phenomena. As vonGlaserfeld [vonGlaserfeld89] has noted, other people are the greatest source of alternative views to challenge our current views and hence to serve as the source of puzzlement that stimulates new learning.”

4.4.1.2 The Situated Learning Perspective

“The activities of a domain are framed by its culture. Their meaning and purpose are socially constructed through negotiations among present and past members. Activities thus cohere in a way that is, in theory, if not always in practice, accessible to members who move within the social framework. These coherent, meaningful, and purposeful activities are authentic. … Authentic activities then, are most simply defined as the ordinary practices of the culture. … Within a culture, ideas are exchanged and modified and belief systems developed and appropriated through conversation and narratives” [Brown89].

According to [Education by Design], four important principles of situated learning are as follows:

1) “Learning takes place through purposeful activities, driven by dilemma in authentic situations.
2) Learners construct meaning in shared social contexts or in ‘communities of practice.’
3) Learning occurs through direct engagement with objects and tools.

4) Learning is the result of reflecting on experience, engaging in dialogue with others, and negotiating meaning within specific contexts."

Situated learning emphasizes learning by doing, in which the focus is on an activity rather than simply on the subject content. Learning does not take place in the abstract or in isolation, but in a rich learning context, where learners work in collaboration with other learners.

4.4.1.3 A Conceptual Model of Collaborative Learning

The paragraphs below describe a conceptual model of collaborative learning, which is developed according to the constructivist and situated learning principles outlined above. The model focuses on the way in which both individual knowledge and shared knowledge evolve during collaborative learning activities [Miao00a]. We do not model collaborative learning in its entirety, but rather we focus specifically on the role played by shared artifacts during collaborative learning. We then go on to use this model to derive requirements for the design of a graphical knowledge representation tool, which supports problem-based learning.

In Figure 4.6, the small circles at the top of the diagram represent the individual memories of two people, while the big lower circle represents the shared artifact that is collaboratively constructed by the learners. Both the knowledge which is held inside individual memory and the information that is carried by the shared artifact, can each be defined as being in one of two states: conflict or coherent. At the individual level, the learner constructs new knowledge by integrating the new information into his own cognitive structure. When new information contradicts existing knowledge of the individual then we say that conflict occurs. The learner must therefore reconcile the conflict, perhaps by modifying his cognitive structure. At the group level, when one or more learners disagree with existing information in the shared workspace then this also provides a conflict. In this case, the learners in the group might negotiate together to decide how both perspectives can be represented in the shared workspace. They will need to identify points of conflict, and then to reconcile the conflict by discussing with one another.

The arrows between the shared artifact and the individual memories represent information flow within the collaborative learning process. Information flows in two directions. Firstly, individuals use the shared artifact to represent the knowledge that they have, and want to communicate to the rest of the group. As a result of this representation activity information is able to flow from the individual to the shared artifact. Secondly, each learner explores the information that has been represented, by themselves and by others, in the shared artifact. As a result, information flows from the shared artifact to each individual.

Through the four activities of construction, representation, exploration and negotiation, knowledge in the individuals' minds and the information which is held in the shared artifact evolve together throughout the collaborative learning process.

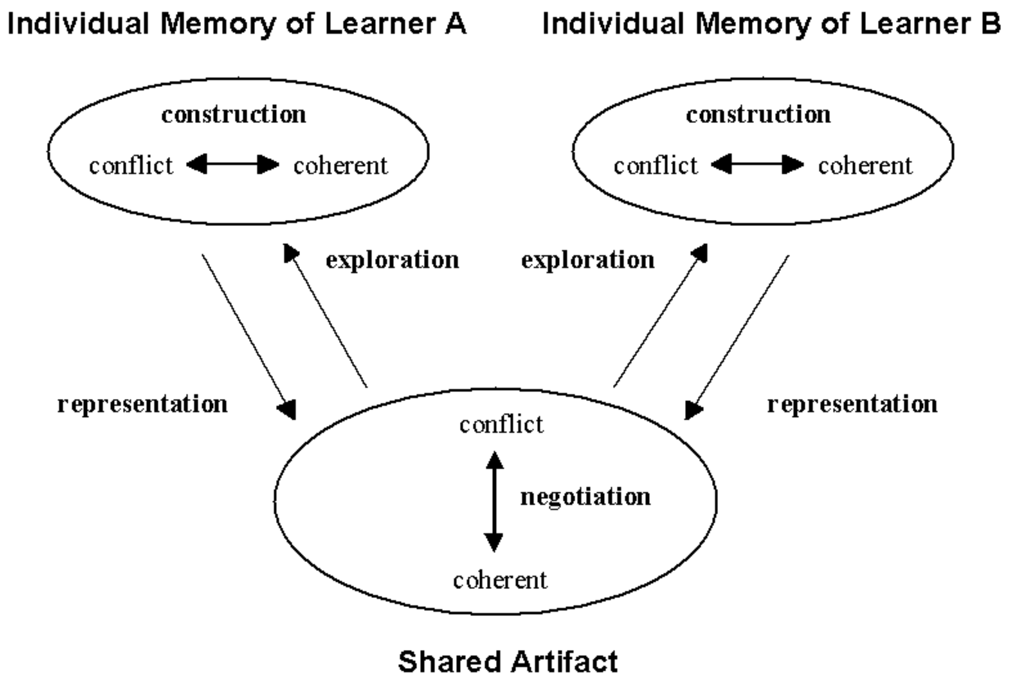


Figure 4.6: Conceptual Model of Collaborative Learning Between Two Learners

4.4.2 Requirements

The shared artifact is an important part of the learning context in collaborative learning processes. In conventional learning environments, the shared artifact is usually carried on paper or blackboards and is recorded in the form of text, tables, or diagrams, etc. The learners have rich communication channels with which to exchange their ideas and negotiate knowledge face-to-face. However, in virtual learning environments, because of the distribution of learners in time and space, and the limitation of the communication channels, exchange of knowledge mainly relies on the shared artifact being represented in an electronic form. An essential requirement to support exchange of knowledge in such virtual environments is to provide a shared workspace. Within the shared workspace learners can access and construct a shared artifact. However, if the collection of information is ill-organized, then this makes it difficult for learners to represent their knowledge, and to negotiate and explore the information in the shared artifact.

It has been suggested that graphical knowledge representation methods can be used to organize the information contained in shared artifacts [Suthers99a]. Such methods should help the learners to clarify thinking, to represent and reflect on their knowledge, to integrate new knowledge, to identify misconception, to detect points at which their individual knowledge structures are in conflict, and thereby to pursue common understanding and to build a coherent representation of their common knowledge. We suggest that a method, which gives more explicit support for the process of representation, negotiation, and exploration of information, will aid collaborative learning.

The construction activity cannot be directly facilitated by the system, but when we support the other processes effectively, then the construction activity will benefit in turn. We claim that a graphical knowledge representation method, which gives effective support for representation, negotiation and exploration, will indirectly improve the way in which learners individually construct their own knowledge. Representation, negotiation and exploration are directly associated with the shared artifact, which serves as a medium for communication and cooperation and provides a group memory.

We now describe in more detail what is required from the graphical knowledge representation method in order to support above three activities. Among the requirements described below, some of them can be directly derived from the abstract model of collaborative learning. Others have been identified in the chapter 3 and are repeated here.

4.4.2.1 Support for Representation

To support the representation process, the following factors should be considered:

Types of knowledge. When the learning content is ill-structured, as for example in PBL, the learners should be able to express their knowledge according to the activities which they are carrying out. More specifically, the individual needs some way to express why they are including a new piece of information, and what role this information plays within the overall shared knowledge structure. It should be possible to clarify the contribution that a particular piece of information makes to the overall task. This could be achieved by allowing the learner to label their contributions.

Expressing relationships. In order to support learners to describe the role that each piece of information plays in the whole, it is needed to allow them to build and label relationships between their contributions.

Integrating associated information. Learners should be able to show how additional information, which perhaps describes the wider context or gives more detail, relates to the information displayed on the shared artifact.

Expressing perspective. Once two or more learners have created their contributions to the shared artifact, each individual learner should be able to indicate the extent to which they agree or disagree with the contributions made by others.

4.4.2.2 Support for Exploration

To support the exploration process, the following factors should be considered:

Providing an overview and point of access. Learners should be supported to have an overall picture of the shared information that has been contributed by the group members. In our view, this overview should also provide the access point from which learners can reach all parts of the information carried in the shared artifact.

Supporting search. Learners should be supported to search the information in the shared artifact. For example they might search for the most recently contributed information chunks, or for certain types of knowledge. In PBL, for example, learners might want to view their learning goals in isolation, or simply to view problems and their solutions. All other information could then be hidden. In this way the group could focus on particular aspects of their task without interference from the wider context.

Detecting conflict. The system should provide a visual indication of the points at which conflict occurs. When exploring the information space, learners can then easily see points they should further discuss with other learners, or gather more information.

Reusability. The knowledge in the shared artifact should be reusable for the learners themselves at a later point in time, and could also serve as a source of information for other interested learners who may not have originally contributed to the shared artifact. The information in the shared artifact should be stored persistently, and consideration should be given to the level of accessibility given to other interested parties.

4.4.2.3 Support for Negotiation

To support the negotiation process, the following factors should be considered:

Automatically initiating negotiation process. In addition to measure conflict, the system should be able to initiate negotiation processes automatically according to the result of the measurement.

Conflict Resolution. Depending on the nature of the conflict, the system should be able to provide support to guide the learners to resolve their different opinions.

4.4.3 An Activity-oriented, Graphical Knowledge Representation Method

In the paragraphs below, we describe the conceptual design of our graphical knowledge representation method, which addresses above requirements.

The method that we have adopted is based on the idea of an *Activity Space* in which knowledge is represented as a network of nodes and links [Streitz89, Streitz92, Haake92]. The concept of the activity space was used by Streitz et al. to support the task of authoring. Four aspects of the authoring task were identified: planning (the planning space), collecting content (the content space), elaborating arguments (the argumentation space), and the creation of a reader-oriented and coherent final document (the rhetorical space). Four separate activity spaces were used to enable authors to visually represent their knowledge in the form of specialized hyperdocuments. They were specialized in the sense that in any one activity space only a restricted set of node and link types could be used. These node and link types were designed to match the characteristics of the activity. In this way authors were forced to structure their hyperdocument in a pre-specified way. The authors performed their authoring tasks by travelling through the four activity spaces.

We have applied the activity space concept to collaborative learning situations. We are particularly concerned to support situations in which the content domain is ill-structured. This is typically the case when learning topics arise on the basis of need, rather than within more traditional subject-based contexts. In such more traditional learning situations, the content of the knowledge domain itself can be used to structure knowledge, and therefore naturally provides the representational means to present this knowledge piece by piece to the learner. In contrast, the knowledge handled in learner-centered approaches, such as the problem-based learning process, is ill-structured. We therefore cannot use the content as the basis on which to organize the shared knowledge. We choose instead to organize the shared knowledge on the basis of learning actions, such as exploring problems, identifying learning issues, setting learning goals, planning, collecting learning resources, applying knowledge, or negotiating shared knowledge. For each type of learning action, we propose that specific node types and link types can be designed, which will appropriately restrict the structure of the hyperdocument that the learners can build. Thus, this method is called an *activity-oriented knowledge representation method* [Miao00a]. It meets the requirements for the representation of ill-structured knowledge.

In the activity-oriented knowledge representation method, the first step is to identify what learning actions we should support. The second step is to define which knowledge types are handled in these actions. These knowledge types are used to indicate the purpose and intention of a created information unit. The third step is to define the relationship between the knowledge types. These relationships are indicated by the link types. The final step is to define the task-specific operations that can be performed on the information units represented as typed nodes and typed links. The identified node types and link types form a knowledge representation schema. According to this schema, learners can represent their ideas with indicating the types of the ideas. They can also indicate the types of relationships between the ideas. In addition, they can provide more detailed information for explaining their ideas. The typed ideas and typed relationships form a net of information items.

As illustrated in Figure 4.7, a net of information items consists of typed nodes and typed links. A typed node may refer to a document that provides more detailed information for the node. All nets in a virtual institute form a net base that is the shared artifact of the virtual institute.

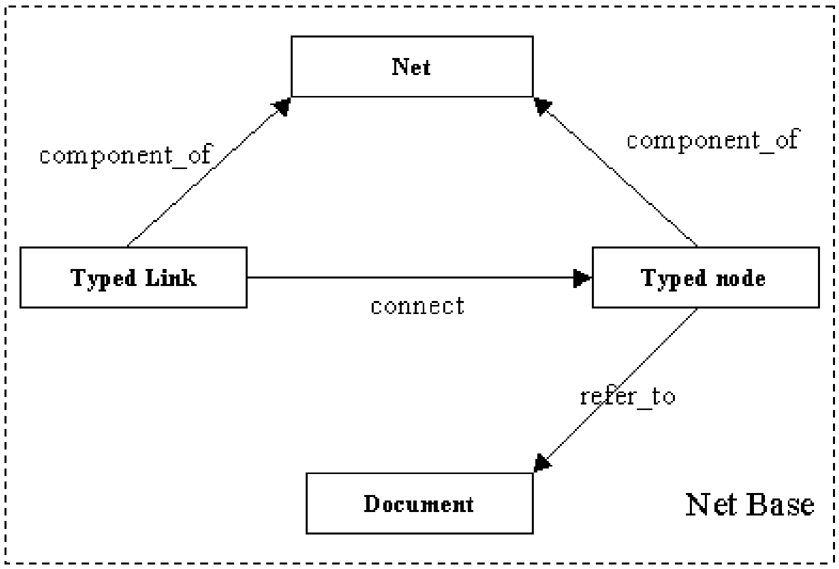


Figure 4.7: Conceptual Architecture of the Shared Artifact

The paragraphs below formally specify the net base.

Definition (Node Type): A *node type* is used to indicate a specific type of information unit.

NodeType = = STRING

Definition (Link Type): A *link type* is defined as a pair. The first element of the pair is a string that serves as a label. The second element is a pair that indicates between which node types the link can connect.

$\text{LinkType} == \text{STRING} \leftrightarrow (\text{NodeType} \leftrightarrow \text{NodeType})$

Definition (Net Schema): A *net schema* is defined by a name, a set of node types and a set of link types. Within a net schema, every node type has unique label. Every link type has a label and can only connect node types present in the NetSchema. A net schema specifies all elements of a knowledge representation.

NetSchema

name : STRING

nodeTypes : $\mathbb{P}$ NodeType

linkTypes : $\mathbb{P}$ LinkType

$\forall\, s, d : \text{LinkType} \mid s \in \text{NodeType} \wedge d \in \text{NodeType} \bullet s \neq d$

$\forall\, l : \text{LinkType} \mid l \in \text{linkTypes} \bullet$

$\quad \text{first}\ (\text{second}\ l) \in \text{nodeTypes} \wedge \text{second}\ (\text{second}\ l) \in \text{nodeTypes}$

In order to support knowledge representation in a virtual institute, different net schemata can be defined and used for facilitating different learning activities. When a net schema is chosen, information units and their relations should be organized according to the framework defined by the selected net schema. As mentioned above, these information units and their relations are represented as typed nodes and typed links. The node types and link types are used as an attribute to define data type: TypedNode and TypedLink.

Definition (Typed Node): A *typed node* represents an information unit that is categorized by its node type. A typed node has a statement attribute to represent knowledge. The information about the creator of the node is recorded.

```
┌─ TypedNode ──────────────────────────────
│ statement : STRING
│ nodeType : NodeType
│ owner : Actor
└──────────────────────────────────────────
```

Definition (Typed Link): A *typed link* is a data type that represents a relation between two information units. A typed link is categorized by its link type. A typed link records information about two typed nodes between which a link is connected. The information about the creator of the link is recorded as well.

```
┌─ TypedLink ──────────────────────────────
│ linkType : LinkType
│ sourceNode : TypedNode
│ destinationNode : TypedNode
│ owner : Actor
└──────────────────────────────────────────
```

It is important to note that typed nodes and typed links may have distinct attributes. The specific operations on these typed nodes and typed links will be defined based on these specific attributes. For example, in problem based learning, the typed node "learning issue" has additional attributes such as "who knows", "who don't know", "who needs to know", and so on. Specific operations are defined to change the values of these attributes. However, to simplify discussion, we don't specify such details here. Only for some typed nodes, we will show a more detailed specification.

Definition (Net): A *net* can contain typed nodes and typed links to show the intention of the authors when creating information. However, it can not contain basic information elements such as text and image. Each net has an "net type" attribute that is specified by a net schema.

```
┌─ Net ────────────────────────────────────
│ title : STRING
│ topic : STRING
│ owner : Actor
│ netType : NetSchema
└──────────────────────────────────────────
```

Definition (Net Base): A *net base* consists of a set of nets, a set of typed nodes, and a set of typed links. Each typed node or typed link belongs to a certain net. Some typed nodes refer to documents. Typed nodes can connect to documents via hyperlinks. If a typed link connects two typed nodes, all of them should be in the same net, and the type of the typed nodes should be consistent with the definition of the typed link.

```
┌─ NetBase ─────────────────────────────────────────────────
│ nets : ℙ Net
│ typedNodes : TypedNode → Net
│ typedLinks : TypedLink → Net
│ referToDoc : TypedNode ⇸ Document
├───────────────────────────────────────
│ ran typedNodes ⊆ nets ∧ ran typedLinks ⊆ nets
│ ∀ s, d : TypedNode; l : TypedLink | l.sourceNode = s ∧ l.destinationNode = d •
│       (typedNodes s) = (typedNodes d) = (typedLinks l) ∧
│       s.nodeType = first (second l.linkType) ∧
│       d.nodeType = second (second l.linkType)
└───────────────────────────────────────────────────────────
```

So far, some data types are specified. For a complete specification of the activity-oriented approach to knowledge representation, the operations are still unspecified. As mentioned above, the specification of operations depends on a specific activity. In the next subsection, we will discuss how this approach is applied to support problem based learning activities.

4.4.4 PBL-net

In this subsection, we discuss how the activity-oriented knowledge representation method is applied to support problem-based learning activities. First of all we describe the *PBL schema* and how it is defined. The schema then provides the framework according to which learners will collaborate with each other in order to create their own PBL-nets. Then we will explain how the PBL-net supports representation, exploration and negotiation in PBL processes.

4.4.4.1 PBL-net Schema

When applying the activity-oriented knowledge representation method in supporting PBL, the first step is to identify what types of nodes and links are used in the problem based learning process. The result is a PBL-specific net schema, called PBL-net schema. The types of nodes and links that we define are based on the various tasks that make up the PBL process. One can refer to the scenario described in chapter 2. There are various descriptions of the tasks involved in PBL, for example the eight tasks detailed in [Course Material]. These are (1) explore the problem, (2) identify what learners know, (3) identify what learners do not know, (4) identify the goals and make action plan, (5) collect information, (6) discuss information collected, (7) apply knowledge to the problem, (8) review the process.

For the task of exploring the problem, the learner must define problems. Therefore we define a ‘problem’ node type. A problem can be decomposed into sub-problems, using an ‘is_a_sub_of_problem’ link between the main problem and its sub-parts. A ‘source’ node type is defined so that background material to the problem can also be represented. An information unit with a ‘source’ node type can inform about an information unit with a ‘problem’ node type. A ‘inform_about’ link type is introduced in the PBL-net schema. Similarly, learners will need to identify what aspects of the problem they need to learn about. In the schema, a ‘issue’ node type is provided as a means to allow them to indicate this.

When performing tasks (2), (3), and (4), learners declare what learning issues they know or don’t know. They identify what knowledge is still missing. They decide who will be responsible for collecting information about what issues. They also have to identify the relations between learning issues. Link types to express relationships among the learning issues are ‘is_prior_to’, ‘is_sub_of_, and ‘is_a_ prerequisite_for’. This information will be used as the basis on which to define learning goals and to make a learning plan. The process to make an action plan and to execute the defined action plan will be discussed in the section 4.6.

In order to decide what information is needed and to integrate the collected information, the ‘resource’ node type and the ‘concern’ link type are defined. The ‘concern’ link type is used to indicate to which learning issue an information unit is related.

Then, learners will debrief information and discuss by abstracting what they learnt from the collected information. The ‘principle’ and ‘evidence’ node types serve to represent the acquired knowledge. Meanwhile, it is needed to indicate from which resource the new knowledge was derived by using the ‘derived_from’ link type.

To support task (7), the ‘hypothesis’ and ‘solution’ node types are defined. There are three possible relations between ‘hypothesis’ node type: ‘is_similar_to’, ‘is_contrary_to’, and ‘is_a_prerequisite_for’. The ‘based_on’ relation between ‘solution’ and ‘hypothesis’ is defined to indicate that a solution is generated based on a hypothesis. A hypothesis is generated to suggest which problem or sub-problem can be represented by using ‘suggest’ link type. A solution is generated to solve a problem or sub-problem, this can be represented by using the ‘solve’ link type. Learners can use ‘support’ or ‘counter’ link type to represent the relations among ‘principle’, ‘evidence’, ‘hypothesis’, and ‘solution’ in negotiation processes.

To support task (8), the ‘comment’ node type is defined. The relation between a comment and an information unit being commented is indicated by using the ‘comment_on’ link type.

It is important to note that information units of ‘comment’, ‘hint’, and ‘question’ node type can be created whenever it is needed. Participants of a PBL activity can connect these types of information units to any type of information unit by using links of type ‘comment_on’, ‘about’, and ‘about’, respectively. An information unit with node type ‘answer’ can be created to answer a ‘question’ node with a ‘answer’ type link.

For each task in PBL, we have chosen appropriate node types and link types. All node types and link types for each task are represented in the PBL net schema. In the virtual institute, a PBL-net schema is initialized according to the specification in this thesis. Therefore, it is defined as a variable with net schema type.

Definition (PBL-net Schema): A *PBL-net schema* is specific net schema for structure knowledge for PBL. Its value of name attribute is ‘PBL-net schema’. The node types and link types are those identified above. The current version of the PBL-net schema is defined as following.

```
pblNetSchema = = ( ‘PBL-Net schema’,
                  {‘source’, ‘problem’, ‘issue’, ‘resource’, ‘evidence’, ‘principle’,
                  ‘hypothesis’, ‘solution’, ‘comment’, ‘hint’, ‘question’, ‘answer’},
                  { (‘inform_about’ ↦ ‘source’ ↦ ‘problem’),
                    (‘is_a_sub_problem_of’ ↦ ‘problem’ ↦ ‘problem’),
                    (‘is_a_sub_issue’ ↦ ‘issue’ ↦ ‘issue’),
                    (‘is_prior_to’ ↦ ‘issue’ ↦ ‘issue’),
                    (‘is_a_prerequisite_for’ ↦ ‘issue’ ↦ ‘issue’),
                    (‘repect_to’ ↦ ‘issue’ ↦ ‘problem’),
                    (‘concern’ ↦ ‘resource’ ↦ ‘issue’),
                    (‘derive_from’ ↦ principle’ ↦ ‘resource’),
                    (‘derive_from’ ↦ ‘evidence’ ↦ ‘resource’),
                    (‘support’ ↦ ‘principle’ ↦ ‘hypothesis’),
                    (‘support’ ↦ ‘principle’ ↦ ‘solution’),
                    (‘support’ ↦ ‘evidence’ ↦ ‘hypothesis’),
                    (‘support’ ↦ ‘evidence’ ↦ ‘solution’),
                    (‘counter’ ↦ ‘principle’ ↦ ‘hypothesis’),
                    (‘counter’ ↦ ‘principle’ ↦ ‘solution’),
                    (‘counter’ ↦ ‘evidence’ ↦ ‘hypothesis’),
                    (‘counter’ ↦ ‘evidence’ ↦ ‘solution’),
                    (‘solve’ ↦ ‘solution’ ↦ ‘problem’),
                    (‘based_on’ ↦ ‘solution’ ↦ ‘hypothesis’),
                    (‘is_similar_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’),
                    (‘is_contrary_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’),
                    (‘is_a_prerequisite_for’ ↦ ‘hypothesis’ ↦ ‘hypothesis’),
                    (‘to’ ↦ ‘answer’ ↦ ‘question’),
                    (‘comment_on’ ↦ ‘comment’ ↦ ‘source’),
                    (‘comment_on’ ↦ ‘comment’ ↦ ‘problem’),
                    (‘comment_on’ ↦ ‘comment’ ↦ ‘issue’),
                    (‘comment_on’ ↦ ‘comment’ ↦ ‘resource’),
                    (‘comment_on’ ↦ ‘comment’ ↦ ‘evidence’),
```

(‘comment_on’ ↦ ‘comment’ ↦ ‘principle’),
(‘comment_on’ ↦ ‘comment’ ↦ ‘hypothesis’),
(‘comment_on’ ↦ ‘comment’ ↦ ‘solution’),
(‘comment_on’ ↦ ‘comment’ ↦ ‘comment’),
(‘comment_on’ ↦ ‘comment’ ↦ ‘hint’),
(‘comment_on’ ↦ ‘comment’ ↦ ‘question’),
(‘comment_on’ ↦ ‘comment’ ↦ ‘answer’),
(‘about’ ↦ ‘hint’ ↦ ‘source’),
(‘about’ ↦ ‘hint’ ↦ ‘problem’),
(‘about’ ↦ ‘hint’ ↦ ‘issue’),
(‘about’ ↦ ‘hint’ ↦ ‘resource’),
(‘about’ ↦ ‘hint’ ↦ ‘evidence’),
(‘about’ ↦ ‘hint’ ↦ ‘principle’),
(‘about’ ↦ ‘hint’ ↦ ‘hypothesis’),
(‘about’ ↦ ‘hint’ ↦ ‘solution’),
(‘about’ ↦ ‘hint’ ↦ ‘comment’),
(‘about’ ↦ ‘hint’ ↦ ‘hint’),
(‘about’ ↦ ‘hint’ ↦ ‘question’),
(‘about’ ↦ ‘hint’ ↦ ‘answer’),
(‘about’ ↦ ‘question’ ↦ ‘source’),
(‘about’ ↦ ‘question’ ↦ ‘problem’),
(‘about’ ↦ ‘question’ ↦ ‘issue’),
(‘about’ ↦ ‘question’ ↦ ‘resource’),
(‘about’ ↦ ‘question’ ↦ ‘evidence’),
(‘about’ ↦ ‘question’ ↦ ‘principle’),
(‘about’ ↦ ‘question’ ↦ ‘hypothesis’),
(‘about’ ↦ ‘question’ ↦ ‘solution’),
(‘about’ ↦ ‘question’ ↦ ‘comment’),
(‘about’ ↦ ‘question’ ↦ ‘hint’),
(‘about’ ↦ ‘question’ ↦ ‘question’),
(‘about’ ↦ ‘question’ ↦ ‘answer’) }
)

A diagram representation of the PBL-net schema is shown in Figure 4.8. In this figure, the squares represent node types, while the arrows represent link types. By using the schema, learners can be supported to create task-specific knowledge representations as a knowledge representation language.

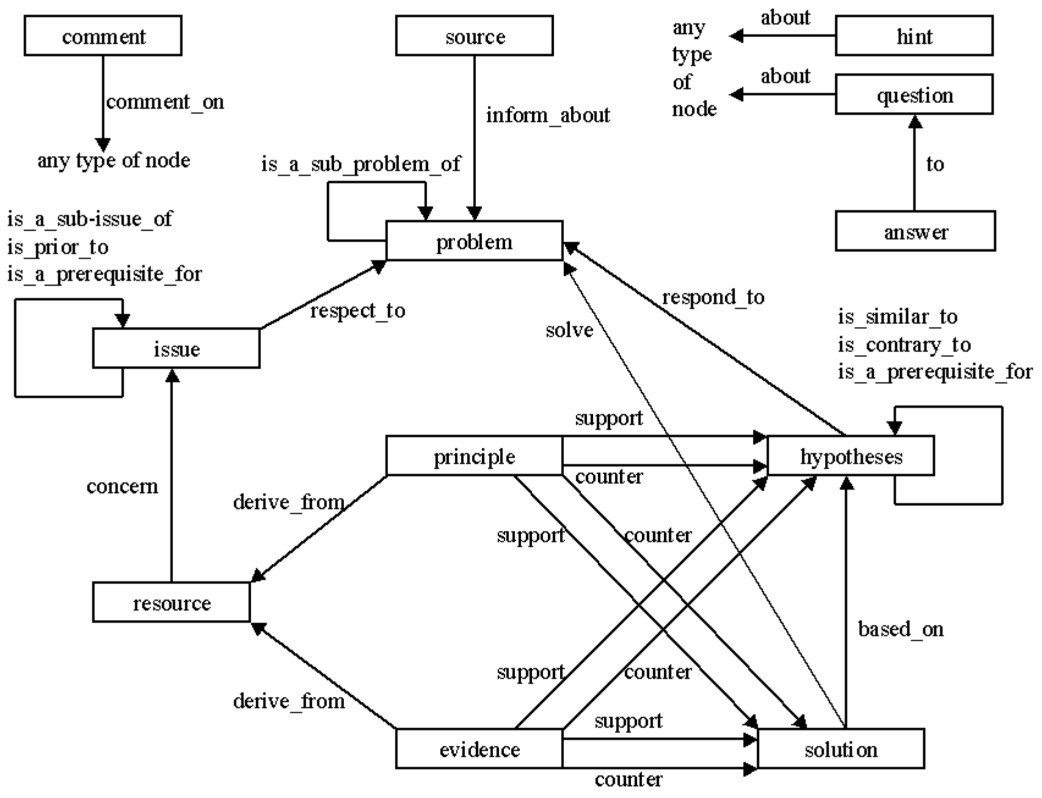


Figure 4.8: PBL-Net Schema

4.4.4.2 PBL-net and its Operations

After defining the PBL-net schema, we can define the PBL-net. Then, operations on a PBL-net can be specified.

Definition (PBL-net): A *PBL-net* is a net whose value of the net type attribute is the pblNetSchema defined above.

Definition (Declaration): A *declaration* is a data type that is used to represent the perspective of learners to a certain discussion point. The declarer attribute is used to indicate who declares. The confidence attribute is used to indicate the degree of confidence the declarer has on this perspective. This issue will be discussed later in more detail.

Perspective = = $\mathbb{R}$ [-1, 1]

Confidence = = $\mathbb{R}$ [0, 1]

```
┌─ Declaration ──────────────────────────────
│ declarer : Actor
│ perspective : Perspective
│ confidence : Confidence
└────────────────────────────────────────────
```

Definition (PBL-net Base): A *PBL-net base* is a part of net base of a virtual institute. In this thesis, we focus on support for PBL activities. Therefore, we only discuss PBL-nets.

```
┌─ PBLNetBase ──────────────────────────────────────────
│ pblNets : ℙ Net
│ typedNodes : TypedNode → Net
│ typedLinks : TypedLink → Net
│ referToDoc : TypedNode → Document
│ declareNode : Declaration → TypedNode
│ declareLink : Declaration → TypedLink
├──────────────────────────────
│ ∀ n : Net | n ∈ pblNets • n.netType = pblNetSchema
│
│ ran typedNodes ⊆ pblNets ∧ ran typedLinks ⊆ pblNets
│ ∀ s, d : TypedNode; l : TypedLink | l.sourceNode = s ∧ l.destinationNode = d •
│       (typedNodes s) = (typedNodes d) = (typedLinks l) ∧
│       s.nodeType = first (second l.linkType) ∧
│       d.nodeType = second (second l.linkType)
└───────────────────────────────────────────────────────
```

In the paragraphs below, we define the operations on the PBL-net base

4.4.4.2.1 Representation of Shared Knowledge

Using the PBL-net schema that we have just defined, participants of a PBL activity are able to create, delete or modify typed nodes and typed links in order to form their PBL-net. First of all, actors can create a new PBL-net. The specification of this simple operation is ignored. We focus on how an actor creates a typed node or a typed link in a given PBL-net.

To create a typed node in a PBL-net, an actor chooses a node type (e.g. problem, issue, or source, etc) and makes a *statement* about that typed node, which serves to describe the content of that node or publish a point of view to others.

```
┌─ CreateTypedNodeOK ───────────────────────────────────
│ Δ PBLNetBase
│ a? : Actor
│ aNet? : Net
│ statement? : STRING
│ type? : NodeType
├──────────────────────────────
│ let aNode = = ( μ TypedNode | statement = statement? ∧
│                               nodeType = type? ∧
│                               state = proposed ∧
│                               owner = a? ) •
│
```

```
    typedNodes' = typedNodes ∪ { aNode ↦ aNet? }

pblNets' = pblNets
typedLinks' = typedLinks
referToDoc' = referToDoc
declareNode' = declareNode
declareLink' = declareLink
```

To create a typed link on a PBL-net, an actor has to choose a link type and connect two typed nodes with correct types.

```
── CreateTypedLinkOK ──────────────────────────
Δ PBLNetBase
a? : Actor
aNet? : Net
s? : TypedNode
d? : TypedNode
label? : STRING
────────────────────────────
(s? ↦ aNet? ) ∈ typedNodes ∧ (d? ↦ aNet? ) ∈ typedNodes ∧
(label? ↦ s?.nodeType ↦ d?.nodeType) ∈ aNet?.netType.linkTypes

let aLink = = ( μ TypedLink | linkType = (label? ↦ s?.nodeType ↦ d?.nodeType)∧
                              sourceNode = s? ∧
                              destinationNode = d? ∧
                              owner = a? ) •
    typedLinks' = typedLinks ∪ { aLink ↦ aNet? }

pblNets' = pblNets
typedNodes' = typedNodes
referToDoc' = referToDoc
declareNode' = declareNode
declareLink' = declareLink
───────────────────────────────────────────────
```

A typed node can be connected to a normal document as its content. There are two possibilities to connect with documents. One way is to create a content document of the typed node upon the node's creation. Another way is to connect a typed node to an existing document.

```
── CreateContentDocumentForNodeOK ─────────────
Δ PBLNetBase
a? : Actor
n? : TypedNode
topic? : STRING
────────────────────────────
 newDoc = = ( μ Document | title = n?.statement
```

topic = topic? $\wedge$
owner = a? $\wedge$
text = $\emptyset$ $\wedge$
tables = $\emptyset$ $\wedge$
images = $\emptyset$)

documents' = documents $\cup$ { newDoc }
documentRefs' = documentRefs
referTo' = referTo

pblNets' = pblNets
typedNodes' = typedNodes
typedLinks' = typedLinks

referToDoc' = referToDoc $\cup$ { n? $\mapsto$ newDoc }
declareNode' = declareNode
declareLink' = declareLink

ConnectNodeToDocumentOK

Δ PBLNetBase
node? : TypedNode
doc? : Document

doc? $\in$ documents $\wedge$ node? $\in$ **dom** typedNodes

referToDoc' = referToDoc $\cup$ { node? $\mapsto$ doc? }
pblNets' = pblNets
typedNodes' = typedNodes
typedLinks' = typedLinks
declareNode' = declareNode
declareLink' = declareLink

The operations to remove typed node or typed link are specified as follows.

RemoveTypedNodeFromNetOK

Δ PBLNetBase
n? : TypedNode
aNet? : Net

(n? $\mapsto$ aNet?) $\in$ typedNodes $\wedge$ n? $\notin$ **dom** referToDoc
pblNets' = pblNets
typedNodes' = typedNodes \ {n? $\mapsto$ aNet? }
typedLinks' = typedLinks \ { $\forall$ l: TypedLink |
    l.sourceNode = n? $\vee$ l.destinationNode = n? $\bullet$ l $\mapsto$ aNet? }
referToDoc' = referToDoc
declareNode' = declareNode
declareLink' = declareLink

RemoveTypedLinkFromNetOK

Δ PBLNetBase
link? : TypedLink
aNet? : Net

{link? $\mapsto$ aNet? }$\in$ typedLinks
pblNets’ = pblNets
typedNodes’ = typedNodes

typedLinks’ = typedLinks \ { link? $\mapsto$ aNet? }
referToDoc’ = referToDoc
declareNode’ = declareNode
declareLink’ = declareLink

A PBL-net provides a means that all involved actors can contribute to by creating typed nodes and links in the shared PBL-net. However, each contribution represents a personal perspective. Other actors may have different points of views. Different personal perspectives drive collaboration. Most collaborative learning support systems provide mechanisms for learners to represent different perspectives. Systems like CSILE [Scardamalia94] and CaMILE [Soloway94] support personal perspectives by different representations of the same information. WebGuide [Stahl99] supports to represent different perspectives on the same information item by indicating the status of the information item (e.g., personal perspective, team perspective, and negotiation perspective). In WebGuide, the term “perspective” is defined as a particular, restricted segment of an information repository that is being considered, stored, categorized, and annotated. Pfister et al. [Pfister99] defined perspective based on a set of facts pertaining to a topic. A perspective is represented as a possible combination of known facts. In this thesis, a perspective is defined based on a statement that each typed node contains. A perspective is represented as a mapping function from the statement or a relation between two statements to a value that ranges from –1 to 1. Each learner can have personal perspectives on the same statement. In PBL, the knowledge items represented in terms of nodes and links normally are not an elementary assertion that can be simply judged as true or false. Example statements may be “the pollution makes the frog deformed” or “the insect topic is related to the deformed fog problem.” If an actor fully believes a statement, the perspective of this actor on the statement is assigned a 1. If an actor fully does not believe a statement, the perspective of this actor on the statement is assigned a -1. A partially belief or partially unbelief is assigned a value in between –1 and 1. Furthermore, because members of a PBL group may have different background and expertise, their values of confidence in their perspective may vary, no matter whether they totally or partially (don’t) believe a statement. The operation to profess an actor’s perspective on a typed node is defined as follows.

DeclarePerspectiveForNodeOK

Δ PBLNetBase
Ξ Editors
Ξ HyperDocument
a? : Actor

node? : TypedNode
perspective? : Perspective
confidence? : Confidence

let aDeclaration = = (μ Declaration | declarer = a? $\wedge$
perspective = perspective? $\wedge$
confidence = confidence?) $\bullet$

declareNode' = declareNode $\cup$ { aDeclaration $\mapsto$ node?}

pblNets' = pblNets
typedNodes' = typedNodes
typedLinks' = typedLinks
referToDoc' = referToDoc
declareLink' = declareLink

In the same way, the operation to profess an actor's perspective on a typed link can be defined. After learning or discussion, an actor may change his perspective or confidence. The specifications for these two operations are ignored.

4.4.4.2.2 Exploration of Shared Knowledge

Learners are supported to explore the information contained in the shared artifact in a number of ways. Each learner uses the PBL net as an access point from which to retrieve information from the hyperdocument. The learner is able to retrieve hyperdocuments by following hyperlinks.

MoveToDocumentFromNodeOK
Δ PBLNetBase
Δ Editors
Ξ HyperDocument
a? : Actor
editor? : DocumentEditor
node? : TypedNode

{editor?} $\lhd$ usedBy $\rhd$ {a?} $\neq \varnothing$ $\wedge$ node? $\in$ **dom** referToDoc $\wedge$
typedNodes node? = (currentDoc editor?))

currentDoc' = currentDoc $\oplus$ { editor? $\mapsto$ referToDoc node? }
storedIn' = storedIn $\cup$ { currentDoc editor? $\mapsto$
first (bookshelfLocation $\rhd$ {actorLocation a?})}
editor?.history = editor?.history $\frown$ < referToDoc node?>

pblNets' = pblNets
typedLinks' = typedLinks
referToDoc' = referToDoc
declareNode' = declareNode

declareLink’ = declareLink

In addition, learners can be supported by the system to search for specific types of information. For example, the learner requests to view only problem nodes, or only learning issue nodes. Perhaps they could request to see problems and their associated solutions.

ShowProblemAndSolutionOK

Ξ PBLNetBase
aNet? : Net
nodes! : $\mathbb{P}$ TypedNode
links! : $\mathbb{P}$ TypedLink

nodes! = { n : TypedNode | n $\in$ typedNodes $\rhd$ {aNet?} $\wedge$
  (n.nodeType = ‘problem’ $\vee$ n.nodeType = ‘solution’) }
links! = { l : TypedLink | l $\in$ typedLinks $\rhd$ {aNet?} $\wedge$
  l.sourceNode.nodeType = ‘problem’ $\wedge$
  l.destinationNode.nodeType = ‘solution’}

A PBL-net allows learners to know others’ perspective and confidence on a given typed node. The operation to access information about others’ declaration on a given typed link can be defined in a similar way.

KnowActorDeclarationForNodeOK

Ξ PBLNetBase
Ξ Editors
Ξ HyperDocument
a? : Actor
node? : TypedNode
p! : Perspective
c! : Confidence

$\exists$! d : Declaration | d $\in$ **dom** declareNode $\bullet$ d.declarer = a?

p! = (λ PBLNetBase; node? : TypedNode $\bullet$
  { Declaration | $\boldsymbol{\theta}$ Declaration $\in$ (**dom** declareNode) $\wedge$
    d.declarer = a? $\bullet$ perspective })
c! = (λ PBLNetBase; node? : TypedNode $\bullet$
  { Declaration | $\boldsymbol{\theta}$ Declaration $\in$ (**dom** declareNode) $\wedge$
    d.declarer = a? $\bullet$ confidence })

Definition (Mutual Understanding): A *mutual understanding* means that every member knows anyone else’s perspective and confidence, even if they have different perspectives. The pre-condition to get a mutual understanding is that all members of the group declared their perspectives and the declaration information is accessible to all members.

While exploring a shared workspace, learners need know the point at which there is conflicting knowledge. The learners may therefore pay special attention to this point. In WebGuide [Stahl99], the *negotiation perspective* indicates this perspective hasn't be accepted by the group. It needs to be discussed continually. When all members accept the *negotiated perspective*, it becomes a *team perspective*. However, two states (the *negotiation perspective* and the *team perspective*) can not capture the degree of conflict. Pfister et al. [Pfister99] uses quantitative measure to indicate different perspectives. They developed the concept of *degreement*. The degreement between two people on a topic, which consists of a set of facts, is measured by the difference between the ratio of commonly known facts and the ratio of facts that only one of them knows. The limitation of this approach is that the degreement can only be measured when a topic can be decomposed into multiple facts. In PBL, an opinion such as a hypothesis and a solution is not necessarily and even can not be decomposed into multiple facts.

The approach proposed in this thesis combines the advantages of these two approaches and overcomes their limitations. We take an example to explain this approach. We assume that n students ($\boldsymbol{S}_1$, .. $\boldsymbol{S}_n$) learn a database course together through solving a real problem – developing a course management database system. A statement (use ORACLE) as a solution is proposed to solve a sub-problem (which DBMS do we use?). If all of them understand this statement and have consensus, it is not a point that needs to be discussed. Whenever a student has a different perspective to this statement, the student will represent a different perspective on this statement by using a declaration operation described above. This statement becomes a conflict point that forces the group to discuss. Perspectives are distributed in the range [-1, 1]. In order to measure the conflict, some concepts are defined.

Definition (Group Perspective): A *group perspective* is measured as an average sum of all members' perspectives.

$$\text{GroupPerspective} == \left(\sum_{i=1}^{n} \boldsymbol{S}_i.\mathit{perspective}\right) \backslash \text{ n}$$

Definition (Conflict Degree): A *conflict degree* is measured as an average sum of all personal perspectives' weighted deviation from the group perspective.

$$\text{ConflictDegree} == \left(\sum_{i=1}^{n} \mid \mathit{GroupPerspective} - \boldsymbol{S}_i.\mathit{perspective} \mid * \boldsymbol{S}_i.\mathit{confifence}\right) \backslash \text{ n}$$

If the conflict degree is larger that 0.5, this conflict is regarded as a *strong conflict*. Otherwise it is a *weak conflict*. This initial value of 0.5 should be adjusted after experiental evaluation.

4.4.4.2.3 Negotiation of Shared Knowledge

When a conflict has occurred, the conflict will stimulate the learning group to negotiate perspectives and resolve the conflict for building common understanding. Computer-mediated negotiation can be used to merge several perspectives into a common one [Stahl99]. A PBL-net can support negotiation of knowledge in two ways. Firstly, when a conflict degree is larger than a number (e.g., 0.5), the virtual learning environment can automatically initiate a computer-mediated process to support negotiation according to the nature of the conflict. Secondly, the status of typed nodes will influence the negotiation procedure. This topic will be discussed in the next chapter.

4.4.5 Related Work and Discussion

We compare our approach with related work regarding two aspects. Firstly, we compare the activity-oriented, graphical knowledge representation method with existing, alternative graphical knowledge represent methods in the education area. Secondly, we compare the PBL-net with the corresponding features offered by other PBL support systems described in Chapter 3.

4.4.5.1 Alternative Approaches to Visually Represent Structured Knowledge

We first look at existing graphical knowledge representation methods for education. We do not compare with other forms of knowledge representation methods (such as text-based or threaded discussion methods). We adopt a graphical knowledge representation method because of viewing the advantages of this method. As Larkin [Larkin87] pointed out, a diagram conveys a lot of information. Graphical knowledge representation languages can help learners to clarify thinking, to identify misconceptions, to reinforce understanding, and to integrate new knowledge. Similarly, we do not discuss the methods of knowledge representation used in Artificial Intelligence such as predicate logic and frames, since although they have been used to support learning (e.g. in Intelligent Tutoring Systems) they aim at formalizing knowledge for machine reasoning. However, the primary purpose of our graphical knowledge representation language is to help human learners construct shared knowledge and to facilitate interactions among human learners.

We review the state-of-the-art for graphical knowledge representation methods within the following three categories that we ourselves have defined: *content-based* methods, *didactic-oriented* methods and *activity-oriented* methods. We briefly consider each method in terms of the three groups of requirements: representation, exploration and negotiation.

4.4.5.1.1 Content-based knowledge representation methods

These methods, such as concept maps, represent knowledge as a network of nodes and links [Novak84, Jonassen93]. The nodes are used to present concepts or ideas (e.g. human, living thing, food) and they are linked with a content-oriented relation (e.g. is_a, eat) as a formal or semi-formal diagram. This in part satisfies the requirements for the representation of knowledge within a shared artifact. However, the content-based approach is not suitable for representing ill-structured knowledge, because for ill-structured domains also the knowledge structure would be ill-structured. Content-based methods give good support for exploration, since they provide an overview of the learning material, although they have not been used so far to indicate points of conflict between learners. Similarly, content-based methods to date have not been developed in order to address the requirements of negotiation support. In PBL, knowledge related to a real-world problem is ill structured from the perspective of content. Therefore, Content-based knowledge representation methods are not suitable to represent knowledge structure in PBL.

4.4.5.1.2 Didactic-oriented knowledge representation methods

Didactic-oriented representation methods use didactic notions to type the chunks of information within a structured net [Trigg83, Baloian95]. In the didactic net the nodes describe the type of learning material (e.g., concept, definition, explanation, example) while the link types define the type of didactic relation (refined_by, defined_by, explained_by, exemplified_by) between the nodes. This method has been used to produce courseware, and to guide the learner to receive information piece by piece. It satisfies all of the requirements for representation, except that of expressing conflict. In terms of exploration, didactic methods also provide a satisfactory overview of material, and supports reusability by the learners because they make the nature of information chunks and the relations between them explicit and easily understood. However, once more, there was no intention for these nets to be used to give specific support for collaboration and negotiation between students. It is needed to note that a learning environment can be developed as a multi-user system by adapting this approach. The system described in [Baloian95] can support multiple teachers to co-author the courseware collaboratively. When a teacher presents the prepared knowledge, multiple students can synchronously follow the teacher's navigation in the didactic net, exactly as they passively receive knowledge piece by piece in the conventional classrooms. However, in PBL students are actively involved in the construction of shared knowledge collaboratively without any didactic intention.

4.4.5.1.3 Activity-oriented knowledge representation methods

As discussed above, content-based knowledge representation methods and didactic-oriented knowledge representation methods are obviously not suitable for representing knowledge for PBL. An activity-oriented approach to graphical knowledge representation was introduced and extended for supporting PBL. The main goal of activity-oriented methods is not to describe the contents of a topic in its entirety. Rather, a network of labeled nodes and links is used to structure the knowledge in ways that support different activities. According to our definition, some systems can be regarded as having used an activity-oriented approach to represent knowledge, such as gIBIS that was developed to support issue-based argumentation [Conklin87a]. Secondly, SEPIA which was designed to support authoring [Streitz89, Streitz92]. However, neither of these two systems was designed for educational use.

Other examples, which have been used for educational purposes are Belvedere, for example, which supports scientific inquiry [Suthers99a] [Suthers99b], or CLARE [Wan94a] which provides support for understanding papers. CLARE is a text based hypertext system that didn't support graphical knowledge representation. The advantage of the activity-oriented methods is that they are designed to support a specific activity, rather than simply to represent the content of a domain. Because of this they are an ideal means of tackling ill-structured topics. However, all these systems still do not address the question of support for representation of conflict perspective and negotiation.

4.4.5.2 Comparison with PBL Support Systems

In this subsection, we will briefly outline the kind of features offered by other systems designed to support PBL. We used our collaborative learning model as the framework within which to consider these other systems. We now would like to assess the extent to which others existing systems described in chapter 3 can serve to support the PBL learning process under the same conditions. In other words we will assess the extent to which they address support for representation, exploration and negotiation as defined in our model.

Representation. Each of these systems provides typing of information units. This ranges from 3 to 8 different types across the six systems. None of these systems, except for Belvedere, provide link types between the information units as provided by our method. Most of them provide a threaded text outline in which indentations (i.e. sub-headings) are typed. There is no graphical view of the overall structure. Using such a threaded text representation it is harder to identify relationships between the information units. All systems provide support for commenting on information units that can be used for expressing different perspective, but these must be sought and read in detail before the nature of the conflict can be identified.

Exploration. As mentioned, most of these systems provide an overview of all information in the shared artifact, albeit textual. Belvedere provides graphical representation, but Belvedere's Inquiry Diagram only supports inquiry activity. However, all of these systems serve as a pool of information which once entered cannot easily be deleted or rearranged by the learners. Detecting conflict is difficult, since there is no explicit way to indicate the point at which it occurs.

Negotiation. Since conflict perspectives on the same statement cannot be expressed, this cannot serve as a starting point for negotiation. Similarly, there is no special facility by which learners can visually indicate the extent of their knowledge about the contents of a particular information unit. Once again, this can be expressed indirectly by means of textual comments about each unit or by creating separate statements.

4.4.6 Summary

The theoretical basis for this work lies in the constructivist and situated learning paradigms. This led us to identify two key areas in which graphical knowledge representation methods should be developed. We therefore see the following as the two main findings. Firstly, in constructivist the principle of cognitive conflict is central to learning. A conceptual model of collaborative learning is developed that addresses the conflict on individual memory level and on group memory level. Considering the state-of-the-art in terms of graphical knowledge representation methods, we found a marked absence of support for the resolution of such conflict and the support of negotiation during collaborative learning in virtual environments. Secondly, in situated learning theory, activities rather than content are emphasized. By choosing an activity-oriented approach, rather than the content-based or didactic-oriented approaches, we have begun to address the question of how to provide structured shared information spaces, which will be appropriate for ill-structured knowledge domains. We are confident that this new approach to support the representation, exploration and negotiation of shared knowledge can be further developed to provide a significant contribution to cooperative learning in the workplace. As the first application of the activity-oriented graphical knowledge representation approach, a graphical knowledge representation language was developed.

4.5 PBL-protocols: Guiding and Controlling Social Interaction in PBL Processes

As discussed in Chapter 2, problem-based learning is an innovative instruction method and requires learners to actively gather and apply knowledge in order to solve ill-structured real-world problems. Contrary to traditional instructional methods, where the teacher organizes and imparts information to the students, problem-based learning is guided by tutors who take a facilitator role, encouraging students to engage in active and meaningful learning. However, teachers used to teaching through lectures and discussions lack the skills of a facilitator in guiding learners to discover information for themselves. As a facilitator they can give hints, provide resources and ask searching questions, but they must withhold information that they would previously have simply given to the students. Learners are also slow to adjust to the PBL method, and to the change in their role from passively receiving information to actively engaging in a problem-solving process. When teachers and learners are unfamiliar with PBL, they tend to be reluctant to change their traditional roles [Jones94] [Bridges92]. Additional problems arise when the PBL method is applied in virtual learning environments where participants are distributed and weak communication channels make group interactions difficult. It is hard to coordinate operations performed by different people and to make and keep track of progress towards learning goals efficiently. In order to support problem-based learning in virtual learning environments, it is argued that computational mechanisms can be designed to help overcoming the difficulties discussed above. In this section, firstly, a brief introduction to schema theory is given. In the light of schema theory, an approach to model and execute collaboration processes is presented. Secondly, we show how this approach is applied to guide and control problem-based learning processes in virtual learning environments. Thirdly, we compare the proposed approach with other approachs to support cooperative processes and features offered by other PBL support systems discussed in the Chapter 3. Finally, a short summary is presented.

4.5.1 Theoretical Background

Our approach to model and execute a special kind of collaboration processes is theoretically based on *schema theory* [Schank77]. According to schema theory, generalized knowledge about a sequential list of the characteristic events involved in a common routine is called a *script* [Schank77] [Schank82]. Scripts can be used to organize knowledge, to assist recall, to guide behavior, to predict likely happenings, and to help us to make sense of our current experiences. People know how to behave and what to expect in particular situations by using scripts. Scripts are mental structures representing the person's knowledge about objects, people, or situations. They are derived from prior knowledge and experience, and set up expectations about what is probable and appropriate in relation to particular social contexts.

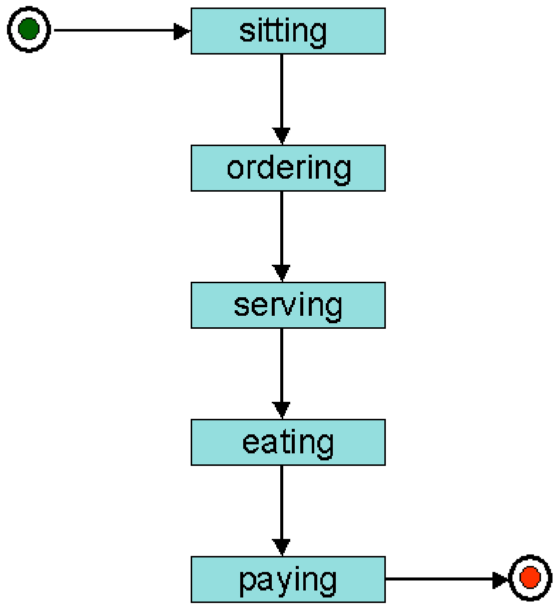


Figure 4.9: A Typical Restaurant Script

Figure 4.9 shows a diagram that illustrates a script of a typical restaurant, in which the process of eating at a restaurant is divided into five 'scenes': sitting, ordering, serving, eating, and paying. When a scene finishes, another scene may start. In this restaurant script, there are three roles: consumer, waiter, and cooker. The scripts embody knowledge about how people in a particular role (e.g. waiter, or customer) are expected to behave in each scene. For example, it is expected that a cook prepares the food that the customer ordered and a waiter passes the food to the customer in the serving scene. After being served, the customer should eat the food in the eating scene. Such expected behaviors are called behavior rules. The people with a certain role should follow the behavior rules in the social interaction. Violation of behavior rules will make trouble, for example, a waiter eats the food. It is important to note that there are many variations possible in this general script having to do with different types of restaurants or procedures, for example, MacDonald's or a Buffet. Such variations are opportunities for misunderstandings or incorrect inferences [Script theory].

4.5.2 Collaboration Protocol

In modern society, success increasingly relies on the collective effort of a group of people. This is because the problems we face are more and more complicated such that no single individual has the complete knowledge and all the necessary skills and they can not finish work in limited time. Group members need to cooperate in order to achieve their shared goal. There are two typical kinds of cooperative processes: collaboration processes and coordination processes.

4.5.2.1 Collaboration

Collaboration is defined here as a cooperative process where individuals share a common goal and need to make collective contributions to achieve the shared goal. There are two categories of collaboration: synchronous collaboration and asynchronous collaboration.

In asynchronous collaboration, the shared goal can be decomposed into sub-goals. Individuals have different roles and contribute seperately (i.e., in turn) in order to achieve these various sub-goals. When all sub-goals are achieved, their overall goal is finally achieved. A simple example of an asynchronous collaboration process is a 4×100 meters relay race. The overall process is decomposed into four phases and each member of a team with a given role is assigned to be responsible for running 100 meters in turn. The relay baton has to be handed over from the first member to the last member one by one. Whenever a member gets the relay baton, this member’s goal is to run and pass the relay baton to the next one or to reach the termination line as quickly as possible. At that time, the efforts of the rest members (e.g., running or shouting) have no or less influence on the result. Although all their contributions as a whole determine their success, the single contribution of individuals can be evaluated separately.

In synchronous collaboration, the shared goal can not be decomposed into sub-goals. All individuals with different roles often work together in order to achieve the common goal. At the end of the synchronous collaborative process, the single contributions of individuals are difficultly isolated. The product is an entity in which the results of individual contributions cannot be seen. The product as a whole is of central importance. A simple example is rowing a boat. There are two different roles in a team: rowers and a coxswain who steer the boat. No matter which role they take, all members have to contribute together from the beginning to the end. The whole process has only one phase. It is expected that members with different roles contribute in different ways. That is, the behavior rules belonging to each role must be complied with. Individuals have no distinct sub-goals, but only the common goal. There is no predefined routine that every member has to follow. Success of the team relies on the collective contribution of all members, but it is difficult to evaluate single contributions of individuals separately. Individuals’ efforts will influence the final result. For example, if one of them rows too hard, the balance may be broken and the boat will change its direction.

Through a close investigation of synchronous collaboration processes, we can further devide synchronous collaboration processes into two categories. Some synchronous collaboration processes have multiple distinct collaboration states. These states are not phases in the sense of the relay-race example given above – i.e. a sequence of steps. Rather they represent distinct conditions or situations in a collaboration process. The transitions between different states are triggered by some events, and not by predefined routines. A simple example is playing soccer. Roughly speaking, there are two distinct states: attacking and defending. Members of a team play different roles such as attacker, rear guard, and goalkeeper. In each state, it is expected that each member with a given role should behave appropriately. The transitions between states are triggered by events, rather than predefined routine. For example, in the state of attacking, if they score a goal or lose the control of the ball, the state automatically changes to defending. For each state, different strategies can be adopted. For example, in attacking state, they may attack from the centerline or from one of the sidelines. Which strategies they apply depend on the characteristics of the team and the characteristics of their opponent, and the specifics of the current situation. When they decide to use a strategy (e.g., centerline), each member with a given role should behave appropriately to realize the strategy. Furthermore, they can shift from the currently used strategy to another strategy (e.g., left sideline), as the situation changes. Each member has to adjust his/her behavior to fit the change. When the state changes, the strategy used in this state will be terminated immediately. For example, if they attack by using centerline-attacking strategy, most members should run forward. In particular, the attack players should look for a chance to approach the goal and the player who has the ball should pass it to him. However, when the ball is intercepted, the state changes into defending state. The currently used centerline-attacking strategy is given up immediately. From this simple example, we analyze the characteristics of such a kind of collaboration processes. A process with such characteristics is defined in this thesis as a *multi-state collaboration process.* However, other synchronous collaboration processes may have a unique collaboration state from the beginning to the end such as rowing a boat, as mentioned above. Such a synchronous collaboration process is called a *single-state collaboration process*.

4.5.2.2 Coordination

Coordination refers to a cooperative process where individuals or teams need to adjust their separate actions with those of others towards a shared goal. The main problem in coordination is the synchronization of people, actions and the consistency of the individual actions with respect to the whole process. Both synchronous collaboration processes and asynchronous collaboration processes need coordination.

In order to support coordination of asynchronous collaboration processes by using information technology, a lot of efforts have been made in the workflow area. Coordination of asynchronous collaboration processes primarily occurs at sub-goal or task level. This topic will be discussed in the next section.

In order to support synchronous collaboration process, a lot of computer-based support systems such as DOLPHIN [Streiz94] and COSOFT [Zhao94] [Zhao95] have been developed by providing a shared workspace that is accessible to all participants. Co-located or distributed participants can manipulate information objects in the shared workspace synchronously. Coordination of synchronous collaboration processes primarily occurs at the operation level. These systems support coordination of synchronous collaboration by controlling concurrent accesses to the same information object.

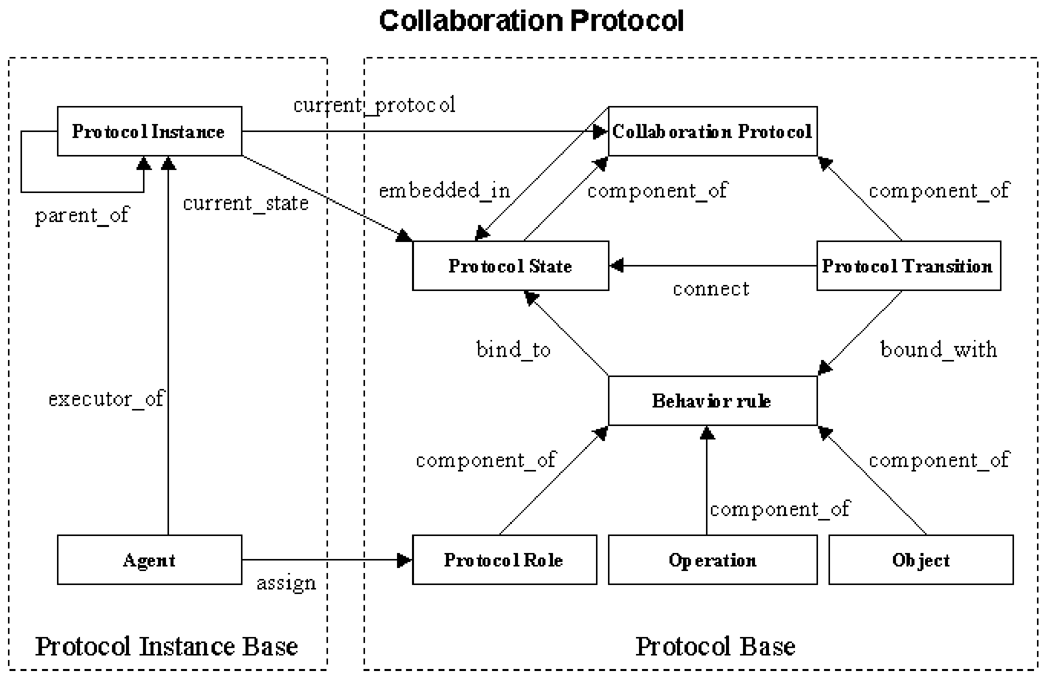


Figure 4.10: Conceptual Architecture of
Collaboration Protocols and Protocol Instances

In order to support multi-state collaboration process, concurrency control is insufficient, because concurrency control does not care of multiple states and multiple roles. This section presents an approach to support the coordination of multi-state collaboration processes. The central concept of this approach is the *collaboration protocol* that is defined here as a computational description of a collaboration policy (strategy) [Miao98b]. In terms of schema theory, a collaboration protocol is a computerised script. A collaboration protocol is described as an extended, hierarchical state-transition-diagram. As illustrated in Figure 4.10, a collaboration protocol consists of a set of *protocol states* connected by a set of *protocol transitions*. A sub-protocol may be embedded in a protocol state. A *behavior rule* combines a *protocol role*, an *operation*, and an *object*. A behavior rule can be bound to protocol states and protocol transitions. That is, a behavior is expected in some state, and a behavior may results in a state transition. As a description of a collaboration strategy, a collaboration protocol may have one or more instances, called *protocol instances*. A protocol instance is executed by one or more agents who play certain protocol roles. A protocol instance has a current state, in which all bound behavior rules are expected to be complied with. A behavior may result in a transition from the current state to another state. In addition, the executors of a protocol instance can initiate a sub-protocol that is embedded in the current state. A formal specification of collaboration protocol and protocol instance is given below.

4.5.2.1 Modeling Collaboration Protocols

This subsection formally specifies the related concepts to collaboration protocol and how to model a collaboration protocol.

Definition (Protocol Role): A *protocol role* represents a certain role played in a given type of collaboration processes. An agent with a role has a set of privileges and responsibilities in a multi-state collaboration process. For example, the possible protocol roles in the playing soccer processes are attacker, rear guard, and goalkeeper.

[ProtocolRole]

Definition (Operation): An *operation* represents a possible act. For example, the operations performed in a playing soccer process are control, pass, and shooting. In this definition, an operation is an abstract description of an act, but not an actual operation. However, it associates to an act operation such as creating a typed node.

[Operation]

Definition (Object): An *object* represents objects on which an operation performed. For example, the objects in a playing soccer process are ball, goal, and opponent.

[Object]

It is important to note that protocol role, operation and object are defined as abstract data types without further characteristics. This provides us with the basic building blocks to develop other notions for defining the collaboration protocol. When defining a concrete collaboration protocol, the protocol roles, operations and objects refer to collaboration protocol specific protocol roles, operations and objects.

Definition (Behavior rule): A *behavior rule* specifies which protocol role is permitted to perform which operation on which object. For example, a possible behavior rule in a playing soccer process is the goalkeeper holds the ball. Other team members can not hold the ball.

$\text{BehaviorRule} == \text{ProtocolRole} \leftrightarrow (\text{Operation} \leftrightarrow \text{Object})$

Definition (Protocol State): A *protocol state* specifies a certain condition or situation that is distinguished from other conditions or situations in the whole collaboration process. For example, a protocol state in a playing soccer process is defending. A protocol state is identified by a name.

```
— ProtocolState ———————————————
| name : LABEL
———————————————————————————————
```

Definition (Protocol Transition): A *protocol transition* specifies a transition between two protocol states. For example, a protocol transition in a playing soccer process is a change from attacking to defending. A protocol transition is identified by a label.

ProtocolTransition = = LABEL ↔ (ProtocolState ↔ ProtocolState)

Definition (Protocol Family): A *protocol family* denotes all possible collaboration protocols that are representing different strategies of the same kind of collaboration. For example, playing soccer is a kind of collaboration and playing volleyball is another kind of collaboration. When playing soccer, different strategies can be used, which form a family. Collaboration protocols within the same family are defined based on the same set of behavior rules. A protocol family is distinguished from other family by its name.

```
┌─ CollaborationProtocolFamily ──────────────────────────
│ name : STRING
└────────────────────────────────────────────────────────
```

Definition (Collaboration Protocol): A *collaboration protocol* (protocol for short in this thesis) represents a policy or a strategy that can be adopted in a given collaboration. For example, in playing soccer, an attack-based strategy can be defined as a protocol, and a defend-based strategy can be regarded as another one. These two strategies have not been in detail described in the example above. However, the sub-strategies (e.g., centerline attacking or sideline attacking) within a state (e.g., attacking) were discussed in the example above. A sub-strategy can be described as a sub-protocol as well. A protocol is distinguished from other protocols by its name. Each protocol belongs to a protocol family.

```
┌─ CollaborationProtocol ────────────────────────────────
│ name : STRING
│ protocolFamily : CollaborationProtocolFamily
└────────────────────────────────────────────────────────
```

Definition (Protocol Base): A *protocol base* consists of a set of protocol families, a set of protocols, and a set of functions. Each protocol state or protocol transition is defined for a certain protocol. Each protocol should have a start state from which a collaboration protocol begins to be executed. Each protocol role is defined for a certain protocol. There is a binding relation between a behavior rule to a protocol state. A protocol transition is bound to a behavior rule. Some protocols can be defined as sub-protocols that can be initiated in a certain protocol state. There are constrains in the protocol base. All protocol states and protocol transitions have to be defined for an existing protocol. If a protocol transition connects two protocol states, these two protocol states and the protocol transition must be defined for the same protocol. If two different protocol states are defined in the same protocol, their names and their bound behavior rules are not equal. Each protocol has only one start state, or the start state hasn't been defined. The start state of a protocol should be one of the defined protocol states of the protocol. If two protocol transitions of the same protocol are equal, it means that their names, departure state and arrival state are the same. If two protocols belong to the same family, their protocol roles should be equal. Furthermore, for two protocol states belonging to different protocols within the same family, they take the same name if and only if they are bound by the same set of behavior rules. This constraint is defined in order to support shifting protocols. For a detailed explanation see section 4.5.2.3.

ProtocolBase

$$
\begin{array}{l}
\text{protocolFamilies} : \mathbb{P}\ \text{CollaborationProtocolFamily} \\
\text{protocols} : \mathbb{P}\ \text{CollaborationProtocol} \\
\text{protocolState} : \text{ProtocolState} \rightarrow \text{CollaborationProtocol} \\
\text{protocolStartState} : \text{CollaborationProtocol} \rightarrow \text{ProtocolState} \\
\text{protocolTransition} : \text{ProtocolTransition} \rightarrow \text{CollaborationProtocol} \\
\text{protocolRole} : \text{ProtocolRole} \rightarrow \text{CollaborationProtocol} \\
\text{stateBinding} : \text{BehaviorRule} \rightarrow \text{ProtocolState} \\
\text{transitionBound} : \text{ProtocolTransition} \rightarrow \text{BehaviorRule} \\
\text{subProtocol} : \text{CollaborationProtocol} ⇸ \text{ProtocolState}
\end{array}
$$

$$
\begin{array}{l}
\mathbf{ran}\ \text{protocolState} \subseteq \text{protocols} \wedge \mathbf{ran}\ \text{protocolTransition} \subseteq \text{protocols} \\
(\forall\, s, d : \text{ProtocolState}; t : \text{ProtocolTransition} \mid \\
\quad \mathbf{first\ second}\ t = s \wedge \mathbf{second\ second}\ t = d \bullet \\
\quad (\text{protocolState}\ s) = (\text{protocolState}\ d) = (\text{protocolTransition}\ t)) \\
\\
(\forall\, s_1, s_2 : \text{ProtocolState} \mid \text{protocolState}\ s_1 = \text{protocolState}\ s_2 \wedge s_1 \neq s_2 \bullet \\
\quad s_1.\text{name} \neq s_2.\text{name}) \wedge \\
\quad \mathbf{dom}\ \text{stateBinding} \rhd \{ s_1 \} \neq \mathbf{dom}\ \text{stateBinding} \rhd \{ s_2 \} \\
\\
(\forall\, p : \text{CollaborationProtocol} \mid p \in \text{protocols} \bullet \#\, (\text{protocolStartState} \rhd \{ p \}) \leq 1) \\
(\forall\, s : \text{ProtocolState}; p : \text{CollaborationProtocol} \mid (p \mapsto s) \in \text{protocolStartState} \bullet \\
\quad s \in \mathbf{dom}\ \text{protocolState} \rhd \{ p \}) \\
\\
(\forall\, t_1, t_2 : \text{ProtocolTransition} \mid (\text{protocolTransition}\ t_1 = \text{protocolTransition}\ t_2 \wedge \\
\quad t_1 = t_2) \Leftrightarrow (\ \mathbf{first}\ t_1 = \mathbf{first}\ t_2 \wedge \\
\qquad\qquad\qquad \mathbf{first\ second}\ t_1 = \mathbf{first\ second}\ t_2 \wedge \\
\qquad\qquad\qquad \mathbf{second\ second}\ t_1 = \mathbf{second\ second}\ t_2) \\
\\
(\forall\, p_1, p_2 : \text{CollaborationProtocol} \mid p_1.\text{protocolFamily} = p_2.\text{protocolFamily} \bullet \\
\quad \mathbf{dom}\ \text{protocolRole}\ p_1 = \mathbf{dom}\ \text{protocolRole}\ p_2 \wedge \\
\quad (\forall\, s_1, s_2 : \text{ProtocolState} \mid s_1 \in p_1 \wedge s_2 \in p_2 \bullet s_1.\text{name} = s_2.\text{name} \Leftrightarrow \\
\quad (\mathbf{dom}\ \text{stateBinding} \rhd \{ s_1 \} = \mathbf{dom}\ \text{stateBinding} \rhd \{ s_2 \})))
\end{array}
$$

So far, data types and an abstract state for modeling collaboration protocols are defined. As mentioned in Chapter 5, a tool, called collaboration protocol definition tool has been developed for defining collaboration protocols. In the paragraphs below, the operations to define collaboration protocols are specified.

First of all, we have to create a protocol family before defining a protocol member in the family.

```
┌─ CreateProtocolFamilyOK ──────────────────────────────────
│ Δ ProtocolBase
│ name? : STRING
├───────────────────────────────
│ (∀ f : CollaborationProtocolFamily | f ∈ protocolFamilies •
│       f.name ≠ name? )
│
│ let family = = (μ CollaborationProtocolFamily | name = name?) •
│               protocolFamilies' = protocolFamilies ∪ { family }
│ protocols' = protocols
│ protocolState' = protocolState
│ protocolStartState' = protocolStartState
│ protocolTransition' = protocolTransition
│ protocolRole' = protocolRole
│ stateBinding' = stateBinding
│ transitionBound' = transitionBound
│ subProtocol' = subProtocol
└───────────────────────────────────────────────────────────
```

When a protocol family exists, we can define a protocol in this family.

```
┌─ CreateCollaborationProtocolOK ───────────────────────────
│ Δ ProtocolBase
│ name? : STRING
│ family? : CollaborationProtocolFamily
├───────────────────────────────
│ (∀ p : CollaborationProtocol | p ∈ protocols • p.name ≠ name?) ∧
│ family? ∈ protocolFamilies
│ let aProtocol = = (μ CollaborationProtocol | name = name? ∧
│                                            protocolFamily = family?) •
│               protocols' = protocols ∪ { aProtocol }
│ protocolState' = protocolState
│ protocolStartState' = protocolStartState
│ protocolTransition' = protocolTransition
│ protocolRole' = protocolRole
│ stateBinding' = stateBinding
│ transitionBound' = transitionBound
│ protocolFalimies' = protocolFamilies
│ subProtocol' = subProtocol
└───────────────────────────────────────────────────────────
```

The operations to define protocol states and protocol transitions are specified as follows:

CreateProtocolStateOK

Δ ProtocolBase
name? : LABEL
protocol? : CollaborationProtocol

---

protocol? $\in$ protocols
($\forall$ s : ProtocolState | protocolState s = protocol? $\bullet$ s.name $\neq$ name?)

let state = = (μ ProtocolState | name = name?) $\bullet$
    protocolState' = protocolState $\cup$ { state $\mapsto$ protocol?}
protocols' = protocol
protocolStartState' = protocolStartState
protocolTransition' = protocolTransition
protocolRole' = protocolRole
stateBinding' = stateBinding
transitionBound' = transitionBound
protocolFalimies' = protocolFamilies
subProtocol' = subProtocol

CreateProtocolTransitionOK

Δ ProtocolBase
name? : LABEL
protocol? : CollaborationProtocol
s?, d? : ProtocolState

---

protocol? $\in$ protocols $\wedge$
(s? $\mapsto$ protocol?) $\in$ protocolState $\wedge$ (d? $\mapsto$ protocol?) $\in$ protocolState

let transition = = (μ ProtocolTransition | $\bullet$ (name? $\mapsto$ s? $\mapsto$ d?)) $\bullet$
    protocolTransition' = protocolTransition $\cup$ { transition $\mapsto$ protocol?}
protocols' = protocol
protocolStartState' = protocolStartState
protocolRole' = protocolRole
stateBinding' = stateBinding
transitionBound' = transitionBound
protocolFalimies' = protocolFamilies
subProtocol' = subProtocol

A protocol role is defined by performing the following operation:

DefineProtocolRoleOK

Δ ProtocolBase
role? : ProtocolRole

protocol? : CollaborationProtocol

$(role? \mapsto protocol?) \notin protocolRole$

protocols' = protocols
protocolState' = protocolState
protocolStartState' = protocolStartState
protocolTransition' = protocolTransition
protocolRole' = protocolRole $\cup$ { role? $\mapsto$ protocol? }
stateBinding' = stateBinding
transitionBound' = transitionBound
protocolFalimies' = protocolFamilies
subProtocol' = subProtocol

For executing a protocol, it is needed to define a start state for the protocol. Otherwise, the initiated protocol has no start point.

DefineProtocolStartStateOK
Δ ProtocolBase
state? : ProtocolRole
protocol? : CollaborationProtocol

$(state? \mapsto protocol?) \in protocolState$

protocols' = protocols
protocolState' = protocolState
protocolStartState' = (protocolStartState $\rhd$ { protocol? }) $\cup$ {state? $\mapsto$ protocol?}
protocolTransition' = protocolTransition
protocolRole' = protocolRole
stateBinding' = stateBinding
transitionBound' = transitionBound
protocolFalimies' = protocolFamilies
subProtocol' = subProtocol

Binding a behavior rule with a protocol state or a protocol transition for a protocol is realized by performing the following operation:

BindRuleWithProtocolStateOK
Δ ProtocolBase
rule? : BehaviorRule
state? : ProtocolRole
protocol? : CollaborationProtocol

$(state? \mapsto protocol?) \in protocolState \wedge (rule? \mapsto state?) \notin stateBinding$

protocols' = protocols

protocolState’ = protocolState
protocolStartState’ = protocolStartState
protocolTransition’ = protocolTransition
protocolRole’ = protocolRole
stateBinding’ = stateBinding $\cup$ { rule? $\mapsto$ state?}
transitionBound’ = transitionBound
protocolFalimies’ = protocolFamilies
subProtocol’ = subProtocol

BindRuleWithProtocolTransitionOK
Δ ProtocolBase
rule? : BehaviorRule
transition? : ProtocolTransition
protocol? : CollaborationProtocol

(transition? $\mapsto$ protocol?) $\in$ protocolState $\wedge$
(transition? $\mapsto$ rule?) $\notin$ transitionBound

protocols’ = protocols
protocolState’ = protocolState
protocolStartState’ = protocolStartState
protocolTransition’ = protocolTransition
protocolRole’ = protocolRole
stateBinding’ = stateBinding
transitionBound’ = transitionBound $\cup$ { transition? $\mapsto$ rule?}
protocolFalimies’ = protocolFamilies
subProtocol’ = subProtocol

A protocol can be defined as a sub-protocol under a state. Consequently, the sub-protocol can be initiated only under this state.

DefineSubProtocolOK
Δ ProtocolBase
state? : ProtocolRole
subProtocol? : CollaborationProtocol

(subProtocol? $\mapsto$ state?) $\notin$ subProtocol

protocols’ = protocols
protocolState’ = protocolState
protocolStartState’ = protocolStartState
protocolTransition’ = protocolTransition
protocolRole’ = protocolRole
stateBinding’ = stateBinding
transitionBound’ = transitionBound
protocolFalimies’ = protocolFamilies

| subProtocol' = subProtocol ∪ { subProtocol? ↦ state?}

The major operations to define collaboration protocols were specified above. Collaboration protocols can be predefined and can be stored in a protocol base. The predefined protocols can be initiated as an instance when using them. It is important to note that the predefined protocol can be modified even when it is executed. It provides flexibility for users to change the definition of the currently used protocol to fit some situations that were not predicted when the protocol was defined. In the next subsection, we discuss how an instance of a protocol is initiated and executed.

4.5.2.2 Protocol Instances

The purpose of defining a collaboration protocol is to use it at run time. When group members perform a collaborative activity, they can use a pre-defined collaboration protocol to guide and control the group interaction. A collaboration protocol can be executed as a *protocol instance*. One collaboration protocol can have more than one protocol instance at the same time. Each protocol instance is executed independently following the definition of the chosen collaboration protocol.

Definition (Protocol Instance): A *protocol instance* represents an execution of a collaboration protocol.

[ProtocolInstance]

Definition (Protocol Instance Base): A *protocol instance base* consists of a protocol base and a set of a protocol instances. Each protocol instance exploits definitely one collaboration protocol as the currently executed protocol. Each protocol instance records the current state of protocol execution. An assign relation specifies which agent is assigned to a certain protocol role of a protocol instance. The parentOf relation is used to specify that the first protocol instance is a parent of the second one. In other words, the second protocol instance is initiated from the current state of the first protocol instance. In the protocol instance base, the set of protocol instances is exactly equal to the domain of the currentProtocol function and equal to the domain of the currentState function. The protocol role to be assigned for a protocol instance should be one of the protocol roles defined in the protocol that serves as the current protocol of the protocol instance. If one protocol instance is the parent of another protocol instance, the currently used protocol of the latter protocol instance should be one of the sub-protocols defined in the current state of former protocol instance.

— ProtocolInstanceBase —
| ProtocolBase
| protocolInstances : $\mathbb{P}$ ProtocolInstance
| currentProtocol : ProtocolInstance → CollaborationProtocol
| currentState : ProtocolInstance → ProtocolState
| assign : Agent → (ProtocolRole → ProtocolInstance)
| parentOf : ProtocolInstance ⇸ ProtocolInstance

$\text{protocolInstances} = \textbf{dom}\ \text{currentProtocol} = \textbf{dom}\ \text{currentState}$

$\forall\, a : \text{assign} \mid (\textbf{first second}\ a \mapsto \text{currentProtocol}\ (\textbf{second second}\ a)) \in \text{protocolRole}$

$\forall\, i_1, i_2 : \text{ProtocolInstance} \mid (i_1 \mapsto i_2) \in \text{parentOf} \bullet$

$\quad \text{currentProtocol}\ i_2 \in \textbf{dom}\ \text{subProtocol} \rhd \{\text{currentState}\ i_1\}$

A protocol instance can be initiated simply by choosing a defined protocol in the protocol base. After a protocol instance is initiated, it will begin execution at the start state of the collaboration protocol. That is, the current state of the instance is the start state.

InitiateRootProtocolInstanceOK

$\Delta\ \text{ProtocolInstanceBase}$

$\Xi\ \text{ProtocolBase}$

$\text{selectedProtocol?} : \text{CollaborationProtocol}$

$\text{selectedProtocol?} \in \text{protocols}$

$\text{initiatedInstance} == (\mu\ \text{ProtocolInstance})$

$\text{protocolInstances}' = \text{protocolInstances} \cup \{\text{initiatedInstance}\}$

$\text{currentProtocol}' = \text{currentProtocol} \cup \{\text{initiatedInstance} \mapsto \text{selectedProtocol?}\}$

$\text{currentState}' = \text{currentState} \cup$

$\qquad \{\text{initiatedInstance} \mapsto (\text{protocolStartState selectedProtocol?})\}$

$\text{assign}' = \text{assign}$

$\text{parentOf}' = \text{parentOf}$

When executing a protocol instance, the initiator of the protocol instance needs to assign protocol-roles to the potential participants of the collaborative learning activity.

AssignProtocolRoleOK

$\Delta\ \text{ProtocolInstanceBase}$

$\Xi\ \text{ProtocolBase}$

$\text{instance?} : \text{ProtocolInstance}$

$\text{agent?} : \text{Agent}$

$\text{role?} : \text{ProtocolRole}$

$\text{instance?} \in \text{protocolInstances} \wedge$

$\text{role?} \in \textbf{dom}\ \text{protocolRole} \rhd \{\text{currentProtocol instance?}\}$

$\text{protocolInstances}' = \text{protocolInstances}$

$\text{currentProtocol}' = \text{currentProtocol}$

$\text{currentState}' = \text{currentState}$

$\text{assign}' = \text{assign} \cup \{\text{agent?} \mapsto \text{role?} \mapsto \text{instance?}\}$

$\text{parentOf}' = \text{parentOf}$

A protocol can be initiated as a sub-protocol instance under the current state of a protocol instance, if the selected protocol is defined as a sub-protocol of state of the protocol, which is exactly the currently used protocol of the protocol instance.

InitiateSubProtocolInstanceOK

Δ ProtocolInstanceBase
Ξ ProtocolBase
instance? : ProtocolInstance
subProtocol? : CollaborationProtocol

$\text{instance?} \in \text{protocolInstances} \wedge \text{selectedProtocol?} \in \text{protocols} \wedge$
$\text{selectedProtocol?} \in \textbf{dom}\ \text{subProtocol} \rhd \{ \text{currentState instance?} \}$

$\text{subProtocolInstance} == (\mu\ \text{ProtocolInstance})$
$\text{protocolInstances}' = \text{protocolInstances} \cup \{ \text{subProtocolInstance} \}$
$\text{currentProtocol}' = \text{currentProtocol} \cup \{ \text{subProtocolInstance} \mapsto \text{selectedProtocol?} \}$
$\text{currentState}' = \text{currentState} \cup$
$\qquad \{ \text{subProtocolInstance} \mapsto (\text{protocolStartState selectedProtocol?}) \}$
$\text{assign}' = \text{assign}$
$\text{parentOf}' = \text{parentOf} \cup \{ \text{instance?} \mapsto \text{subProtocolInstance} \}$

In order to make the specifications of the following operations short and clear, we define a relation "_belongTo_". The first element of a pair in this relation is an actor and the second element of the pair is a group. This means that an actor directly or indirectly belongs to the group.

$\_\text{belongTo}\_ = \{ \forall\ \text{anActor : Actor; aGroup : Group} \mid$
$\qquad \text{anActor} \in \text{actors} \wedge \text{aGroup} \in \text{groups} \wedge$
$\qquad ((\text{anActor} \mapsto \text{aGroup}) \in \text{aMemberOf} \vee$
$\qquad\ \ (\exists\ \text{anotherGroup : Group} \mid \text{anotherGroup} \in \text{groups} \bullet$
$\qquad\qquad (\text{anActor} \mapsto \text{anotherGroup}) \in \text{aMemberOf} \vee$
$\qquad\qquad (\text{anotherGroup} \mapsto \text{aGroup}) \in \text{aSubGroupOf}^{+})) \bullet$
$\qquad \text{anActor} \mapsto \text{aGroup} \}$

When an actor performs an operation on an object, it will be checked whether the actor is permitted to do it in the current state of the protocol instance according to the bound behavior rules, assignment, and relationships between agents. It is needed to note that the operation and object specified so far are still abstract notions. In a concrete collaboration protocol, the operation and object has concrete meaning. Therefore, there is no change in the protocol base and protocol instance base so far.

```
┌─ PerformOperationOnObjectOK ──────────────────────────
│ Ξ ProtocolInstanceBase
│ Ξ ProtocolBase
│ instance? : ProtocolInstance
│ actor? : Actor
│ operation? : Operation
│ object? : Object
├──────────────────────────────────
│ instance? ∈ protocolInstances ∧
│ ( ∃ rule : BehaviorRule | rule ∈ dom stateBinding ▷{ currentState instance? }•
│       first second rule = operation? ∧ second second rule = object? ∧
│       (∃ agent : Agent; a : assign | agent = first a ∧
│               first second assign = first rule ∧
│               second second assign = instance? •
│               actor actor? = agent ∨
│               (∃ g : Group | actor? _belongTo_ g ∧ group g = agent)))
└──────────────────────────────────────────────────────
```

When an actor with an appropriate protocol-role performs an operation that is bound to a protocol transition, it will trigger a state transition according to the definition of collaboration protocol. The destination state of the transition then becomes the current state of the protocol instance. All instances initiated in the previous state will be terminated automatically.

```
┌─ TriggerTransitionOK ─────────────────────────────
│ Δ ProtocolInstanceBase
│ Ξ ProtocolBase
│ instance? : ProtocolInstance
│ actor? : Actor
│ operation? : Operation
│ object? : Object
├──────────────────────────────────
│ instance? ∈ protocolInstances ∧
│ (∃ transition : ProtocolTransition | first second transition = currentState instance?∧
│       first second transitionBound transition = operation? ∧
│       second second transitionBound transition = object? ∧
│       (∃ agent : Agent; a : assign | agent = first a ∧
│               first second assign = first transitionBound transition ∧
│               second second assign = instance? •
│               actor actor? = agent ∨
│               (∃ g : Group | actor? _belongTo_ g ∧ group g = agent)))
│
│ protocolInstances' = protocolInstances \ { ∀ child : ProtocolInstance |
│               (instance? ↦ child ) ∈ parentOf ⁺ • child }
│
│ currentProtocol' = currentProtocol \ { ∀ child : ProtocolInstance |
│               (instance? ↦ child ) ∈ parentOf ⁺ • child ↦ (currentProtocol child )}
```

currentState’ = currentState \ { ∀ child : ProtocolInstance |
(instance? ↦ child) ∈ parentOf $^+$ • child ↦ (currentState child)}
⊕ { initiatedInstance ↦
($\exists_1$ transition : ProtocolTransition | **first** transition = currentState instance? ∧
first second transitionBound transition = operation? ∧
second second transitionBound transition = object? ∧
(∃ agent : Agent; a : assign | agent = **first** a ∧
first second assign = **first** transitionBound transition ∧
second second assign = instance? •
actor actor? = agent ∨
(∃ g : Group | actor? _belongTo_ g ∧ group g = agent)) •
second second transition)}

assign’ = assign \{ ∀ a : assign | **second second** a = instance? • a}\
{ ∀ child : ProtocolInstance; a : assign | (instance? ↦ child) ∈ parentOf $^+$ ∧
second second a = child • a }

parentOf’ = (∀ child : ProtocolInstance | (instance? ↦ child) ∈ parentOf $^+$ •
parentOf ⩥ { child })

The operation to terminate a protocol instance means to terminate the protocol instance itself and it children.

TerminateInstanceOK
Δ ProtocolInstanceBase
Ξ ProtocolBase
instance? : ProtocolInstance

instance? ∈ protocolInstances

protocolInstances’ = protocolInstances \ { instance?}\
{ ∀ child : ProtocolInstance | (instance? ↦ child) ∈ parentOf $^+$ • child }

currentProtocol’ = currentProtocol \ { instance? ↦ (currentProtocol instance?)} \
{ ∀ child : ProtocolInstance | (instance? ↦ child) ∈ parentOf $^+$•
child ↦ (currentProtocol child)}

currentState’ = currentState \ { instance? ↦ (currentState instance?)} \
{ ∀ child : ProtocolInstance | (instance? ↦ child) ∈ parentOf $^+$ •
child ↦ (currentState child)}

assign’ = assign \{ ∀ a : assign | **second second** a = instance? • a}\

$\{ \forall child : ProtocolInstance; a : assign \mid (instance? \mapsto child) \in parentOf^{+} \wedge$
second second $a = child \bullet a \}$

$parentOf' = (\forall child : ProtocolInstance \mid (instance? \mapsto child) \in parentOf^{+} \bullet$
$parentOf \ntriangleright \{ child \})$

The operations to execute abstract collaboration protocols are now specified. We will discuss concrete collaboration protocols and its execution when we apply this approach to support problem based learning in the next section. Before discussing it, we discuss an open issue of collaboration protocols: how to shift between protocols.

4.5.2.3 Shifting between Collaboration Protocols

As discussed above, group interaction is supported at run time by the execution of a protocol instance of a pre-defined collaboration protocol. In order to support a certain collaborative activity (e.g., playing soccer), a family of collaboration protocols can be defined. As we known, team members vary in age, running speed, skills, personal character, and so on. Different teams may exploit different collaboration protocols to carry out their collaboration processes, because of the characteristics of the teams or the characteristics of their opponent. There is no single collaboration protocol that is suitable to every type of team and to every match. Furthermore, as team conducts a collaborative activity, some factors may change over time. For example, when a team attacks by using the strategy of centerline breakthrough, the opponent defends by using a strategy of tight formation in centerline. Therefore, teams may want to shift from the currently used collaboration protocol to another collaboration protocol to fit the changes.

A simple approach to shift between protocols is to terminate the currently executed protocol instance first and then to initiate a new protocol instance based on the target collaboration protocol. However, in some cases, this is not allowed. For example, in playing soccer, it is impossible to resume an attack from the beginning. In some cases, it is possible. However, the execution environment of the current protocol instance will be lost upon terminating the current protocol instance. The group members then have to repeat efforts to reach the equivalent state in the new protocol instance. For example, when a group uses the "first-request-first-take" floor token control protocol, the members' requests for the floor token are recorded in the request queue of the current protocol instance. If the group wants to change the floor token control protocol by stopping the current protocol instance and initiating a new protocol instance of another floor token control protocol (e.g., the "mediator-assigning" floor token control protocol), the request queue of the current protocol instance will be lost. Group members need to request the new floor token in the new protocol instance again.

An alternative approach is to enable end-users to modify the definition of the collaboration protocol in use at run time. However, our experiences reveal that it is a difficult and time-consuming task for end-users to change a collaboration protocol dynamically.

Therefore, we adopted an approach where end-users are enabled to shift between collaboration protocols by simply choosing another protocol from the same family of pre-defined collaboration protocols [Miao00c]. Because all collaboration protocols in a family are carefully defined to support the same collaborative activity, they use identical sets of objects, such as the types of objects and their associated operations, and identical protocol-roles. Collaboration protocols within the same family vary in the definition of the state of collaboration and transitions, and in the binding relation. An analysis of collaboration protocols in the same family showed that some states of collaboration defined in different collaboration protocols are bound to the same set of behavior rules, and that some states of collaboration defined in different collaboration protocols are refined to different degrees. Manually, for any state of any collaboration protocol from which to shift we can find an appropriate state in any target collaboration protocol from where to continue execution after shifting. Therefore, a family of collaboration protocols can be defined by a protocol diagram in which all collaboration protocols are contained as sub-diagrams. A new *shift relation* is defined by edges that connect each state in any sub-diagram to a state in any other sub-diagram, from which execution can be continued after a shift.

In order to discuss the general case, we suppose that in a family there are m collaboration protocols denoted by $\mathbf{P}_1$, $\mathbf{P}_2$, ... $\mathbf{P}_m$, respectively. The numbers of states in each collaboration protocol are n_1, n_2, ..., n_m, respectively. Then, the number of states in the diagram of the protocol family is $n = n_1 + n_2 + \ldots + n_m$ and the number of shift arrows is $\sum_{i=1}^{n} n_i * (m - 1)$. When adding a new collaboration protocol with n_k states in this family, we have to add $\sum_{i=1}^{n} n_i + m * n_k$ shift arrows. This diagram becomes very complex when the number of collaboration protocols and the numbers of states in these protocols are large. It is also difficult to maintain the diagram when adding or removing or modifying collaboration protocols. Thus, manual maintenance of the shift relation is difficult.

In order to shift between collaboration protocols in a systematic manner, we defined a rule-based method for capturing the shift relation. Readers should remember that a state within a collaboration protocol is distinguished from other states, because a certain sub set of behavior rules is bound to this state. Within a collaboration protocol, two different states are bound to two different sub sets of behavior rules. Different collaboration protocols within a protocol family are defined to support the same type of collaboration. That is, the set of protocol roles, objects and their associated operations are the same. Collaboration protocols are different from each other because the states in different protocols are defined in different ways. That is, in different collaboration protocols, states are refined or specialized to different degree by binding to different sub sets of behavior rules. According to the sub set of behavior rules bound to states, the relation between two states within two different collaboration protocols can fall into one of four categories. Given s_1 is a state of a collaboration protocol $\mathbf{P}_1$ and s_2 is a state of a collaboration protocol $\mathbf{P}_2$ ($\mathbf{P}_1 \neq \mathbf{P}_2$), the relation between **first** stateBinding$\rhd\{s_1\}$ and **first** stateBinding$\rhd\{s_2\}$ leads to one of the following four situations:

1) equal: **first** stateBinding$\rhd\{s_1\}$ = **first** stateBinding$\rhd\{s_2\}$;
2) contained: **first** stateBinding$\rhd\{s_1\} \subset$ **first** stateBinding$\rhd\{s_2\} \vee$
   first stateBinding$\rhd\{s_1\} \supset$ **first** stateBinding$\rhd\{s_2\}$;
3) intersected: **first** stateBinding$\rhd\{s_1\} \not\subset$ **first** stateBinding$\rhd\{s_2\} \wedge$
   first stateBinding$\rhd\{s_2\} \not\subset$ **first** stateBinding$\rhd\{s_1\} \wedge$
   (**first** stateBinding$\rhd\{s_1\} \cap$ **first** stateBinding$\rhd\{s_2\} \neq \emptyset$);
4) separated: **first** stateBinding$\rhd\{s_1\} \cap$ **first** stateBinding$\rhd\{s_2\} = \emptyset$.

For all collaboration protocols in the same family, we define a unified set of labels. According to these four situations we can establish a shift relation between labels of states instead of establishing a shift relation between states of two collaboration protocols. In the first case, we label two states by the same name. In the other three cases, we have to use different names for these two states.

Definition (Label Relation Graph): A *label relation graph* represents relations between labels assigned to protocol states of all protocols within the same family.

— LabelRelationGraph ————————————————

aProtocolFamily : CollaborationProtocolFamily

labels: $\mathbb{P}$ LABEL
uniDirectedRelation : LABEL $\mathrel{+\!\!\!\rightarrow}$ LABEL
biDirectedRelation : LABEL $\mathrel{+\!\!\!\rightarrow}$ LABEL

————————————————

labels = $\{\forall$ s : ProtocolState | (protocolState s).protocolFamily = aProtocolFamily $\bullet$
  s.name $\}$

uniDirectedRelation = $\{\forall\, s_1, s_2$: ProtocolState |
  protocolState $s_1 \neq$ protocolState $s_2 \wedge$
  s_1.name $\in$ labels $\wedge$ s_2.name $\in$ labels $\wedge$
  dom stateBinding $\rhd \{s_1\} \supset$ **dom** stateBinding $\rhd\{s_2\}\bullet$
  s_1.name $\mapsto$ s_2.name $\}$

biDirectedRelation $\subseteq \{\forall\, s_1, s_2$: ProtocolState |
  protocolState $s_1 \neq$ protocolState $s_2 \wedge$
  s_1.name $\in$ labels $\wedge$ s_2.name $\in$ labels $\wedge$
  (**dom** stateBinding $\rhd \{s_1\} \cap$ **dom** stateBinding $\rhd \{s_2\}) \neq \emptyset \wedge$
  dom stateBinding $\rhd \{s_1\} \not\subset$ **dom** stateBinding $\rhd \{s_2\}\wedge$
  dom stateBinding $\rhd \{s_2\} \not\subset$ **dom** stateBinding $\rhd\{s_1\}\bullet$
  s_1.name $\mapsto$ s_2.name$\}$

$(\forall\ label_1, label_2 : LABEL \mid (label_1 \mapsto label_2) \in biDirectedRelation \bullet$
$\quad (label_2 \mapsto label_1) \in biDirectedRelation$

It is important to notice that not all intersected states have bi-directional relations between their labels. In some cases, the behavior rules bound to two states are almost the same but they have subtle difference because they are defined to emphasize or ignore intentionally some details more or less. They can be regarded as mutually shiftable states and a bi-directional relation can be defined between the labels. In other cases, no shift should be possible between two states although the behavior rules bound to the two states are associated to more or less common behavior rules. Thus, the bi-directional relations need to be designed carefully by protocol designers.

After defining the label relation graph, we can define the operation to shift between protocols and describe the algorithm. It is important to note that Z language is used to specify the change of an abstract state before and after an operation. How to implement this change is hidden in the specification. It gives a freedom for software developer to code by using different algorithm. Therefore, in this thesis the algorithm used to shift between protocols is specified separately by using a Z language-like description method.

ShiftProtocolOK

Δ ProtocolInstanceBase
Ξ ProtocolBase
instance? : ProtocolInstance
anotherProtocol? : CollaborationProtocol

$instance? \in protocolInstances \wedge anotherProtocol? \in protocols \wedge$
$anotherProtocol?.protocolFamily = (currentProtocol\ instance?).protocolFamily$

$protocolInstances' = protocolInstances \setminus \{ child : ProtocolInstance \mid$
$\quad (instance? \mapsto child) \in parentOf^{+} \bullet child \}$

$currentProtocol' = currentProtocol \oplus \{ instance? \mapsto anotherProtocol? \} \setminus$
$\quad \{ child : ProtocolInstance \mid (instance? \mapsto child) \in parentOf^{+} \bullet$
$\quad child \mapsto (currentProtocol\ child) \}$

$currentState' = currentState \setminus \{ child : ProtocolInstance \mid$
$\quad (instance? \mapsto child) \in parentOf^{+} \bullet child \mapsto (currentState\ child) \}$
$\quad \oplus \{ instance? \mapsto (\textit{the value of the return state of the following algorithm}) \}$
$assign' = assign$
$parentOf' = (\forall\ child : ProtocolInstance \mid (instance? \mapsto child) \in parentOf^{+} \bullet$
$\quad parentOf \rhd \{ child \})$

```
Step 1: if ( $\exists$ state : ProtocolState | (state $\mapsto$ anotherProtocol?) $\in$ protocolState $\wedge$
                (currentState instance?).name = state.name ) then return state
Step 2: if ( $\exists$ state : ProtocolState | (state $\mapsto$ anotherProtocol?) $\in$ protocolState $\wedge$
                (currentState instance?).name $\mapsto$ state.name ) $\in$ biDirectedRelation )
                then return state
                else let searchPoint = (currentState instance?).name
Step 3: let Children = { child : LABEL | child $\in$ labels $\wedge$
                (searchPoint $\mapsto$ child ) $\in$ uniDirectedRelation }
                if #Children = $\varnothing$ $\wedge$ (searchPoint $\neq$ (currentState instance?).name)
                  then return up level
        if ($\exists$ sequence : iseq ProtocolState, someChildren : $\mathbb{P}$ LABEL |
           someChildren $\subseteq$ Children $\wedge$
           ($\forall$ item$_1$, item$_2$ : sequence |
                (second item$_1$ $\mapsto$ anotherProtocol?) $\in$ protocolState $\wedge$
                (second item$_2$ $\mapsto$ anotherProtocol?) $\in$ protocolState $\wedge$
                (second item$_1$ $\mapsto$ second item$_2$ ) $\in$
                (ran (dom protocolTransition $\rhd$ { anotherProtocol?})) $^+$ $\bullet$
                        first item$_1$ > first item$_2$ ) $\wedge$
                ($\forall$ state : ProtocolState | state $\in$ ran sequence $\bullet$
                        ($\exists$ c : someChildren | state.name = c $\wedge$
                        (state $\mapsto$ anotherProtocol?) $\in$ protocolState ))
           then return sequence 1
        if ($\exists$ sequence : seq$_1$ ProtocolState, someChildren : $\mathbb{P}$ LABEL |
           someChildren $\subseteq$ Children $\wedge$
           ($\forall$ item$_1$, item$_2$ : sequence |
                (second item$_1$ $\mapsto$ anotherProtocol?) $\in$ protocolState $\wedge$
                (second item$_2$ $\mapsto$ anotherProtocol?) $\in$ protocolState $\wedge$
                (second item$_1$ $\mapsto$ second item$_2$ ) $\in$
                        (ran (dom protocolTransition $\rhd$ { anotherProtocol?})) $^+$ ) $\wedge$
                ($\forall$ state : ProtocolState | state $\in$ ran sequence $\bullet$
                        ($\exists$ c : someChildren | state.name = c $\wedge$
                        (state $\mapsto$ anotherProtocol?) $\in$ protocolState ))
           then let transitiveRelation = { n : $\mathbb{N}$, state : ProtocolState |
                        state $\in$ ran sequence $\wedge$
                        (protocolStartState anotherProtocol? $\mapsto$ state) =
                        (ran (dom protocolTransition $\rhd$ { anotherProtocol?})) $^n$ $\bullet$
                        n $\mapsto$ state}
                return ( item : transitiveRelation |
                        first item = min dom transitiveRelation $\bullet$ second item)
           else for each c : Children let searchPoint = c goto Step 3
                let searchPoint = (currentState instance?).name
Step 4: let Fathers = { father : LABEL | father $\in$ labels $\wedge$
```

```
            $(father \mapsto searchPoint) \in uniDirectedRelation \}$
            **if** $\# Fathers = \emptyset \wedge (searchPoint \neq (currentState\ instance?).name)$
               **then return** up level
      **if** $(\exists\ sequence : \mathbf{iseq}\ ProtocolState, parents : \mathbb{P}\ LABEL \mid$
         $parents \subseteq Fathers \wedge$
         $(\forall\ item_1, item_2 : sequence \mid$
            $(\mathbf{second}\ item_1 \mapsto anotherProtocol?) \in protocolState \wedge$
            $(\mathbf{second}\ item_2 \mapsto anotherProtocol?) \in protocolState \wedge$
            $(\mathbf{second}\ item_1 \mapsto \mathbf{second}\ item_2) \in$
            $(\mathbf{ran}\ (\mathbf{dom}\ protocolTransition \rhd \{ anotherProtocol?\}))^{+} \bullet$
                  $\mathbf{first}\ item_1 > \mathbf{first}\ item_2\ ) \wedge$
            $(\forall\ state : ProtocolState \mid state \in \mathbf{ran}\ sequence \bullet$
                  $(\exists\ c : someChildren \mid state.name = c \wedge$
                  $(state \mapsto anotherProtocol?) \in protocolState\ ))$
         **then return** sequence 1
      **if** $(\exists\ sequence : \mathbf{seq}_1\ ProtocolState, someChildren : \mathbb{P}\ LABEL \mid$
         $parents \subseteq Fathers \wedge$
         $(\forall\ item_1, item_2 : sequence \mid$
            $(\mathbf{second}\ item_1 \mapsto anotherProtocol?) \in protocolState \wedge$
            $(\mathbf{second}\ item_2 \mapsto anotherProtocol?) \in protocolState \wedge$
            $(\mathbf{second}\ item_1 \mapsto \mathbf{second}\ item_2) \in$
                  $(\mathbf{ran}\ (\mathbf{dom}\ protocolTransition \rhd \{ anotherProtocol?\}))^{+}\ ) \wedge$
            $(\forall\ state : ProtocolState \mid state \in \mathbf{ran}\ sequence \bullet$
                  $(\exists\ c : someChildren \mid state.name = c \wedge$
                  $(state \mapsto anotherProtocol?) \in protocolState\ ))$
         **then let** $transitiveRelation = \{\ n : \mathbb{N}, state : ProtocolState \mid$
                  $state \in \mathbf{ran}\ sequence \wedge$
                  $(protocolStartState\ anotherProtocol? \mapsto state) =$
                  $(\mathbf{ran}\ (\mathbf{dom}\ protocolTransition \rhd \{ anotherProtocol?\}))^{n} \bullet$
                  $n \mapsto state\}$
            **return** $(\ item : transitiveRelation \mid$
                  $\mathbf{first}\ item = \mathbf{min}\ \mathbf{dom}\ transitiveRelation \bullet \mathbf{second}\ item)$
         **else for each** f : Fathers **let** searchPoint = f **goto** Step 3
Step 5: **return** protocolStartState anotherProtocol?
```

The systematic method to model and execute collaboration protocols and shift between protocols has now been formally specified. In the next section, we apply this method to problem based learning.

4.5.3 PBL-protocols

In the last subsection, we discussed the method to support the coordination of multi-state collaboration processes on the operational level by guiding and controlling group interaction. This is a general-purpose method that can be applied to many application domains. An application example is to support collaborative design [Miao98b]. The prerequisite of its application is that the collaboration processes to be supported should have the same characteristics as discussed in the last subsection. Through an analysis of collaborative learning processes, we found that some collaborative learning processes have those characteristics. The approach was applied to support such collaborative learning processes. The concept of *learning protocol* was established, which denotes a computational description of such collaborative learning processes [Pfister98b]. When applying the approach to collaborative learning, this approach was developed to fit the collaborative learning domain and schema theory was used as the theoretical basis of this approach [Wessner99]. Furthermore, we applied this approach to support problem based learning.

In order to apply the approach to a collaborative activity, two important prerequisites have to be met. The first one is whether multiple distinct states and the transitions between them exist in the collaborative activity. The second one is whether a set of the collaborative activity specific behavior rules can be computerized and bound to protocol states and protocol transitions. Some collaborative activities have no multiple states (e.g., rowing boat). They can be regarded as single state collaboration. For such collaborative activities, collaboration protocols make no sense. Some collaborative activities meet the first precondition, but don't meet the second precondition. In this case, a collaboration protocol can be used only for analysis. That is, the computational mechanisms can not be used to guide and control their interaction. For example, the playing soccer activity meets the first one, but the behavior rules can not be computerized. However, if we want to develop a collaborative computer game for playing soccer, the family of collaboration protocols for playing soccer can be developed. Furthermore, the activity-specific behavior rules have to be bound to protocol states and protocol transitions. For example, a collaboration protocol for playing a soccer game may look like a collaboration protocol for playing basketball game from the view of states and transitions. However, the behavior rules associated to states and transitions are totally different.

In this subsection, the application of this approach to support problem based learning is presented. A collaboration protocol for PBL and a sub-protocol for supporting negotiation are specified.

4.5.3.1 Modeling PBL-protocols

In this subsection, we investigate whether the idea of the collaboration protocol can be applied to support the problem based learning activity. Within the literature on problem-based learning it is clear that the problem based learning process is well structured and has a number of distinct states and transitions [Savery95] [Wolfson] [Stepien93b] [Duncan98]. In each state, expected contributions of tutor or of learners are distinguished. Furthermore, PBL-specific behavior rules can be computerized and can be bound with protocol states. This makes problem based learning an ideal application domain for collaboration protocols.

In a virtual institute, as discussed in the last chapter, all participants of a PBL activity can jointly construct a shared PBL-net synchronously or asynchronously. They can make any kind of contributions at any time (e.g., identifying a learning issue, proposing a solution, referring to a document, etc). Without coordination support, the learning process is not effective, in particular when the size of a group is large. Normally, participants have to use social protocols to coordinate their interaction. Although coordination of group interaction is most flexible with vocal agreements, prevention of violations is impossible. That is, potentially unexpected interactions and unpredictable conflicts may occur during manipulating the shared PBL-net. Therefore, the idea of collaboration protocols can be used to guide and control the social interaction for the construction of PBL-nets. The result of the application of collaboration protocol to PBL is the PBL-protocol [Miao00c]. The paragraphs below specify the PBL-protocols.

Definition (PBL-protocol Family): A *PBL-protocol family* is a collaboration protocol family for representing PBL protocols. Its value of the name attribute is 'PBL-protocol family'.

Definition (PBL-protocol): A *PBL-protocol* is a PBL-specific collaboration protocol. It can be defined and modified by using a collaboration protocol definition tool. Its value of the name attribute is the name of the PBL-protocol. Its value of the protocolFamily attribute is a collaboration family with the name of 'PBL-protocol family'. A PBL-protocol is a computational description of a PBL strategy.

Notably, different PBL-protocols can be defined depending on the size and structure of the learning group or according to the knowledge, skills, interests and learning styles of the individual members. These factors will lead to alternative strategies being adopted to perform problem-based learning. Hence, a set of PBL-protocols can be defined in the PBL-protocol family.

So far, three PBL-protocols are defined and stored in the Protocol Base in the virtual institute. In this section, we describe a PBL-protocol whose name is 'Systematic PBL protocol'. In order to simplify the description, the name or label of each entity is used to represent the entity.

Each participant of the PBL activity can be categorized into one of three protocol roles: 'learner', 'tutor', and 'expert'. The operations are defined in the last chapter such as 'create', 'remove', 'move', 'declare', and so on. The objects are typed nodes and typed links defined in the last chapter such as 'problem', 'issue', 'resource', 'hypothesis', 'solution', 'comment', (is_a_sub_problem_of $\mapsto$ 'problem' $\mapsto$ problem') ('solve' $\mapsto$ 'solution' $\mapsto$ 'problem'), and so on. A behavior rule is defined as a tuple element of the Cartesian product of these three sets. For example, ('learner' $\mapsto$ 'create' $\mapsto$ 'hypothesis') is a behavior rule in the PBL activity, and ('expert' $\mapsto$ 'create' $\mapsto$ 'resource') is another behavior rule.

As illustrated in Figure 4.11, the protocol states in the PBL-protocol are: ‘identifying problem’, ‘identifying learning issue’, ‘setting goal & making plan’, ‘learning knowledge’, ‘applying knowledge’, and ‘assessing and reflecting’. The start state of this PBL-protocol is the state ‘identifying problem’. The protocol transitions are:

(‘identifying issue’ ↦ ‘identifying problem’ ↦ ‘identifying learning issue’),
(‘setting goal’ ↦ ‘identifying learning issue’ ↦ ‘setting goal & making plan’),
(‘collecting resource’ ↦ ‘setting goal & making plan’ ↦ ‘learning knowledge’),
(‘applying’ ↦ ‘learning knowledge’ ↦ ‘applying knowledge’),
(‘analyzing problem’ ↦ ‘applying knowledge’ ↦ ‘identifying problem’), and
(‘assessing’ ↦ ‘applying knowledge’ ↦ ‘assessing and reflecting’).

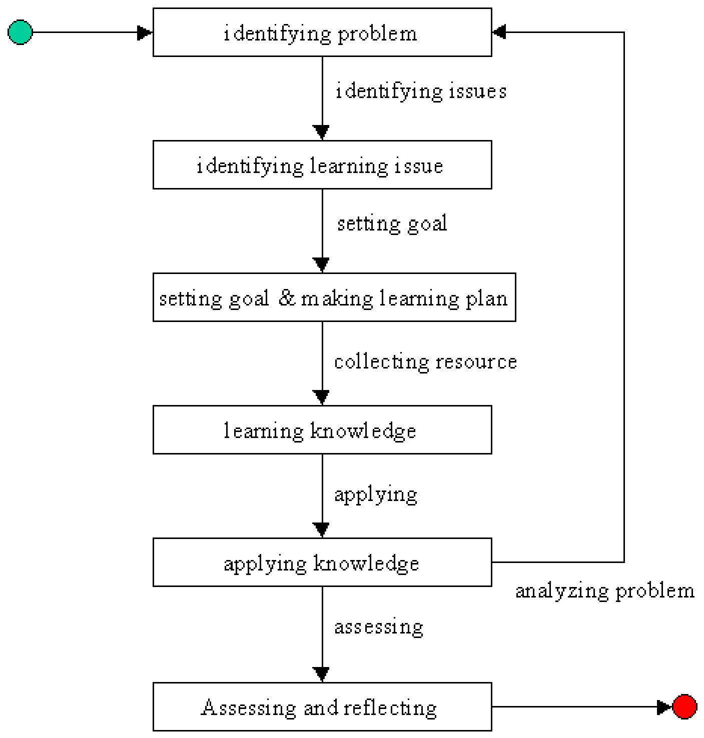


Figure 4.11: A PBL-protocol Diagram

The behavior rules can be bound to protocol states and protocol transitions. We take the ‘identifying problem’ state as an example. In the ‘identifying problem’ state, the bound behavior rules are:

(‘tutor’, ‘create’, ‘source’),
(‘learner’, ‘create’, ‘problem’),
(‘learner’, ‘create’, (is_a_sub_problem_of ↦ ‘problem’ ↦ ‘problem’)),
(‘learner’, ‘create’, (‘inform_about’ ↦ ‘source’ ↦ ‘problem’)),
(‘tutor’, ‘create’, ‘hint’),
(‘tutor’, ‘create’, ‘comment’),
(‘tutor’, ‘create’, (‘about’ ↦ ‘hint’ ↦ ‘problem’)),

('tutor', 'create', ('comment_on' ↦ 'comment' ↦ 'problem')),
('tutor', 'remove', ('inform_about' ↦ 'source' ↦ 'problem')),
('learner', 'remove', 'problem'),
('learner', 'remove', (is_a_sub_problem_of, 'problem', problem')),
('learner', 'remove', ('inform_about' ↦ 'source' ↦ 'problem')),
('tutor', 'remove', 'hint'),
('tutor', 'remove', 'comment'),
('tutor', 'remove', ('about' ↦ 'hint' ↦ 'problem')),
('tutor', 'remove', ('comment_on' ↦ 'comment' ↦ 'problem')).

From the bound behavior rules, we can observe that in PBL the role of tutor is different from a traditional teacher. The tutor can contribute to the shared PBL-net by creating or removing 'source' nodes. Only the learner can actually define the problem using 'problem' nodes and 'is_a_sub_problem_of' links in order to show how the problem decomposes into sub problems. During this state, the tutor can also manipulate 'hint' and 'comment' nodes, and 'about' and 'comment_on' links, giving indirect help in how to define the problem.

Like the 'identifying the problem' state, each protocol state and protocol transition is has a set of associated behavior rules. However, in order to save the space, the specifications of bindings to other protocol states and protocol transitions are omitted in the thesis.

Furthermore, sub-protocols can be defined under a protocol state. The sub-protocols will refine the protocol state by describing the collaboration process in more detail. In the next subsection, we discuss some sub-PBL-protocols.

4.5.3.2 Sub-PBL-protocols

As mentioned in subsection 4.5.2.1, a protocol state can be refined by defining sub-protocols under this protocol state. Each sub-protocol specifies a possible strategy that can be adopted to carry out a task under the protocol state. The use of sub-protocols provides flexibility so that learning groups can initiate them on demand. It doesn't force each learning group to execute all refined sub-processes. We present the idea of sub-protocol by using a sub-protocol that is defined under the 'applying knowledge' state.

In the 'applying knowledge' state, learners can generate hypotheses and solutions, and can support or oppose the generated hypotheses and solutions by providing evidences and principles they learned. That is, they can create corresponding typed nodes and typed links to present their ideas to others on the PBL-net. Each learner can also comment on others' ideas or declare his perspective and confidence to the created ideas. The bound behavior rules are listed as follows.

('learner', 'create', 'hypothesis'),
('learner', 'create', 'solution'),
('learner', 'create', 'evidence'),

(‘learner’, ‘create’, ‘principle’),
(‘learner’, ‘create’, ‘comment’),
(‘learner’, ‘create’, (‘support’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘support’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘create’, (‘support’ ↦ ‘evidence’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘support’ ↦ ‘evidence’ ↦ ‘solution’)),
(‘learner’, ‘create’, (‘counter’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘counter’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘create’, (‘counter’ ↦ ‘evidence’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘counter’ ↦ ‘evidence’ ↦ ‘solution’)),
(‘learner’, ‘create’, (‘solve’ ↦ ‘solution’ ↦ ‘problem’)),
(‘learner’, ‘create’, (‘based_on’ ↦ ‘solution’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘is_similar_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘is_contrary_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘is_a_prerequisite_for’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘derive_from’ ↦ principle’ ↦ ‘resource’)),
(‘learner’, ‘create’, (‘derive_from’ ↦ ‘evidence’ ↦ ‘resource’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘evidence’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘principle’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘solution’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘comment’)),
(‘learner’, ‘modify_statement’, ‘hypothesis’),
(‘learner’, ‘modify_statement’, ‘solution’),
(‘learner’, ‘modify_statement’, ‘evidence’),
(‘learner’, ‘modify_statement’, ‘principle’),
(‘learner’, ‘modify_statement’, ‘comment’),
(‘learner’, ‘remove’, ‘hypothesis’),
(‘learner’, ‘remove’, ‘solution’),
(‘learner’, ‘remove’, ‘evidence’),
(‘learner’, ‘remove’, ‘principle’),
(‘learner’, ‘remove’, ‘comment’),
(‘learner’, ‘remove’, (‘support’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘support’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘remove’, (‘support’ ↦ ‘evidence’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘support’ ↦ ‘evidence’ ↦ ‘solution’)),
(‘learner’, ‘remove’, (‘counter’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘counter’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘remove’, (‘counter’ ↦ ‘evidence’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘counter’ ↦ ‘evidence’ ↦ ‘solution’)),

(‘learner’, ‘remove’, (‘solve’ ↦ ‘solution’ ↦ ‘problem’)),
(‘learner’, ‘remove’, (‘based_on’ ↦ ‘solution’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘is_similar_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘is_contrary_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘is_a_prerequisite_for’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘derive_from’ ↦ principle’ ↦ ‘resource’)),
(‘learner’, ‘remove’, (‘derive_from’ ↦ ‘evidence’ ↦ ‘resource’)),
(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘evidence’)),
(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘principle’)),
(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘hypothesis’)),
(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘solution’)),
(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘comment’)),
(‘learner’, ‘declare’, ‘hypothesis’),
(‘learner’, ‘declare’, ‘solution’),
(‘learner’, ‘declare’, ‘evidence’),
(‘learner’, ‘declare’, ‘principle’),
(‘learner’, ‘declare’, ‘comment’),
(‘learner’, ‘declare’, (‘support’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘support’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘declare’, (‘support’ ↦ ‘evidence’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘support’ ↦ ‘evidence’ ↦ ‘solution’)),
(‘learner’, ‘declare’, (‘counter’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘counter’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘declare’, (‘counter’ ↦ ‘evidence’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘counter’ ↦ ‘evidence’ ↦ ‘solution’)),
(‘learner’, ‘declare’, (‘solve’ ↦ ‘solution’ ↦ ‘problem’)),
(‘learner’, ‘declare’, (‘derive_from’ ↦ principle’ ↦ ‘resource’)),
(‘learner’, ‘declare’, (‘derive_from’ ↦ ‘evidence’ ↦ ‘resource’)),
(‘learner’, ‘declare’, (‘based_on’ ↦ ‘solution’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘is_similar_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘is_contrary_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘is_a_prerequisite_for’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)).

In some cases, learners need to coordinate their contributions by using a sub-strategy. The sub-strategy can be defined in the same way a protocol is defined, except to specify a protocol state under which the sub-protocol can be initiated. As shown in Figure 4.12, an example strategy, called “negotiation protocol” is described. This sub-protocol consists of five sub-protocol states: ‘brainstorming’, ‘declaring’, ‘commenting’, ‘proving’, and ‘summarizing’. The protocol transitions in this sub-protocol are:

(‘declaring’↦ ‘brainstorming’ ↦ ‘declaring’)
(‘commenting’↦ ‘declaring’↦ ‘commenting’)
(‘proving’↦ ‘commenting’↦ ‘proving’)
(‘summarizing’↦ ‘proving’↦ ‘summarizing’)
(‘brainstorming again’ ↦ ‘summarizing’↦ ‘brainstorming’)
(‘declaring again’ ↦ ‘summarizing’↦ ‘declaring’).

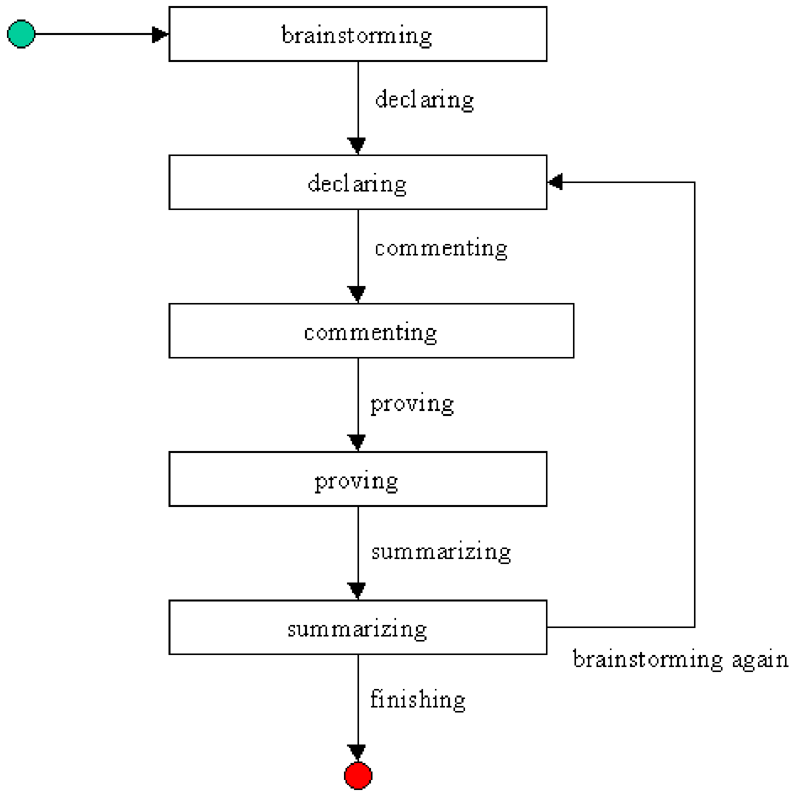


Figure 4.12: A Sub-PBL-Protocol Diagram

The bound behavior rules with each sub state are specified as below.

brainstorming:
(‘learner’, ‘create’, ‘hypothesis’),
(‘learner’, ‘create’, ‘solution’),
(‘learner’, ‘create’, (‘solve’ ↦ ‘solution’ ↦ ‘problem’)),
(‘learner’, ‘create’, (‘based_on’ ↦ ‘solution’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘is_similar_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘is_contrary_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘is_a_prerequisite_for’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘modify_statement’, ‘hypothesis’),
(‘learner’, ‘modify_statement’, ‘solution’),

(‘learner’, ‘modify_statement’, ‘evidence’),
(‘learner’, ‘modify_statement’, ‘principle’),
(‘learner’, ‘modify_statement’, ‘comment’),

declaring:
(‘learner’, ‘declare’, ‘hypothesis’),
(‘learner’, ‘declare’, ‘solution’),
(‘learner’, ‘declare’, ‘evidence’),
(‘learner’, ‘declare’, ‘principle’),
(‘learner’, ‘declare’, ‘comment’),
(‘learner’, ‘declare’, (‘support’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘support’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘declare’, (‘support’ ↦ ‘evidence’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘support’ ↦ ‘evidence’ ↦ ‘solution’)),
(‘learner’, ‘declare’, (‘counter’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘counter’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘declare’, (‘counter’ ↦ ‘evidence’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘counter’ ↦ ‘evidence’ ↦ ‘solution’)),
(‘learner’, ‘declare’, (‘solve’ ↦ ‘solution’ ↦ ‘problem’)),
(‘learner’, ‘declare’, (‘derive_from’ ↦ principle’ ↦ ‘resource’)),
(‘learner’, ‘declare’, (‘derive_from’ ↦ ‘evidence’ ↦ ‘resource’)),
(‘learner’, ‘declare’, (‘based_on’ ↦ ‘solution’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘is_similar_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘is_contrary_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),
(‘learner’, ‘declare’, (‘is_a_prerequisite_for’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),

commenting:
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘evidence’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘principle’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘solution’)),
(‘learner’, ‘create’, (‘comment_on’ ↦ ‘comment’ ↦ ‘comment’)),

proving:
(‘learner’, ‘create’, ‘hypothesis’),
(‘learner’, ‘create’, ‘solution’),
(‘learner’, ‘create’, ‘evidence’),
(‘learner’, ‘create’, ‘principle’),
(‘learner’, ‘create’, ‘comment’),
(‘learner’, ‘create’, (‘support’ ↦ ‘principle’ ↦ ‘hypothesis’)),
(‘learner’, ‘create’, (‘support’ ↦ ‘principle’ ↦ ‘solution’)),
(‘learner’, ‘create’, (‘support’ ↦ ‘evidence’ ↦ ‘hypothesis’)),

(‘learner’, ‘create’, (‘support’ ↦ ‘evidence’ ↦ ‘solution’)),

(‘learner’, ‘create’, (‘counter’ ↦ ‘principle’ ↦ ‘hypothesis’)),

(‘learner’, ‘create’, (‘counter’ ↦ ‘principle’ ↦ ‘solution’)),

(‘learner’, ‘create’, (‘counter’ ↦ ‘evidence’ ↦ ‘hypothesis’)),

(‘learner’, ‘create’, (‘counter’ ↦ ‘evidence’ ↦ ‘solution’)),

(‘learner’, ‘create’, (‘solve’ ↦ ‘solution’ ↦ ‘problem’)),

(‘learner’, ‘create’, (‘based_on’ ↦ ‘solution’ ↦ ‘hypothesis’)),

(‘learner’, ‘create’, (‘is_similar_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),

(‘learner’, ‘create’, (‘is_contrary_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),

(‘learner’, ‘create’, (‘is_a_prerequisite_for’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),

(‘learner’, ‘create’, (‘derive_from’ ↦ principle’ ↦ ‘resource’)),

(‘learner’, ‘create’, (‘derive_from’ ↦ ‘evidence’ ↦ ‘resource’)),

summarizing:
(‘learner’, ‘modify_statement’, ‘hypothesis’),
(‘learner’, ‘modify_statement’, ‘solution’),
(‘learner’, ‘modify_statement’, ‘evidence’),
(‘learner’, ‘modify_statement’, ‘principle’),
(‘learner’, ‘remove’, ‘hypothesis’),
(‘learner’, ‘remove’, ‘solution’),
(‘learner’, ‘remove’, ‘evidence’),
(‘learner’, ‘remove’, ‘principle’),
(‘learner’, ‘remove’, ‘comment’),

(‘learner’, ‘remove’, (‘support’ ↦ ‘principle’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘support’ ↦ ‘principle’ ↦ ‘solution’)),

(‘learner’, ‘remove’, (‘support’ ↦ ‘evidence’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘support’ ↦ ‘evidence’ ↦ ‘solution’)),

(‘learner’, ‘remove’, (‘counter’ ↦ ‘principle’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘counter’ ↦ ‘principle’ ↦ ‘solution’)),

(‘learner’, ‘remove’, (‘counter’ ↦ ‘evidence’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘counter’ ↦ ‘evidence’ ↦ ‘solution’)),

(‘learner’, ‘remove’, (‘solve’ ↦ ‘solution’ ↦ ‘problem’)),

(‘learner’, ‘remove’, (‘based_on’ ↦ ‘solution’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘is_similar_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘is_contrary_to’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘is_a_prerequisite_for’ ↦ ‘hypothesis’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘derive_from’ ↦ principle’ ↦ ‘resource’)),

(‘learner’, ‘remove’, (‘derive_from’ ↦ ‘evidence’ ↦ ‘resource’)),

(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘evidence’)),

(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘principle’)), (‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘hypothesis’)),

(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘solution’)),

(‘learner’, ‘remove’, (‘comment_on’ ↦ ‘comment’ ↦ ‘comment’)),

As discussed in the last chapter, learners need negotiation to pursue consensus and to construct shared knowledge. Stahl et al. [Stahl99] suggested that computational negotiation procedures could help groups to merge individuals’ results systematically into a group product. Stahl et al. developed a such negotiation procedure. In this procedure, a student can pose a personal perspective for team as a proposal. Students can select a proposal, modify it, and decide whether the new proposal should be accepted as a team perspective or not. Some negotiation rules are established in the system. For example, all new proposals must be negotiated by all team members, or alternatively the people who either originally created the proposal or subsequently modified it need to negotiate. In the same line, the sub-protocol described above was developed as a computer-mediated process to support negotiation. The negotiation protocol described above has two distinct features in comparison to the negotiation procedure described in [Stahl99]. Firstly, this protocol is suitable to support synchronous negotiation, whereas Stahl’s procedure only supports asynchronous negotiation. Secondly, it supports negotiation by allowing learners to reason, not only by simply accepting or refusing.

4.5.4 Related Work and Discussion

We compare our approach with related work from two perspectives. Firstly, we compare our collaboration protocol modeling method with state-transition-diagrams and statecharts. Secondly, we compare the PBL-protocol approach with corresponding features offered by other PBL support systems described in Chapter 3.

4.5.4.1 Comparison with State-transition-diagram and Statechart

The state-transition-diagram is a very popular method to model processes. It is easy to use and understand. However, there are two limitations such that it can not be exploited directly to model collaboration protocols. Firstly, it doesn’t provide support to bind behavior rules to states. Secondly, it is described as a flat graph that can not support nested sub-state.

Statechart supports the modelling of nested sub-state. However, in the statechart, the relation between a state and the embedded sub-state is fixed. It is impossible to execute a sub-process or not on demand. Furthermore, it doesn’t provide support to binding behavior rules with states.

The modeling method presented in this section can be regarded as an extended, modularized, hierarchical state-transition-diagram. It combines the advantage of state-transition-diagram and statechart, and overcomes their limitations. This method provides a flexibility for learners to execute a sub-process or not on demand. In addition, binding behavior rules is a special requirement that state-transition-diagram and statechart don’t care about. Behavior rules allow a role-based state-dependent specification of access control, which is required by multiple-state collaboration processes.

4.5.4.2 Comparison with PBL Support Systems

Other PBL support systems do not deal with the question of how to support users to behave within their predefined role types through the different states of the problem based learning process. Collaboratory Notebook [O'Neill94], McBAGEL [Huebscher96], CALE [Mahling95] and CSILE [Scardamalia94] are systems which store the contributions of all users, whether teachers or learners, in a central database. Any user can contribute to the database at any time, and can retrieve the contributions of others at any time in order to read them. Belvedere [Suthers95], McBAGEL [Huebscher96], and Web-SMILE [Guzdial97] distinguish between the different states of the problem based learning process, but does not restrict user operations according to their role as learner, tutor or expert. It doesn't support execution of processes.

4.5.5 Summary

In this section, the theoretical background for the use of collaboration protocols to support collaborative processes was described. The collaboration protocol provided a role-based, state-dependent access control mechanism. The approach to model and execute of collaboration protocols was formally specified. In addition, a systematic method to shift between collaboration protocols was presented.

The approach was applied to support users of the PBL method within virtual learning environments. The PBL-protocol was developed to help learners to overcome two key difficulties. Firstly, neither learners nor tutors, who are used to more traditional methods of teaching, know exactly what to expect from PBL or how to behave appropriately within their new roles. Secondly, these problems are compounded by having to interact within a socially unfamiliar computer-based learning environment. The resulting PBL protocols restrict behavior to fit within pre-defined roles, and to guide users from one state of the PBL process to the next. In addition, the idea of sub-protocol provides flexibility for learners to execute refined sub-processes on demand. A negotiation protocol was described, which can be used to support negotiation processes for construction of shared knowledge.

4.6 PBL-plan: Coordinating Actions in PBL Processes

In traditional education, the teacher and the institution structure the learning activity. The learner is told what objective to work toward, what resources are to be used and how (and when) to use them, and how any accomplishment of the objectives will be evaluated. In PBL, learners are responsible for setting learning goals and making learning plans. That is, they have to identify the important issues in the problem, to identify current gaps in knowledge or understanding, to consider time constrains, learning resources, and objectives, to set priorities regarding the relative importance of each learning issue, to identify prerequisites for researching a learning issue, and to arrange actions and divide labor to research the identified learning issues, and to evaluate whether each learning goal is achieved. The identification of learning issues will reflect the biases and individual characteristics of each learning group.

As mentioned in chapter 2, in order to support such a self-directed learning process, a virtual PBL environment should provide mechanisms to help learners to structure the learning activity and coordinate learning actions. This chapter begins with a brief introduction of the theory of self-directed learning. Based on self-directed learning theory, an approach to model and execute self-directed learning processes is developed in order to support PBL processes in virtual learning environments. The main body of this subsection describes the approach to help learners to set learning goals, to create, monitor, and refine their learning plan, and to coordinate their learning actions by automatic execution of the learning plan. After a comparison with workflow management systems, this chapter ends with a summary.

4.6.1 Theoretical Background

The theoretical background of the research work described in this section is based on the theory of self-directed learning. Self-directed learning has existed even from classical antiquity. However, it is during the last three decades that self-directed learning has become a major research area. Knowles, as the founder of this theory, attempted to develop a theory specifically for adult learning. As Hiemstra [Hiemstra94] wrote: Knowles' publication [Knowles75] "… provided foundational definitions and assumptions that guided much subsequent research: (a) self-directed learning assumes that humans grow in capacity and need to be self-directing; (b) learners' experiences are rich resources for learning; (c) individuals learn what is required to perform their evolving life tasks; (d) an adult's natural orientation is task or problem-centered learning; (e) self-directed learners are motivated by various internal incentives, such as need for self-esteem, curiosity, desire to achieve, and satisfaction of accomplishment." Brockett and Hiemstra [Brockett91] view the term self-directed learning as "an instructional process centering on the assessing needs, securing learning resources, planning, implementing learning activities, and evaluating learning where learners assume primary responsibility for the process." Hiemstra and Sisco [Hiemstra90] have presented an approach for individualizing instruction derived from principles of humanism and designed specifically for working with adult learners. The individualizing instruction process model consists of six steps, which are related to each other in a circular rather than linear sequence. The six steps are: "(a) activities prior to the first session (e.g., developing a rationale, preplanning); (b) creating a positive learning environment (physical, social, and psychological); (c) developing the instructional plan (with active involvement of participants in assessing personal and relevant group needs, ascertaining the relevance of past experience, and prioritizing knowledge areas to be covered); (d) identifying the learning activities (determining learning activities and techniques); (e) putting learning into action and monitoring progress (formative evaluation); and (f) evaluating individual learning outcomes (matching learning objectives to mastery)."

The work of Knowles [Knowles80] [Knowles84] has resulted in a need by many teachers of adults to provide some mechanism for learners to build on past experience and determined needs as they carry out learning activities. The use of learning contracts with adult learners has gained cogency during the past decade. The learning contract is a device that provides a vehicle for making the planning of learning experiences a mutual undertaking between learners and facilitators. Most contracts contain information on the learning goals, anticipated learning resources and strategies, a projected time line, and ideas for how to evaluate or validate the learning achievements [Knowles86]. According to [LEARNING CONTRACTS], how to complete and utilize a learning contract can be summarized as below:

"***Diagnose learning needs***. A learning need is the gap between where you are now and where you want to be in regard to a particular set of competencies. You may already be aware of certain learning needs as a result of a personal appraisal or the long accumulation of evidence for yourself regarding any gaps between where you are now and where you would like to be.

Specify learning objectives. Each of the learning needs should be translated into a learning objective. Be sure that your objectives describe what you will learn, not what you will do. State them in terms that are most meaningful to you--Content acquisition, terminal behaviors, or direction of growth.

Specify learning resources and strategies. Identify the resources (material and human) you plan to use in your various learning experiences and the strategies (techniques, tools) you will employ in making use of them.

Specify target dates for completion. Put realistic dates, unless there are institutionally or other required deadlines.

Specify evidence of accomplishment. Describe what evidence you will collect to indicate the degree to which you have achieved each objective.

Specify how the evidence will be validated. For each objective, first specify the criteria by which you propose the evidence will be judged. After you have specified the criteria, indicate the means you propose for verifying the evidence according to these criteria. For example, if you produce a paper, who will read it and what are their qualifications?

Review the contract with consultants. After you have completed the first draft of your contract, you will find it useful to review it with two or three friends, supervisors, or other expert resource people to obtain their reaction and suggestions.

Carry out the contract. You now simply do what the contract calls for. But keep in mind that as you work on it you may find that your notions about what you want to learn and how you want to learn are changing. So don't hesitate to revise or renegotiate your contract as you go along.

Evaluate the learning. When you have completed your contract you will want to get some assurance that you have in fact learned what you set out to learn."

The concept of the learning contracts provides the guideline to develop a computational mechanism for representing and executing learning plans in virtual PBL environments.

4.6.2 PBL-plan

Through an analysis of the example scenario described in chapter 2 according to the theory of self-directed learning, the characteristics of PBL processes at the action level can be identified.

Firstly, the overall collaborative learning process encompasses multiple actions and each action takes place in a certain place. The PBL group often divides the labor and normally multiple learners are responsible for performing an action. Some actions are performed in synchronous sessions and others are carried out in asynchronous sessions. In collaborative learning processes, artifacts such as learning materials and reports are collected or constructed jointly.

Secondly, the actions are executed sequentially, concurrently, or in parallel. For example, some learning actions have to be performed after the prerequisite knowledge is acquired. Learning knowledge of different topics can be carried out in parallel. The actions (e.g., searching on the Web and inquiring experts) to collect the information on the same topic can be performed concurrently. The difference between concurrent sessions and parallel sessions is that the former deal with the same artifact at the same time and the later need not. Some artifacts produced in one session will be consumed in other sessions, and the delivery of artifacts in one action may results in the starting of other actions that consume the delivered artifacts. Some actions that are performed in the synchronous sessions will start at scheduled time or when all participants of the actions join the actions.

Thirdly, performing a PBL activity often is a long-term task (ranging from a couple of days to a semester). A learning group has to work out a detailed project plan to help communication, understanding, and coordination among group members, rather than teachers or institutions predefine the learning plan. Each learning group has a unique learning plan. In addition, the main part of a learning plan is defined during the learning process as a collective effort.

Fourthly, the learning group executes the PBL activity according the definition of the learning plan. However, it is impossible to define a correct and detailed project plan in advance, which then would be executed exactly. That is, collaboration processes are a kind of ad hoc processes. Some details of the learning plan are defined during the execution of the learning process and the learning plan has to be revised in the progress to fit some changes.

In order to support collaborative processes with the characteristics described above in a computer-based system, the concept of *session-based collaborative processes* has been developed [Miao98a] [Miao99a]. The term *session* is defined as that a process is executed in a synchronous or asynchronous collaboration mode on a shared workspace by a group of people to achieve a goal. The notion of a *session-based collaboration process* denotes the whole work process that consists of a coordinated set of sessions. An approach to support session-based collaboration processes in computer-based collaboration environments is proposed in [Miao98a] [Miao99a]. The proposed approach is characterized by providing:

1) a visual process modeling language for describing a session-based collaborative process,
2) a collaborative tool to support definition of session-based collaborative processes as a hypertext document, and
3) a cooperative environment to execute session-based collaborative processes by a team and to enact the sessions and provide shared data and shared views.

This approach is further developed in this thesis to fit self-directed learning processes and is then applied to support PBL activities in virtual learning environments. By using this approach, a problem based learning process can be described as a hypertext document, called a PBL-plan [Miao00d].

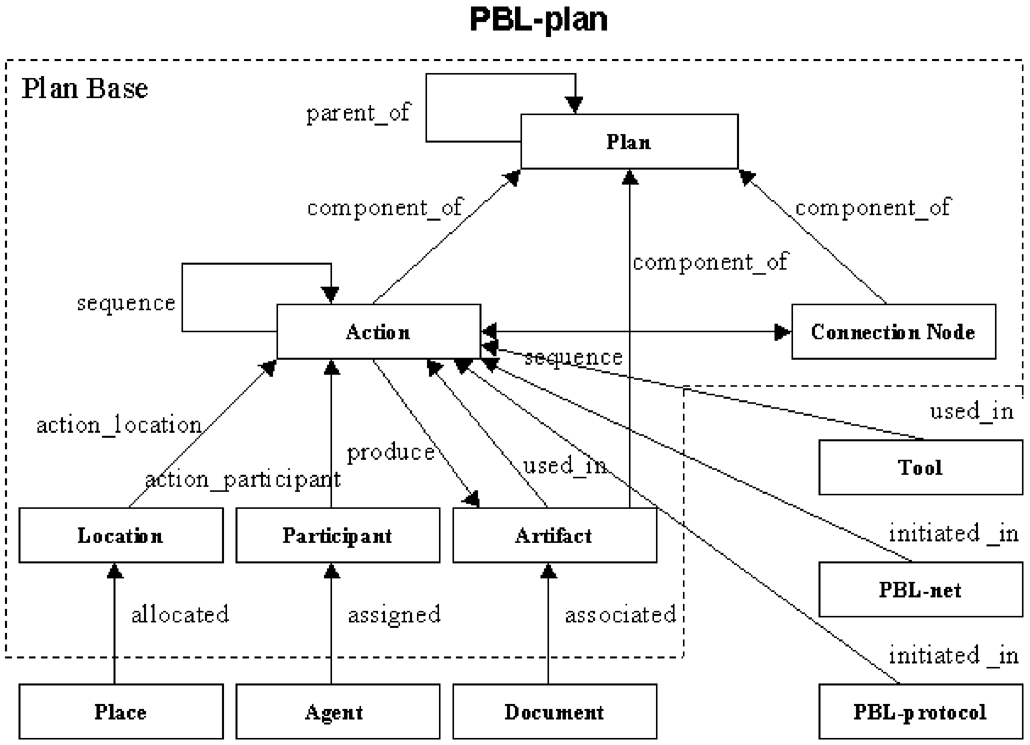


Figure 4.13: Conceptual Architecture of the PBL-plan

As illustrated in Figure 4.13, a PBL-plan consists of a set of action nodes, a set of connection nodes, and a set of artifact nodes. A PBL-plan may have a sub-plan. Action nodes and connection nodes can be connected in sequence. A place should be allocated as a location for each action. One or more agents should be assigned to each action as the participants of the action. An action may use and produce artifacts, which refer to documents in the virtual institute. Tools may be used in actions. A PBL-net and a PBL-protocol can be initiated in actions. Rather than a representation of learning procedure, the PBL-plan can be executed automatically to coordinate actions carried out by learners or sub-groups at same/different time and in same/different virtual places. This subsection describes the PBL-plan formally.

Definition (PBL-plan): A *PBL-plan* represents a whole or a part of a problem based learning process. Each PBL-plan has a name.

```
┌─ PBLPlan ──────────────────────────────
│ name : STRING
└────────────────────────────────────────
```

Definition (Plans): *Plans* represent a set of learning plans. Each learning plan has a state. The possible states are *created, defined, active,* and *finished.*

PlanState ::= created | defined | active | finished

```
┌─ Plans ────────────────────────────────
│ plans : ℙ PBLPlan
│ currentPlanState : PBLPlan → PlanState
├──────────────────────────
│ dom currentPlanState = plans
└────────────────────────────────────────
```

Definition (Action): An *action* represents a scheduled and executable learning task. Each action has a name and a goal. When scheduling an action, it is needed to specify when and how long the action will be carried out. It is needed to specify the collaboration mode used to perform the action. A synchronous session requires that all participants are present when the action is carried out, and an asynchronous session allows that participants contribute at different times. It is also needed to specify the active-condition under which the action can be started and the terminated-condition under which the action can be terminated. The active-condition and terminated-condition of an action are specified by predicates. A predicate is represented as a logical expression that is a combination of logical operations on the elementary logical statements. Examples of elementary logical statements are: whether the scheduled start time has arrived, whether all participants of the action have joined the action, whether the necessary documents of the action are available, whether the produced documents of the action are finished, whether the preceding actions are terminated, and so on. The actual start time of an action is used to record the point of time when the action actually starts.

Condition = = Predicate

Duration = = $\mathbb{N}$
Mode :: = synchronousSession | asynchronousSession

```
┌─ Action ───────────────────────────────
│ name : STRING
│ goal : STRING
│ scheduledStartTime : TIME
│ estimatedDuration : Duration
│ collaborationMode : Mode
│ actionActiveCondition, actionTerminateCondition : Condition
│ actualStartTime : TIME
└────────────────────────────────────────
```

Definition (Actions): *Actions* represents a set of learning actions. Some learning actions are defined as a step of a learning plan. Others can be defined as independent actions from learning plans. Each action has a state and the possible states are *created, defined, enabled, active, suspended,* and *finished*. An action is assigned to some agents who are responsible for carrying out the action. A place is assigned as a location for an action where the action is carried out.

ActionState :: = created | defined | enabled | active | suspended | finished

Actions

VirtualInstitute

Plans

actions : ℙ Action

actionInPlan : Action → PBLPlan

currentActionState : Action → ActionState

actionParticipants : Action ↔ Agent

actionLocation : Action → Place

dom actionInPlan ⊆ actions ∧ **ran** actionInPlan ⊆ plans

dom currentActionState = actions

dom actionParticipants ⊆ actions ∧ **ran** actionParticipants ⊆ agents

dom actionLocation ⊆ actions ∧ **ran** actionLocation ⊂ places

Definition (Artifact): An *artifact* represents a document reference in a PBL-plan. Each artifact has a name and a refered document.

Artifact

name : STRING

referTo : Document

Definition (Artifacts): *Artifacts* represent a set of artifacts, which is defined in a plan and is used, shared, or produced in one or more actions. Each artifact has an attribute to represent the current state of the artifact. The possible states are *created, inEditing,* and *finished*.

ArtifactState :: = created | inEditing | finished

Artifacts

Plans

Actions

artifacts: ℙ Artifact

artifactInPlan : Artifact → PBLPlan

currentArtifactState : Artifact → ArtifactState

actionProduceArtifact : Action ⇸ Artifact

artifactConsumedByAction : Artifact ⇸ Action

actionSharedArtifact : Action ⇸ Artifact

$\mathbf{dom}\ \text{artifactInPlan} = \text{artifacts} \wedge \mathbf{ran}\ \text{artifactInPlan} \subseteq \text{plans}$
$\mathbf{dom}\ \text{currentArtifactState} = \text{artifacts}$
$\mathbf{dom}\ \text{actionProduceArtifact} \subseteq \text{actions} \wedge \mathbf{ran}\ \text{actionProduceArtifact} \subseteq \text{artifacts}$
$\mathbf{dom}\ \text{artifactConsumedByAction} \subseteq \text{artifacts}$
$\mathbf{ran}\ \text{artifactConsumedByAction} \subseteq \text{actions}$
$\mathbf{dom}\ \text{actionSharedArtifact} \subseteq \text{actions} \wedge \mathbf{ran}\ \text{actionSharedArtifact} \subseteq \text{artifacts}$

Definition (Connection Nodes): *Connection Nodes* represent special check points in learning plans. There are six types of connection nodes: *StartPoint, EndPoint, AndJoin, OrJoin, AndSplit*, and *OrSplit.* StartPoint is used to trigger actions that are connected to the StartPoint when a plan starts to execute. EndPoint is used to terminate a plan. A plan can have only one StartPoint and one EndPoint. Split nodes are used to fork actions in a way that either all subsequent actions will be executed in parallel or one of them will be executed and others will be suspended. Split nodes are used to specify when the execution can continue: either all of the preceding actions need to be finished or at least one of them needs to be finished. Join nodes provide a simple mechanism to synchronize actions that specifies when the execution can continue: either all of the preceding actions need to be finished or at least one of them needs to be finished. The detailed semantics of these data types will be specified in section 4.6.4.

[StartPoint, EndPoint, AndJoin, OrJoin, AndSplit, OrSplit]

ConnectionNodes
Plans
$\text{planStartPoint} : \text{PBLPlan} \rightarrowtail \text{StartPoint}$
$\text{planEndPoint} : \text{PBLPlan} \rightarrowtail \text{EndPoint}$
$\text{andJoinInPlan} : \text{AndJoin} \rightarrow \text{PBLPlan}$
$\text{orJoinInPlan} : \text{OrJoin} \rightarrow \text{PBLPlan}$
$\text{andSplitInPlan} : \text{AndSplit} \rightarrow \text{PBLPlan}$
$\text{orSplitInPlan} : \text{OrSplit} \rightarrow \text{PBLPlan}$

$\mathbf{dom}\ \text{planStartPoint} \subseteq \text{plans} \wedge \mathbf{dom}\ \text{planEndPoint} \subseteq \text{plans}$
$\mathbf{ran}\ \text{andJoinInPlan} \subseteq \text{plans} \wedge \mathbf{ran}\ \text{orJoinInPlan} \subseteq \text{plans}$
$\mathbf{ran}\ \text{andSplitInPlan} \subseteq \text{plans} \wedge \mathbf{ran}\ \text{orSplitInPlan} \subseteq \text{plans}$

Definition (Temporal Relations): *Temporal Relations* are used to represent the temporal relations between plans, actions, and connection nodes in plans. It is important to note that a plan can be embeded in another plan as a sub-plan, but it is not allowed to form a loop when defining the relation "parentPlan". For all relations except for the relation "parentPlan" specified in this abstract state, if a pair of elements belongs to one of these relations, they should be defined in the same plan. Each action and each plan can connect to at maximum one of action, plan, or a type of connection nodes and can be connected at maximum to one of action, plan, or a type of connection nodes. Each startPoint can connect to at maximum one of action, plan, or a type of other connection nodes. Each endPoint can connect to at maximum one of action, plan, and a type of other connection nodes. Each AndSplit and each OrSplit can be connected at maximum to one of action, plan, or a type of other connection nodes. Each AndJoin and each OrJoin can connect to at maximum one of action, plan, or a type of other connection nodes. Note that in a well-defined learning plan, each action should be exactly one destination and one source of temporal relationships. However, during definition, it is possible that no temporal relationship is defined for an action. Therefore, as specified by a predicate in the following schema, the number of incoming temporal relationship of an action node is either zero or one, and the number of outgoing temporal relationship of an action node is either zero or one as well. Similar constraints apply to a plan node.

TemporalRelations

Plans
Actions
ConnectionNodes
parentPlan : PBLPlan ⇸ PBLPlan
startPointToAction : StartPoint ⤔ Action
startPointToPlan : StartPoint ⤔ PBLPlan
actionToEndPoint : Action ⤔ EndPoint
planToEndPoint : PBLPlan ⤔ EndPoint
actionSequence : Action ⤔ Action
planToPlan : PBLPlan ⤔ PBLPlan
actionToPlan : Action ⤔ PBLPlan
planToAction : PBLPlan ⤔ Action
actionToAndJoin : Action ⇸ AndJoin
actionToOrJoin : Action ⇸ OrJoin
actionToAndSplit : Action ⤔ AndSplit
actionToOrSplit : Action ⤔ OrSplit
andJoinToAction : AndJoin ⤔ Action
orJoinToAction : OrJoin ⤔ Action
andSplitToAction : AndSplit ⇸ Action
orSplitToAction : OrSplit ⇸ Action
planToAndJoin : PBLPlan ⇸ AndJoin
planToOrJoin : PBLPlan ⇸ OrJoin
planToAndSplit : PBLPlan ⤔ AndSplit
planToOrSplit : PBLPlan ⤔ OrSplit
andJoinToPlan : AndJoin ⤔ PBLPlan
orJoinToPlan : OrJoin ⤔ PBLPlan
andSplitToPlan : AndSplit ⇸ PBLPlan
orSplitToPlan : OrSplit ⇸ PBLPlan
startPointToOrJoin : StartPoint ⤔ OrJoin
startPointToAndSplit : StartPoint ⤔ AndSplit
startPointToOrSplit : StartPoint ⤔ OrSplit
andJoinToEndPoint : AndJoin ⤔ EndPoint
orJoinToEndPoint : OrJoin ⤔ EndPoint

disjoint < parentPlan $^{+}$, **id** PBLPlan >

(∀ sa : startPointToAction | (∃ p : PBLPlan | p ∈ plans ∧

$(p \mapsto \mathbf{first}\ sa) \in planStartPoint \land actionInPlan\ (\mathbf{second}\ sa) = p\ ))$

$(\forall\ sp : startPointToPlan \mid (\ \exists\ p : PBLPlan \mid p \in plans \land$
$(p \mapsto \mathbf{first}\ sp) \in planStartPoint \land (\mathbf{second}\ sp \mapsto p\ ) \in parentPlan)$

$(\forall\ ae : actionToEndPoint \mid (\ \exists\ p : PBLPlan \mid p \in plans \land$
$(p \mapsto \mathbf{second}\ ae) \in planEndPoint \land actionInPlan\ (\mathbf{first}\ ae) = p\ ))$

$(\forall\ pe : planToEndPoint \mid (\ \exists\ p : PBLPlan \mid p \in plans \land$
$(p \mapsto \mathbf{second}\ pe) \in planEndPoint \land (\mathbf{first}\ pe \mapsto p\ ) \in parentPlan)$

$(\forall\ a_1, a_2\ : Action \mid (a_1 \mapsto a_2) \in actionSequence \Rightarrow$
$actionInPlan\ a_1 = actionInPlan\ a_2\ )$
$(\forall\ a : Action \mid (\ \#(\ \mathbf{dom}\ startPointToAction \rhd \{a\}) +$
$\#(\ \mathbf{dom}\ actionSequence \rhd \{a\})\ + \#(\ \mathbf{dom}\ planToAction \rhd \{a\}) +$
$\#(\ \mathbf{dom}\ andJoinToAction \rhd \{a\})\ + \#(\ \mathbf{dom}\ orJoinToAction \rhd \{a\}) +$
$\#(\ \mathbf{dom}\ andSplitToAction \rhd \{a\})\ + \#(\ \mathbf{dom}\ orSplitToAction \rhd \{a\}) \leq 1) \land$
$(\#(\ \mathbf{ran}\ \{a\} \lhd\ actionToEndPoint) +$
$\#(\ \mathbf{ran}\ \{a\} \lhd\ actionSequence) + \#(\ \mathbf{ran}\ \{a\} \lhd\ actionToPlan) +$
$\#(\ \mathbf{ran}\ \{a\} \lhd\ actionToAndJoin) + \#(\ \mathbf{ran}\ \{a\} \lhd\ actionToOrJoin) +$
$\#(\ \mathbf{ran}\ \{a\} \lhd\ actionToAndSplit) + \#(\ \mathbf{ran}\ \{a\} \lhd\ actionToOrSplit) \leq 1)$

$(\forall\ a : Action;\ p : PBLPlan \mid (a \mapsto p) : actionToPlan\ \Rightarrow$
$actionInPlan\ a = parentPlan\ p\ )$
$(\forall\ a : Action;\ p : PBLPlan \mid (p \mapsto a) : planToAction\ \Rightarrow$
$actionInPlan\ a = parentPlan\ p\ )$
$(\forall\ a : Action;\ andJoin : AndJoin \mid (a \mapsto andJoin\ ) \in actionToAndJoin \Rightarrow$
$actionInPlan\ a = andJoinInPlan\ andJoin)$
$(\forall\ a : Action;\ orJoin : OrJoin \mid (a \mapsto orJoin\ ) \in actionToOrJoin \Rightarrow$
$actionInPlan\ a = orJoinInPlan\ orJoin)$
$(\forall\ a : Action;\ andSplit : AndSplit \mid (a \mapsto andSplit\ ) \in actionToAndSplit \Rightarrow$
$actionInPlan\ a = andSplitInPlan\ andSplit)$
$(\forall\ a : Action;\ orSplit : OrSplit \mid (a \mapsto orSplit\ ) \in actionToOrSplit \Rightarrow$
$actionInPlan\ a = orSplitInPlan\ orSplit)$
$(\forall\ a : Action;\ andJoin : AndJoin \mid (andJoin \mapsto a\ ) \in andJoinToAction \Rightarrow$
$actionInPlan\ a = andJoinInPlan\ andJoin\ )$
$(\forall\ a : Action;\ orJoin : OrJoin \mid (orJoin \mapsto a\ ) \in orJoinToAction \Rightarrow$
$actionInPlan\ a = orJoinInPlan\ orJoin)$
$(\forall\ a : Action;\ andSplit : AndSplit \mid (andSplit \mapsto a\ ) \in andSplitToAction \Rightarrow$
$actionInPlan\ a = andSplitInPlan\ andSplit)$
$(\forall\ a : Action;\ orSplit : OrSplit \mid (orSplit \mapsto a\ ) \in orSplitToAction \Rightarrow$

$\quad actionInPlan\ a = orSplitInPlan\ orSplit)$

$(\forall\ ae : andJoinToEndPoint \mid (\exists\ p : PBLPlan \mid p \in plans \land$

$\quad (p \mapsto \mathbf{second}\ ae) \in planEndPoint \land andJoinInPlan\ (\mathbf{first}\ ae) = p\ ))$

$(\forall\ oe : orJoinToEndPoint \mid (\exists\ p : PBLPlan \mid p \in plans \land$

$\quad (p \mapsto \mathbf{second}\ oe) \in planEndPoint \land orJoinInPlan\ (\mathbf{first}\ oe) = p\ ))$

$(\forall\ p_1, p_2 : PBLPlan \mid (p_1 \mapsto p_2) \in planToPlan \Rightarrow parentPlan\ p_1 = parentPlan\ p_2)$

$(\forall\ p : PBLPlan \mid (\#(\mathbf{dom}\ planToPlan \rhd \{p\}) +$

$\quad \#(\mathbf{dom}\ startPointToPlan \rhd \{p\}) + \#(\mathbf{dom}\ actionToPlan \rhd \{p\}) +$

$\quad \#(\mathbf{dom}\ andJoinToPlan \rhd \{p\}) + \#(\mathbf{dom}\ orJoinToPlan \rhd \{p\}) +$

$\quad \#(\mathbf{dom}\ andSplitToPlan \rhd \{p\}) + \#(\mathbf{dom}\ orSplitToPlan \rhd \{p\}) \leq 1) \land$

$\quad (\#(\mathbf{ran}\ \{p\} \lhd planToPlan) +$

$\quad \#(\mathbf{ran}\ \{p\} \lhd planToEndPoint) + \#(\mathbf{ran}\ \{p\} \lhd planToAction) +$

$\quad \#(\mathbf{ran}\ \{p\} \lhd planToAndJoin) + \#(\mathbf{ran}\ \{p\} \lhd planToOrJoin) +$

$\quad \#(\mathbf{ran}\ \{p\} \lhd planToAndSplit) + \#(\mathbf{ran}\ \{p\} \lhd planToOrSplit) \leq 1)$

$(\forall\ p : PBLPlan;\ andJoin : AndJoin \mid (p \mapsto andJoin) \in planToAndJoin \Rightarrow$
$\quad parentPlan\ p = andJoinInPlan\ andJoin)$

$(\forall\ p : PBLPlan;\ orJoin : OrJoin \mid (p \mapsto orJoin) \in planToOrJoin \Rightarrow$
$\quad parentPlan\ p = orJoinInPlan\ orJoin)$

$(\forall\ p : PBLPlan;\ andSplit : AndSplit \mid (p \mapsto andSplit) \in planToAndSplit \Rightarrow$
$\quad parentPlan\ p = andSplitInPlan\ andSplit)$

$(\forall\ p : PBLPlan;\ orSplit : OrSplit \mid (p \mapsto orSplit) \in planToOrSplit \Rightarrow$
$\quad parentPlan\ p = orSplitInPlan\ orSplit)$

$(\forall\ p : PBLPlan;\ andJoin : AndJoin \mid (andJoin \mapsto p) \in andJoinToPlan \Rightarrow$
$\quad parentPlan\ p = andJoinInPlan\ andJoin)$

$(\forall\ p : PBLPlan;\ orJoin : OrJoin \mid (orJoin \mapsto p) \in orJoinToPlan \Rightarrow$
$\quad parentPlan\ p = orJoinInPlan\ orJoin)$

$(\forall\ p : PBLPlan;\ andSplit : AndSplit \mid (andSplit \mapsto p) \in andSplitToPlan \Rightarrow$
$\quad parentPlan\ p = andSplitInPlan\ andSplit)$

$(\forall\ p : PBLPlan;\ orSplit : OrSplit \mid (orSplit \mapsto p) \in orSplitToPlan \Rightarrow$
$\quad parentPlan\ p = orSplitInPlan\ orSplit)$

$(\forall\ n : StartPoint \mid \#(\mathbf{ran}\ \{n\} \lhd startPointToAction) +$

$\quad \#(\mathbf{ran}\ \{n\} \lhd startPointToPlan) + \#(\mathbf{ran}\ \{n\} \lhd startPointToOrJoin) +$

$\quad \#(\mathbf{ran}\ \{n\} \lhd startPointToAndSplit) + \#(\mathbf{ran}\ \{n\} \lhd startPointToOrSplit) \leq 1$

$(\forall\ n : EndPoint \mid$

$\quad \#(\mathbf{dom}\ actionToEndPoint \rhd \{n\}) + \#(\mathbf{dom}\ planToEndPoint \rhd \{n\}) +$

$\quad \#(\mathbf{dom}\ andJoinToEndPoint \rhd \{n\}) + \#(\mathbf{dom}\ orJoinToEndPoint \rhd \{n\}) \leq 1$

$(\forall\ n : AndSplit \mid \#(\mathbf{dom}\ startPointToAndSplit \rhd \{n\}) +$

$\quad \#(\mathbf{dom}\ actionToAndSplit \rhd \{n\}) + \#(\mathbf{dom}\ planToAndSplit \rhd \{n\}) \leq 1$

$(\forall$ n : OrSplit | #(**dom** startPointToOrSplit $\rhd$ {n}) +
 #(**dom** actionToOrSplit $\rhd$ {n}) + #(**dom** planToOrSplit $\rhd$ {n}) ≤ 1
$(\forall$ n : AndJoin | #(**ran** {n} $\lhd$ andJoinToPlan) +
 #(**ran** {n} $\lhd$ andJoinToAction) + #(**ran** {n} $\lhd$ andJoinToEndPoint) ≤ 1
$(\forall$ n : OrJoin | #(**ran** {n} $\lhd$ orJoinToPlan) +
 #(**ran** {n} $\lhd$ orJoinToAction) + #(**ran** {n} $\lhd$ orJoinToEndPoint) ≤ 1

Definition (Plan Base): A *plan base* represents a set of defined plans in which the actions, artifacts, connection nodes, and their relations are specified.

PlanBase
Plans
Actions
Artifacts
ConnectionNodes
TemporalRelations

So far, the data types and abstract state of the PBL-plan have been specified. Now, the operations to define PBL-plans are specified below.

A plan can be created as a root plan or as a sub-plan of another plan. Users can create a root plan by assigning a name to the plan or create a sub-plan by assigning a name to the currently created plan and specifying the parent plan.

CreateRootPlanOK
Δ Plans
name? : STRING

$(\forall$ p : PBLPlan | p $\in$ plans $\bullet$ p.name $\neq$ name?)

let aPlan = = (μ PBLPlan | name = name?) $\bullet$
 plans' = plans $\cup$ { aPlan } $\wedge$
 currentPlanState' = currentPlanState $\cup$ { aPlan $\mapsto$ created? }

ΔChangeSubPlan $\triangleq$ ΞTemporalRelations \ (parentPlan, parentPlan') $\wedge$ **Δ**TemporalRelations

Here, hiding a particular before and after component in ΞTemporalRelations \ (parentPlan, parentPlan') gives a before and after state that does not have parentPlan and parentPlan', but still has all the other components, unchanged. Conjoining this with **Δ**TemporalRelations reintroduces the declaration of have parentPlan and parentPlan', and any predicate involving them, but does not include the predicate parentPlan = parentPlan'. Hence ΞTemporalRelations \ (parentPlan, parentPlan') $\wedge$ **Δ**TemporalRelations is a schema describing a before and after state of

TemporalRelations that includes all the constructs on TemporalRelations and TemporalRelations', and in addition has all the components, except parentPlan, unchanged.

CreateSubPlanOK

Δ Plans
Δ ChangeSubPlan
name? : STRING
plan? : PBLPlan

plan? $\in$ plans $\wedge$ ($\forall$ p : PBLPlan | p $\in$ plans $\bullet$ p.name $\neq$ name?)

let aPlan = = (μ PBLPlan | name = name?) $\bullet$
plans' = plans $\cup$ { aPlan } $\wedge$
currentPlanState' = currentPlanState $\cup$ { aPlan $\mapsto$ created? } $\wedge$
parentPlan' = parentPlan $\cup$ { aPlan $\mapsto$ plan? }

The operations to create and define an action are specified as follows. The operation to modify the definition of an action is similar to the operation "DefineActionOK". Therefore, the specification of this operation is omitted.

CreateActionOK

Δ Actions
Ξ Plans
Ξ VirtualInstitute
name? : STRING
aPlan?: PBLPlan

($\forall$ a : Action | a $\in$ actions $\bullet$ a.name $\neq$ name?) $\wedge$ aPlan $\in$ plans

anAction = = (μ Action | name = name? $\wedge$ goal = $\emptyset$ $\wedge$
scheduledStartTime = $\emptyset$ $\wedge$ estimatedDuration = $\emptyset$ $\wedge$
actualStartTime = $\emptyset$ $\wedge$ collaborationMode = $\emptyset$ $\wedge$
actionActiveCondition = $\emptyset$ $\wedge$ actionTerminateCondition = $\emptyset$)
actions' = actions $\cup$ { anAction }
currentActionState' = currentActionState $\cup$ { anAction $\mapsto$ created }
actionInPlan' = actionInPlan $\cup$ { anAction $\mapsto$ aPlan? }
actionParticipants' = actionParticipants
actionLocation' = actionLocation

DefineActionOK

Δ Actions
Ξ Plans
Ξ VirtualInstitute
a? : Action
goal?: STRING

startTime? : TIME
duration? : Duration
activeCondition?, terminateCondition? : Condition

$a? \in actions \wedge currentActionState\ a? = created$

$(\boldsymbol{\theta}\ a?' \mid name = a?.name \wedge goal = goal? \wedge$
$\quad scheduledStartTime = startTime? \wedge$
$\quad estimatedDuration = duration? \wedge$
$\quad collaborationMode = aMode? \wedge$
$\quad actionActiveCondition = activeCondition? \wedge$
$\quad actionTerminateCondition = terminateCondition?\ )$
$actions' = actions$
$currentActionState' = currentActionState \oplus \{\ a? \mapsto defined\ \}$
$actionInPlan' = actionInPlan$
$actionParticipants' = actionParticipants$
$actionLocation' = actionLocation$

The operations to assign participants and allocate a place for an action are specified as follows.

AssignActionParticipantOK

Δ Actions
Ξ Plans
Ξ VirtualInstitute
a? : Action
agent?: Agent

$a? \in actions \wedge agent? = agents \wedge (a? \mapsto agent?) \notin actionParticipants$

$actions' = actions$
$currentActionState' = currentActionState$
$actionInPlan' = actionInPlan$
$actionParticipants' = actionParticipants \cup \{\ a? \mapsto agent?\ \}$
$actionLocation' = actionLocation$

AllocateActionLocationOK

Δ Actions
Ξ Plans
Ξ VirtualInstitute
a? : Action
place?: Place

$a? \in actions \wedge place? = place\ s \wedge (a? \mapsto place?) \notin actionLocation$

actions’ = actions
currentActionState’ = currentActionState
actionInPlan’ = actionInPlan
actionParticipants’ = actionParticipants
actionLocation’ = actionLocation ⊕ { a? ↦ place? }

The following schema is a specification of the operation to define an artifact in a plan.

```
── CreateArtifactOK ──────────────────────
Δ Artifacts
Ξ Plans
Ξ Actions
name? : STRING
aPlan?: PBLPlan
──────────────────────
aPlan ∈ plans

aDocument = = (μ Document | title = name?)
anArtifact = = (μ Artifact | name = name? ∧ referTo = aDocument)
artifacts’ = artifacts ∪ { anArtifact }
artifactInPlan’ = artifactInPlan ∪ { anArtifact ↦ aPlan? }
currentArtifactState’ = currentArtifactState ∪ { anArtifact ↦ created }
actionProduceArtifact’ = actionProduceArtifact
artifactConsumedByAction’ = artifactConsumedByAction
actionSharedArtifact’ = actionSharedArtifact
──────────────────────────────────────────
```

In each PBL-plan, there is a start point from which a plan starts to execute, and an end point at which a plan is finished. Because these two types of points are created in the same way, only the operation to create a start point of a plan is specified.

```
── CreateStartOK ──────────────────────
Δ ConnectionNodes
Ξ Plans
startPoint? : StartPoint
aPlan?: PBLPlan
──────────────────────
{aPlan?} ◁ planStartPoint = ∅

startPoint?.currentState = false
planStartPoint’ = planStartPoint ∪ {aPlan? ↦ startPoint?}
planEndPoint’ = planEndPoint
andJoinInPlan’ = andJoinInPlan
orJoinInPlan’ = orJoinInPlan
andSplitInPlan’ = andSplitInPlan
orSplitInPlan’ = orSplitInPlan
──────────────────────────────────────────
```

Four types of connection nodes (AndJoin, OrJoin, AndSplit, and OrSplit) can be used to define the temporal relations between actions. We take the specification of the operation to create an AndJoin node as an example to show how to specify the operations for creating connection nodes.

CreateAndJoinOK

Δ ConnectionNodes
Ξ Plans
andJoinPoint? : AndJoin
aPlan?: PBLPlan

(andJoinPoint? $\mapsto$ aPlan?) $\notin$ andJoinInPlan’

andJoinPoint?.currentState = false
planStartPoint’ = planStartPoint
planEndPoint’ = planEndPoint
andJoinInPlan’ = andJoinInPlan $\cup$ { andJoinPoint? $\mapsto$ aPlan? }
orJoinInPlan’ = orJoinInPlan
andSplitInPlan’ = andSplitInPlan
orSplitInPlan’ = orSplitInPlan

The operation “CreateActionSequenceOK” is used to define a temporal relation between two actions.

CreateActionSequenceOK

Δ TemporalRelations
Ξ Plans
Ξ Actions
Ξ ConnectionNodes
a_1?, a_2? : Action
aPlan?: PBLPlan

a_1? $\in$ actions $\wedge$ a_2? $\in$ actions $\wedge$
actionInPlan a_1? = actionInPlan a_2? = aPlan? $\wedge$
(a_1? $\mapsto$ a_2?) $\notin$ actionSequence

parentPlan’ = parentPlan
startPointToAction’ = startPointToAction
startPointToPlan’ = startPointToPlan
actionToEndPoint’ = actionToEndPoint
planToEndPoint’ = planToEndPoint
actionSequence’ = actionSequence $\cup$ { a_1? $\mapsto$ a_2? }
planToPlan’ = planToPlan
actionToPlan’ = actionToPlan
planToAction’ = planToAction

actionToAndJoin’ = actionToAndJoin
actionToOrJoin’ = actionToOrJoin
actionToAndSplit’ = actionToAndSplit
actionToOrSplit’ = actionToOrSplit
andJoinToAction’ = andJoinToAction
orJoinToAction’ = orJoinToAction
andSplitToAction’ = andSplitToAction
orSplitToAction’ = orSplitToAction
planToAndJoin’ = planToAndJoin
planToOrJoin’ = planToOrJoin
planToAndSplit’ = planToAndSplit
planToOrSplit’ = planToOrSplit
andJoinToPlan’ = andJoinToPlan
orJoinToPlan’ = orJoinToPlan
andSplitToPlan’ = andSplitToPlan
orSplitToPlan’ = orSplitToPlan
startPointToOrJoin’ = startPointToOrJoin
startPointToAndSplit’ = startPointToAndSplit
startPointToOrSplit’ = startPointToOrSplit
andJoinToEndPoint’ = andJoinToEndPoint
orJoinToEndPoint’ = orJoinToEndPoint

The relation between the start point of a plan and an action in the plan is defined by using the following operation.

CreateStartActionOK
Δ TemporalRelations
Ξ Plans
Ξ Actions
Ξ ConnectionNodes
a? : Action
startPoint? : StartPoint
aPlan?: PBLPlan

a? $\in$ actions $\wedge$ actionInPlan a? = aPlan $\wedge$
startPoint? = planStartPoint aPlan $\wedge$ (startPoint? $\mapsto$ a?) $\notin$ startPointToAction

parentPlan’ = parentPlan
startPointToAction’ = startPointToAction $\cup$ { startPoint? $\mapsto$ a? }
startPointToPlan’ = startPointToPlan
actionToEndPoint’ = actionToEndPoint
planToEndPoint’ = planToEndPoint
actionSequence’ = actionSequence
planToPlan’ = planToPlan
actionToPlan’ = actionToPlan
planToAction’ = planToAction
actionToAndJoin’ = actionToAndJoin
actionToOrJoin’ = actionToOrJoin

actionToAndSplit’ = actionToAndSplit
actionToOrSplit’ = actionToOrSplit
andJoinToAction’ = andJoinToAction
orJoinToAction’ = orJoinToAction
andSplitToAction’ = andSplitToAction
orSplitToAction’ = orSplitToAction
planToAndJoin’ = planToAndJoin
planToOrJoin’ = planToOrJoin
planToAndSplit’ = planToAndSplit
planToOrSplit’ = planToOrSplit
andJoinToPlan’ = andJoinToPlan
orJoinToPlan’ = orJoinToPlan
andSplitToPlan’ = andSplitToPlan
orSplitToPlan’ = orSplitToPlan
startPointToOrJoin’ = startPointToOrJoin
startPointToAndSplit’ = startPointToAndSplit
startPointToOrSplit’ = startPointToOrSplit
andJoinToEndPoint’ = andJoinToEndPoint
orJoinToEndPoint’ = orJoinToEndPoint

The operation to define a relation between an action and an AndJoin node is specified below. In the same way, we can specify the operation to define a relation between an action and an OrJoin node. In order to save space, the specification of this operation is omitted.

CreateActionToAndJoinOK
Δ TemporalRelations
Ξ Plans
Ξ Actions
Ξ ConnectionNodes
a? : Action
andJoin? : AndJoin
aPlan?: PBLPlan

$a? \in actions \wedge actionInPlan\ a? = aPlan \wedge$
$andJoin? \in \mathbf{dom}\ (andJoinInPlan \rhd \{aPlan\}) \wedge$
$(a? \mapsto andJoin?) \notin actionToAndJoin$

parentPlan’ = parentPlan
startPointToAction’ = startPointToAction
startPointToPlan’ = startPointToPlan
actionToEndPoint’ = actionToEndPoint
planToEndPoint’ = planToEndPoint
actionSequence’ = actionSequence
planToPlan’ = planToPlan
actionToPlan’ = actionToPlan
planToAction’ = planToAction
$actionToAndJoin' = actionToAndJoin \cup \{ a? \mapsto andJoin? \}$

actionToOrJoin’ = actionToOrJoin
actionToAndSplit’ = actionToAndSplit
actionToOrSplit’ = actionToOrSplit
andJoinToAction’ = andJoinToAction
orJoinToAction’ = orJoinToAction
andSplitToAction’ = andSplitToAction
orSplitToAction’ = orSplitToAction
planToAndJoin’ = planToAndJoin
planToOrJoin’ = planToOrJoin
planToAndSplit’ = planToAndSplit
planToOrSplit’ = planToOrSplit
andJoinToPlan’ = andJoinToPlan
orJoinToPlan’ = orJoinToPlan
andSplitToPlan’ = andSplitToPlan
orSplitToPlan’ = orSplitToPlan
startPointToOrJoin’ = startPointToOrJoin
startPointToAndSplit’ = startPointToAndSplit
startPointToOrSplit’ = startPointToOrSplit
andJoinToEndPoint’ = andJoinToEndPoint
orJoinToEndPoint’ = orJoinToEndPoint

The operation to define a relation between an action and an AndSplit node is specified as follows. An important predicate of this operation is that there is no such pair between the action and the AndSplit node in the relation “actionToAndSplit”. In the same way, we can specify the operation to define a relation between an action and an OrSplit node. Here, the specification of the operation to define a relation between an action and an OrSplit node is omitted.

CreateActionToAndSplitOK
Δ TemporalRelations
Ξ Plans
Ξ Actions
Ξ ConnectionNodes
a? : Action
andSplit? : AndSplit
aPlan?: PBLPlan

a? ∈ actions ∧ actionInPlan a? = aPlan? ∧
andSplit? ∈ **dom** (andSplitInPlan ⊳ {aPlan?}) ∧
(a? ↦ andSplit?) ∉ actionToAndSplit ∧ actionToAndSplit ⊳ {andSplit?} = ∅

parentPlan’ = parentPlan
startPointToAction’ = startPointToAction
startPointToPlan’ = startPointToPlan
actionToEndPoint’ = actionToEndPoint
planToEndPoint’ = planToEndPoint
actionSequence’ = actionSequence
planToPlan’ = planToPlan

actionToPlan’ = actionToPlan
planToAction’ = planToAction
actionToAndJoin’ = actionToAndJoin
actionToOrJoin’ = actionToOrJoin
actionToAndSplit’ = actionToAndSplit ∪ { a? ↦ andSplit? }
actionToOrSplit’ = actionToOrSplit
andJoinToAction’ = andJoinToAction
orJoinToAction’ = orJoinToAction
andSplitToAction’ = andSplitToAction
orSplitToAction’ = orSplitToAction
planToAndJoin’ = planToAndJoin
planToOrJoin’ = planToOrJoin
planToAndSplit’ = planToAndSplit
planToOrSplit’ = planToOrSplit
andJoinToPlan’ = andJoinToPlan
orJoinToPlan’ = orJoinToPlan
andSplitToPlan’ = andSplitToPlan
orSplitToPlan’ = orSplitToPlan
startPointToOrJoin’ = startPointToOrJoin
startPointToAndSplit’ = startPointToAndSplit
startPointToOrSplit’ = startPointToOrSplit
andJoinToEndPoint’ = andJoinToEndPoint
orJoinToEndPoint’ = orJoinToEndPoint

In a similar way, we can specify the operations to define relations between one type of connection nodes and an action.

CreateAndJoinToActionOK
Δ TemporalRelations
Ξ Plans
Ξ Actions
Ξ ConnectionNodes
a? : Action
andJoin? : AndJoin
aPlan?: PBLPlan

a? ∈ actions ∧ actionInPlan a? = aPlan? ∧
andJoin? ∈ **dom** (andJoinInPlan ▷ {aPlan?}) ∧
(andJoin? ↦ a?) ∉ andJoinToAction ∧ {andJoin?} ◁ andJoinToAction = ∅

parentPlan’ = parentPlan
startPointToAction’ = startPointToAction
startPointToPlan’ = startPointToPlan
actionToEndPoint’ = actionToEndPoint
planToEndPoint’ = planToEndPoint
actionSequence’ = actionSequence
planToPlan’ = planToPlan
actionToPlan’ = actionToPlan

planToAction’ = planToAction
actionToAndJoin’ = actionToAndJoin
actionToOrJoin’ = actionToOrJoin
actionToAndSplit’ = actionToAndSplit
actionToOrSplit’ = actionToOrSplit
andJoinToAction’ = andJoinToAction $\cup$ { andJoin? $\mapsto$ a?}
orJoinToAction’ = orJoinToAction
andSplitToAction’ = andSplitToAction
orSplitToAction’ = orSplitToAction
planToAndJoin’ = planToAndJoin
planToOrJoin’ = planToOrJoin
planToAndSplit’ = planToAndSplit
planToOrSplit’ = planToOrSplit
andJoinToPlan’ = andJoinToPlan
orJoinToPlan’ = orJoinToPlan
andSplitToPlan’ = andSplitToPlan
orSplitToPlan’ = orSplitToPlan
startPointToOrJoin’ = startPointToOrJoin
startPointToAndSplit’ = startPointToAndSplit
startPointToOrSplit’ = startPointToOrSplit
andJoinToEndPoint’ = andJoinToEndPoint
orJoinToEndPoint’ = orJoinToEndPoint

CreateAndSplitToActionOK
Δ TemporalRelations
Ξ Plans
Ξ Actions
Ξ ConnectionNodes
a? : Action
andSplit? : AndSplit
aPlan?: PBLPlan

a? $\in$ actions $\wedge$ actionInPlan a? = aPlan? $\wedge$
andSplit? $\in$ **dom** (andSplitInPlan $\rhd$ {aPlan?}) $\wedge$
(andSplit? $\mapsto$ a?) $\notin$ andSplitToAction

parentPlan’ = parentPlan
startPointToAction’ = startPointToAction
startPointToPlan’ = startPointToPlan
actionToEndPoint’ = actionToEndPoint
planToEndPoint’ = planToEndPoint
actionSequence’ = actionSequence
planToPlan’ = planToPlan
actionToPlan’ = actionToPlan
planToAction’ = planToAction
actionToAndJoin’ = actionToAndJoin
actionToOrJoin’ = actionToOrJoin
actionToAndSplit’ = actionToAndSplit

actionToOrSplit' = actionToOrSplit
andJoinToAction' = andJoinToAction
orJoinToAction' = orJoinToAction

andSplitToAction' = andSplitToAction ∪ { andSplit? ↦ a?}
orSplitToAction' = orSplitToAction
planToAndJoin' = planToAndJoin
planToOrJoin' = planToOrJoin
planToAndSplit' = planToAndSplit
planToOrSplit' = planToOrSplit
andJoinToPlan' = andJoinToPlan
orJoinToPlan' = orJoinToPlan
andSplitToPlan' = andSplitToPlan
orSplitToPlan' = orSplitToPlan
startPointToOrJoin' = startPointToOrJoin
startPointToAndSplit' = startPointToAndSplit
startPointToOrSplit' = startPointToOrSplit
andJoinToEndPoint' = andJoinToEndPoint
orJoinToEndPoint' = orJoinToEndPoint

The operations to create relations between sub-plan and connection nodes are similar to the operations that create relations between action and connection nodes. Therefore, the specifications of operations to create relations between sub-plan and connection nodes are omitted.

The relations between actions and artifacts are defined by using the following operations. The first operation is used to specify that an artifact is produced in an action. The second operation is used to specify that an artifact is consumed by an action. The third operation is used to specify that an action shares an artifact usually with other concurrent actions.

CreateActionProduceArtifactOK
Δ Artifacts
Ξ Plans
Ξ Actions
artifact? : Artifact
a? : Action
aPlan?: PBLPlan

artifact? ∈ artifacts ∧ artifactInPlan artifact? = aPlan? ∧
a? ∈ actions ∧ actionInPlan a? = aPlan? ∧
(a? ↦ artifact?) ∉ actionProduceArtifact ∧
(artifact? ↦ a?) ∉ artifactConsumedByAction ∧
(a? ↦ artifact?) ∉ actionSharedArtifact

artifacts' = artifacts
artifactInPlan' = artifactInPlan
actionProduceArtifact' = actionProduceArtifact ∪ {a? ↦ artifact?}

artifactConsumedByAction’ = artifactConsumedByAction
actionSharedArtifact’ = actionSharedArtifact

CreateArtifactConsumedByActionOK

Δ Artifacts
Ξ Plans
Ξ Actions
artifact? : Artifact
a? : Action
aPlan?: PBLPlan

artifact? $\in$ artifacts $\wedge$ artifactInPlan artifact? = aPlan? $\wedge$
a? $\in$ actions $\wedge$ actionInPlan a? = aPlan? $\wedge$
(artifact? $\mapsto$ a?) $\notin$ artifactConsumedByAction $\wedge$
(a? $\mapsto$ artifact?) $\notin$ actionProduceArtifact $\wedge$
(a? $\mapsto$ artifact?) $\notin$ actionSharedArtifact

artifacts’ = artifacts
artifactInPlan’ = artifactInPlan
actionProduceArtifact’ = actionProduceArtifact
artifactConsumedByAction’ = artifactConsumedByAction $\cup$ {artifact? $\mapsto$ a?}
actionSharedArtifact’ = actionSharedArtifact

CreateActionSharedArtifactOK

Δ Artifacts
Ξ Plans
Ξ Actions
artifact? : Artifact
a? : Action
aPlan?: PBLPlan

artifact? $\in$ artifacts $\wedge$ artifactInPlan artifact? = aPlan? $\wedge$
a? $\in$ actions $\wedge$ actionInPlan a? = aPlan? $\wedge$
(a? $\mapsto$ artifact?) $\notin$ actionSharedArtifact $\wedge$
(a? $\mapsto$ artifact?) $\notin$ actionProduceArtifact $\wedge$
(artifact? $\mapsto$ a?) $\notin$ artifactConsumedByAction

artifacts’ = artifacts
artifactInPlan’ = artifactInPlan
actionProduceArtifact’ = actionProduceArtifact
artifactConsumedByAction’ = artifactConsumedByAction
actionSharedArtifact’ = actionSharedArtifact $\cup$ {a? $\mapsto$ artifact?}

The operations to create elements of a PBL-plan and the relations between these elements were specified in this section. In order to focus on the major design ideas and to save space, the operations to delete and modify the definition of the created elements and their relations were omitted in this specification. The users of a virtual learning environment can define and modify their own PBL-plan by using these operations. When a PBL-plan is defined, it can be executed according to the definition of the PBL-plan. It is allowed to modify those parts of a PBL-plan that are currently not executed. Before specifying how a PBL-plan is executed, we describe facilities to help users to define PBL-plans in the next two sections.

4.6.3 From Learning Issues and Necessary Learning Resources to a PBL-plan

As described in chapter 2, a learning group has to discover areas in which the collective knowledge is deficient. Recognizing such a deficiency, the learning group may elect to treat it as a learning issue which will need to be researched, applied to the problem, and appropriately integrated with other information. For researching learning issues, the learning group has to identify learning resources including texts, journal articles, library resources, computer information and database, and faculty.

Dolmans et al. wrote: "It is the policy of the particular implementation of PBL that learning issues are always to be generated by the learners in the PBL group, rather than determined in advance by the faculty. Producing a learning issue is a collaborative effort, therefore, requiring the learners to assess their current understanding and evaluate their current need to know. Recording an item as a learning issue, therefore, represents a commitment on the part of the group to further research the topic. Learning issues have been shown to be critical determinants of learners' self-directed learning and, on this basis, they represent an important component of the method" [Dolmans94].

Koschmann et al. [Koschmann97] pointed out that "… to become a learning issue a topic must satisfy three conditions. Firstly, there must be a recognizable knowledge deficiency. Secondly, the learners must see the missing knowledge as relevant to or necessary for understanding and solving the problem under study. Thirdly, there must be consensus about the timeliness of undertaking the study."

During identifying learning issues, according to [Koschmann97], "participants should continuously re-negotiate the boundaries of the topic. In general, any group member may clarify, expand, restrict, or otherwise alter a topic. The set of identified learning issues is not static but dynamic and emergent. Much of the conversational work that takes place within this discourse is devoted to specifying just what the topic of the discussion actually is. This process is important, for it directly affects how a learning issue gets identified, which in turn will crucially influence the success of subsequent research on the issue."

In [Pross99], it is suggested that the learners need to clarify their plans for their own learning by:

1) "identifying all of the significant issues arising from the problem under study - what is known for understanding and solving the problem and what do they need to know?
2) settling on a 'do-able' list of learning tasks and deciding which issues everyone will tackle and which will be divided up (some issues are so fundamental to the whole area that all students should read about them themselves).
3) deciding what specific questions individuals will try to answer (even minor issues should be looked up by at least two individuals, to promote discussion).
4) deciding on the 'enquiry strategy' - how they will address these learning issues (e.g. by looking up notes from a course, reading a section of a textbook, doing a literature search, searching the internet, consulting an expert, accessing community resources, conducting experiment, and so on)."

After identifying the learning issues and learning resources, a learning group will arrange actions to research the learning issues by collecting and using learning resources. As described in chapter 3, in the CALE system, an items in the *need more information* category can be turned into an action item, which in turn would be assigned to team members with a due date. The transformed action items are organized as a list of commitments. Following this idea, a virtual PBL environment can provide further support to help learners make learning plans by creating preliminary learning plans based on the information recorded during the discourse of identifying learning issues and resources.

As discussed in section 4.4, all members of a PBL group collaboratively construct a shared PBL-net. A PBL-net provides a means for learners to negotiate learning issues and identify resources. As a result, the *issue* nodes, *resource* nodes, and their relations (represented as typed links) are created in the PBL-net.

In order to support the process of identifying learning issues, we define following data types and operations.

Definition (Profession): A *profession* is used to make a public profession of a learner's belief about a learning issue. The needToKnow attribute is used to indicate whether the declarer needs to know the learning issue. The knowOrNotKnow attribute is used to indicate whether the declarer has knowledge about the learning issue.

```
┌─ Profession ──────────────────────────────────
│ declarer : Actor
│ needToKnow : BOOLEAN
│ knowOrNotKnow : BOOLEAN
└───────────────────────────────────────────────
```

Definition (Profession Issues): *Profession issues* are used to record the learner's professions about learning issues.

```
┌─ ProfessionIssues ────────────────────────────
│ PBLNetBase
│
│ professLearningIssue : Profession → TypedNode
├───────────────────────────────────────
```

| $\forall$ p : professLearningIssue | p.nodeType = ‘issue’

The operation to make a public profession is specified as follows.

```
── ProfessOK ──────────────────────────────
Δ ProfessionIssues
a? : Actor
node? : TypedNode
need? : BOOLEAN
knk? : BOOLEAN
──────────────────────────
node?.nodeType = ‘issue’

let aProfession = = ( μ Profession | declarer = a? ∧
                      needToKnow = need? ∧
                      knowOrNotKnow = knk? ) •
    professLearningIssue’ = professLearningIssue ∪ { aProfession ↦ aNode }
───────────────────────────────────────────
```

Based on the information in a PBL-net, a preliminary learning plan can be created by the system. The algorithm to create a preliminary learning plan is described below.

Given (aPBLNet : Net | aPBLNet $\in$ pblNets) that recorded the information created during the discourse of identifying learning issues. Only the *issue* and *resource* nodes and the links between these nodes in the given PBL-net are used as original information to create the preliminary learning plan. These nodes and links form a sub-net on the given PBL-net. In such a sub-net, there are two types of nodes and four types of links.

Issues = { n : TypedNode | n.nodeType = ‘issue’ $\wedge$ (n $\mapsto$ aPBLNet) $\in$ typedNodes }

Resources = { n : TypedNode | n.nodeType = ‘resource’ $\wedge$ (n $\mapsto$ aPBLNet) $\in$ typedNodes }

SubLinks = { l : TypedLink | l.linkType = ‘is_a_sub_issue’ $\wedge$ (l $\mapsto$ aPBLNet) $\in$ typedLinks }

PriorLinks = { l : TypedLink | l.linkType = ‘is_prior_to’ $\wedge$ (l $\mapsto$ aPBLNet) $\in$ typedLinks }

PrerequisiteLinks = { l : TypedLink | l.linkType = ‘is_a_prerequisite_for’ $\wedge$ (l $\mapsto$ aPBLNet) $\in$ typedLinks }

ConcernLinks = { l : TypedLink | l.linkType = ‘concern’ $\wedge$ (l $\mapsto$ aPBLNet) $\in$ typedLinks }

We take an example to explain the algorithm to create a preliminary learning plan. We assume that a PBL-net contains ten issue nodes (I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9, and I_{10}), one resource node (R), six prerequisite links (pre_1, pre_2, pre_3, pre_4, pre_5, and pre_6), six sub-links (sub_1, sub_2, sub_3, sub_4, sub_5, and sub_6), and two concern links (con_1 and con2). The sub-net of this PBL-net is shown in Figure 4.14. We assume that the name of the preliminary learning plan created based on this sub-net is P_0.

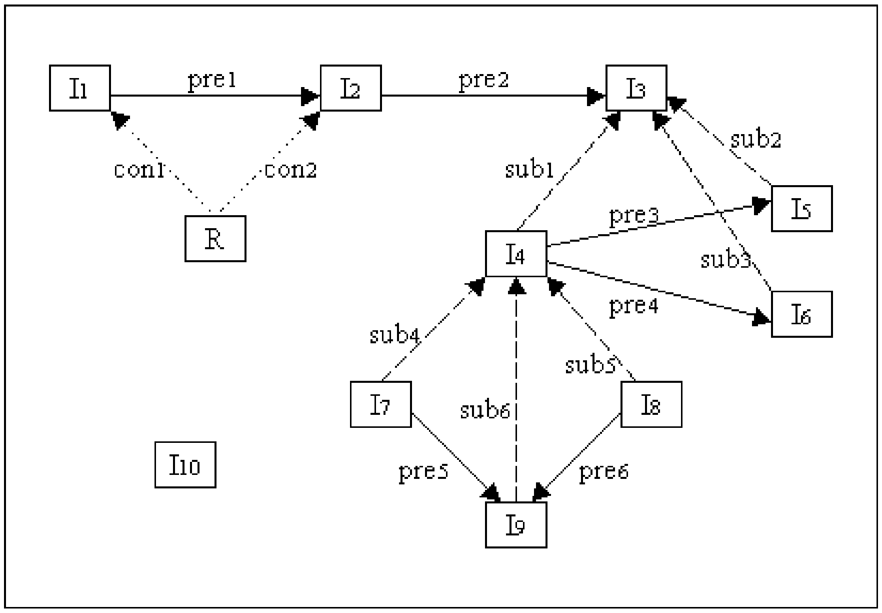


Figure 4.14: The Sub-net of a PBL-net

Each issue node belongs to one of four categories, according to the relations between the issue node and other issue nodes. These four categories of issue nodes are turned into actions or sub-plans, respectively.

Step 1: creating plans and actions for corresponding issue nodes.

For the learning issue nodes that are only connected to other issue nodes with SubLinks, create a corresponding plan node (represented as n≫plan) that is a sub-plan of the aPlan.

$\forall$ n $\in$ Issues; $\nexists$ l_1 $\in$ SubLinks; $\exists$ l_2 $\in$ SubLinks | $l_1 \neq l_2 \wedge$

$$l_1.\text{sourceNode} = n \wedge l_2.\text{destinationNode} = n$$

create n≫plan : PBLPlan

plans' = plans $\cup$ { n≫plan }

parentPlan' = parentPlan $\cup$ {n≫plan $\mapsto$ aPlan}

In the example net, only I_3 meets the condition. Therefore, a sub-plan P_1 is created as a sub-plan of P_0 (see Figure 4.15).

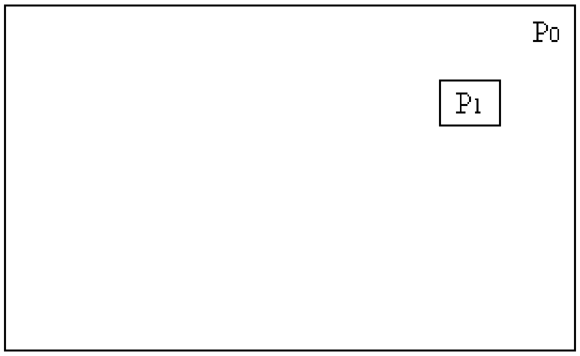


Figure 4.15: Creating a Sub-plan of the Overall PBL-plan

For the learning issue nodes that connect to and are connected from other issue nodes with SubLinks, create a corresponding plan node (n≫plan) that is a sub-plan of the plan (l_1.destinationNode≫plan) that is created correspondingly for the issue node l_1.destinationNode.

$\forall$ n $\in$ Issues; $\exists$ l_1, l_2 $\in$ SubLinks | $l_1 \neq l_2 \wedge$

l_1.sourceNode = n $\wedge$ l_2.destinationNode = n

create n≫plan : PBLPlan

plans’ = plans $\cup$ { n≫plan }

parentPlan’ = parentPlan $\cup$ {n≫plan $\mapsto$ l_1.destinationNode≫plan }

In the example net, only I_4 meets the condition. Therefore, a sub-plan P_2 is created as a sub-plan of P_1 (see Figure 4.16).

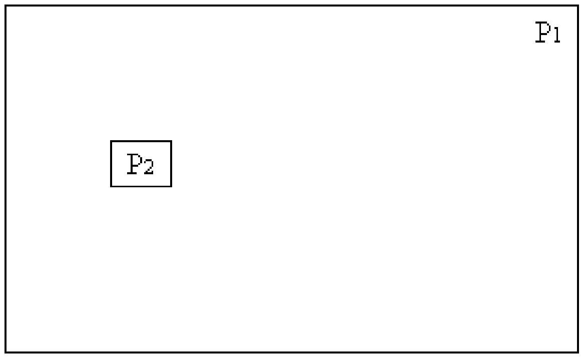


Figure 4.16: Creating a Sub-plan of Another Sub-plan

For the learning issue nodes that only connect to other issue nodes with SubLinks, create a corresponding action node (n≫action) that is defined in the plan (l_2.destinationNode≫plan), which is created correspondingly for the issue node l_2.destinationNode.

$\forall$ n $\in$ Issues; $\nexists$ l_1 $\in$ SubLinks; $\exists$ l_2 $\in$ SubLinks | $l_1 \neq l_2 \wedge$

```
                              $l_1$.destinationNode = n $\wedge$ $l_2$.sourceNode = n
    create n≫action: Action
    actions' = actions $\cup$ { n≫action }
    actionInPlan' = actionInPlan $\cup$ {n≫action $\mapsto$ $l_2$.destinationNode≫plan }
```

In the example net, I_5, I_6, I_7, I_8, and I_9 meets the condition. Therefore, five actions are created in sub-plan P_1 and P_2 (see Figure 4.17).

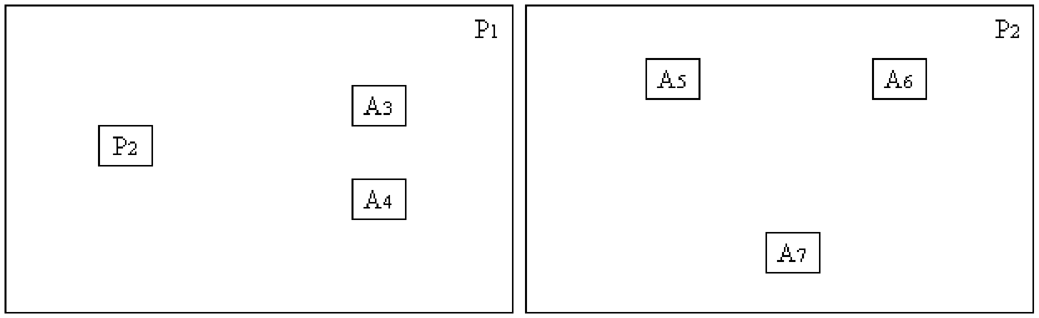


Figure 4.17: Creating Actions in Sub-plans

For the learning issue nodes that neither connect to nor are connected from other issue nodes with SubLinks, create a corresponding action node (n≫action) that is defined in the aPlan.

```
$\forall$ n $\in$ Issues; $\nexists$ l $\in$ SubLinks | l.sourceNode = n $\vee$ l.destinationNode = n
    create n≫action: Action
    actions' = actions $\cup$ { n≫action }
    actionInPlan' = actionInPlan $\cup$ {n≫action $\mapsto$ aPlan }
```

In the example net, I_1, I_2, and I_{10} meets the condition. Therefore, three actions are created in the overall plan P_0 (see Figure 4.18).

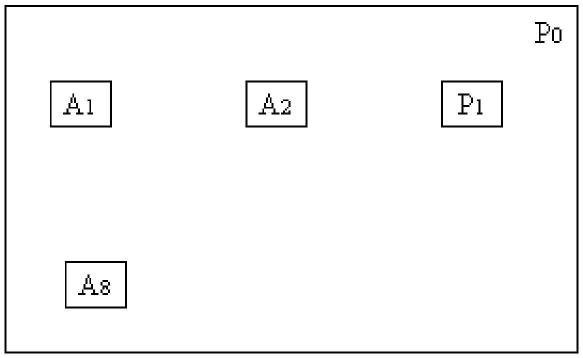


Figure 4.18: Creating Actions in the Overall PBL-plan

Step 2: creating temporal relations between plans, actions, and connection nodes for corresponding PrerequisiteLinks.

For two issue nodes that are connected by a unique link in the PrerequisiteLinks, create a temporal relationship according to the correspondingly created actions or plans.

$\forall\, n_1, n_2 \in$ Issues; $\exists\, l_1 \in$ PrerequisiteLinks; $\nexists\, l_2 \in$ PrerequisiteLinks |

$n_1 \neq n_2 \wedge l_1 \neq l_2 \wedge$

l_1.sourceNode = $n_1 \wedge l_1$.destinationNode = $n_2 \wedge$

(l_2.sourceNode = $n_1 \vee l_2$.destinationNode = n_2) •

actionSequence' = actionSequence $\cup$ {$n_1 \gg$action $\mapsto n_2 \gg$action } $\vee$

planToPlan' = planToPlan $\cup$ {$n_1 \gg$plan $\mapsto n_2 \gg$plan} $\vee$

actionToPlan' = actionToPlan $\cup$ {$n_1 \gg$action $\mapsto n_2 \gg$plan } $\vee$

planToAction' = planToAction $\cup$ {$n_1 \gg$plan $\mapsto n_2 \gg$action }

In the example net, (I_1, I_2) and (I_2, I_3) meets the condition. Therefore, two temporal relationships are created (see Figure 4.19).

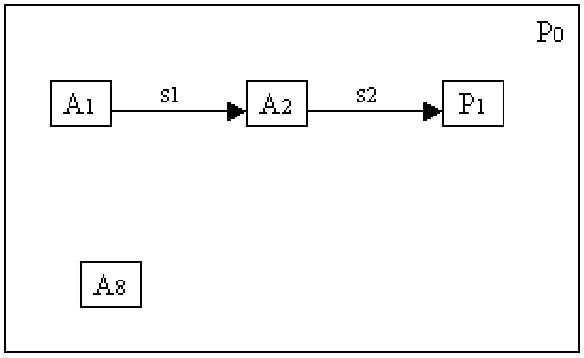


Figure 4.19: Creating Temporal Relationships

For a set of 'is_a_prerequisite_for' links that connect to the same issue node, create an AndJoin node and create temporal relationships between the correspondingly created actions or plans and the AndJoin node, and create a temporal relationship between the AndJoin node and the correspondingly created action or plan for the common destination.

$\forall\, n \in$ Issues | # { l $\in$ PrerequisiteLinks | l.destinationNode = n } ≥ 2

create anAndJoin : AndJoin;

(actionToAndJoin' = actionToAndJoin $\cup$

{l.sourceNode$\gg$action $\mapsto$ anAndJoin } $\vee$

planToAndJoin' = planToAndJoin $\cup$ {l.sourceNode$\gg$plan $\mapsto$ anAndJoin }) $\wedge$

(andJoinToAction' = andJoinToAction $\cup$ {anAndJoin $\mapsto$ n$\gg$action} $\vee$

andJoinToPlan' = andJoinToPlan $\cup$ {anAndJoin $\mapsto$ n$\gg$plan})

In the example net, (I_7, I_9) and (I_8, I_9) meets the condition. Therefore, an AndJoin node and three temporal relationships are created (see Figure 4.20).

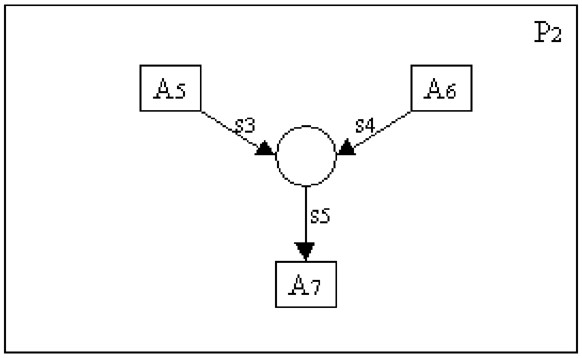


Figure 4.20: Creating an AndJoin Node and Temporal Relationships

For a set of 'is_a_prerequisite_for' links that are connected from the same issue node, create an AndSplit node and create temporal relations between the AndSplit node and the correspondingly created actions or plans, and create a temporal relation between the correspondingly created action or plan for the common source and the AndSplit node.

```
$\forall$ n $\in$ Issues | # { l $\in$ PrerequisiteLinks | l.sourceNode = n } $\geq$ 2
    create anAndSplit : AndSplit
    (actionToAndSplit' = actionToAndSplit $\cup$ {n$\gg$action $\mapsto$ anAndSplit} $\vee$
    planToAndSplit' = planToAndSplit $\cup$ {n$\gg$plan $\mapsto$ anAndSplit}) $\wedge$
    (andSplitToAction' = andSplitToAction $\cup$ {anAndSplit $\mapsto$
        l.destinationNode$\gg$action} $\vee$
    andSplitToPlan' = andSplitToPlan $\cup$
        {anAndSplit $\mapsto$ l.destinationNode$\gg$plan})
```

In the example net, (I_4, I_5) and (I_4, I_6) meets the condition. Therefore, an AndSplit node and three temporal relationships are created (see Figure 4.21).

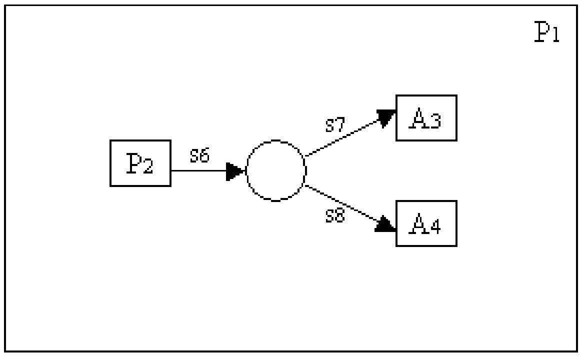


Figure 4.21: Creating an AndSplit Node and Temporal Relationships

Step 3: creating artifacts and relations between artifacts and actions.

For resource nodes that connect to an issue node by a link that is a member of the set ConcernLinks, create a corresponding artifact node (n≫artifact) and a relation between this node and the correspondingly created action for the issue node.

```
∀ n ∈ Resources | (∃ l : ConcernLinks | l.sourceNode = n )
        create n≫artifact : Artifact
        artifacts' = artifacts ∪ { n≫artifact }
        artifactConsumedByAction' = artifactConsumedByAction ∪
        { n≫artifact ↦ l.destinationNode≫action }
```

In the example net, R is a resource node, which connects to I_1 and I_2. Therefore, an artifact node and two artifact relationships are created (see Figure 4.22).

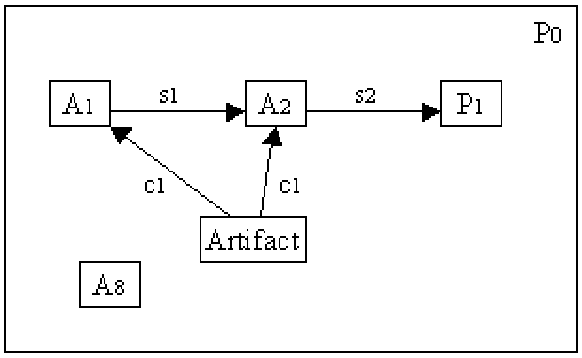


Figure 4.22: Creating a Artifact and Two Artifact Relationships

Step 4: assign participants for each action.

For each action (issue≫action) that is created correspondingly for a learning issue (issue : Issues), assign participants for the action.

```
∀ issue≫action : Action |
        actionParticipants' = actionParticipants ∪
        { ∀ p : Profession | (p ↦ issue) ∈ professLearningIssue ∧
                p.needToKnow = true ∧ p.knowOrNotKnow = false •
                (issue≫action ↦ p.declarer)}
```

By using this algorithm, a preliminary learning plan can be created. The preliminary learning plan created based on the example net is shown in Figure 4.23.

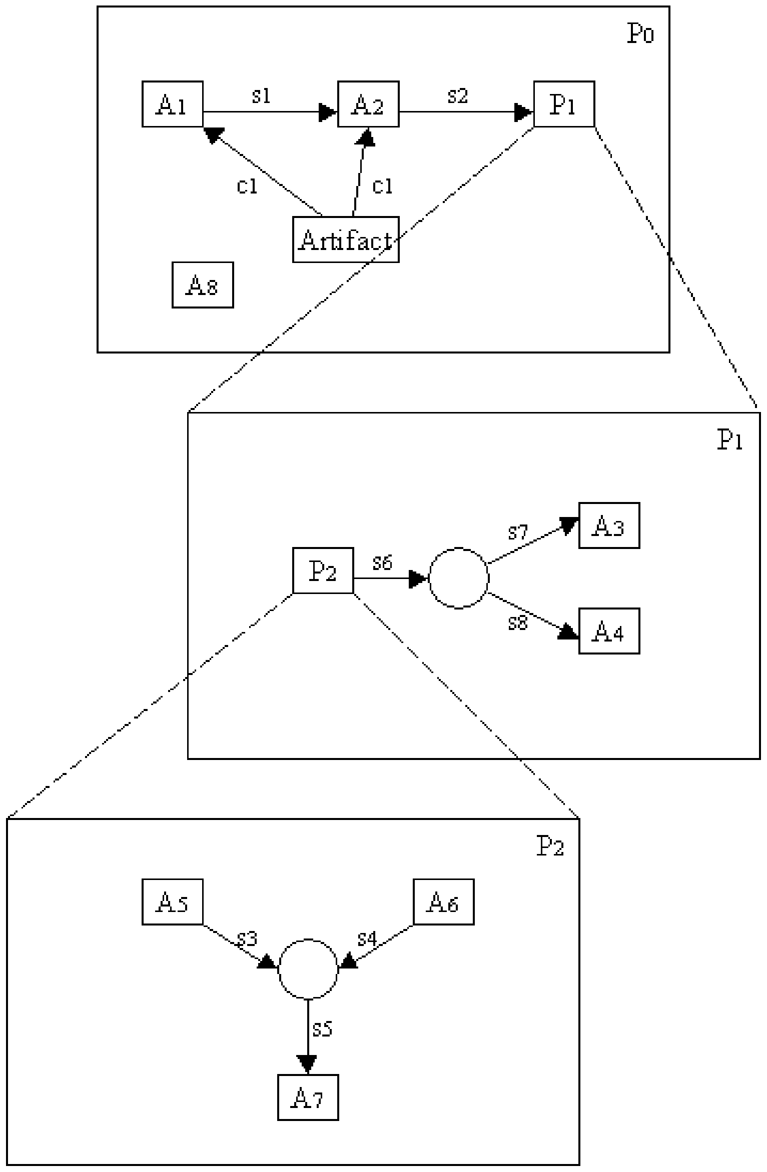


Figure 4.23: The Example Preliminary Learning Plan

This transformation automates a lot of users' work to define a learning plan. However, in order to complete a definition of learning plan, learners have to continually work on the definition of the learning plan. For example, they should allocate a place for each action, specify the active-condition and terminated-condition for each action, schedule start time and duration, specify which actions produce or share which artifacts, and so on. If necessary, they have to modify the learning plan. Defining a learning plan is a very error-prone and time-consuming task. The next section presents an approach to support refining learning plans.

4.6.4 Modifying and Refining PBL-plans Interactively

As mentioned above, it is not an easy task to define a good learning plan. Many actions may be carried out in the same period of time. Normally, multiple participants are assigned to perform an action and the action may be carried out in a synchronous session. An actor may take part in multiple actions. Furthermore, some resources have to be shared across time. For example, some virtual places provide specific learning contexts which are suitable to perform task-specific actions. However, these actions can not be performed in the same virtual place and at the same time. In addition, a set of constraints are specified as temporal relations and artifact relations between actions in plans. Generally, arranging resources (people, time, place, and artifact) for actions by systems is problematic. Some efforts are made to schedule meetings automatically. For example, the electronic calendar system [Ehrlich87] tried to find a time convenient for all participants by checking the calendar for each person. Why it is rarely used was explained in [Grudin94]. The major problems are "disparity between those who will benefit and those who must do the work" and "free time is not free". Instead of providing fully automatic support, in this thesis an alternative approach is developed. Adopting this approach, the system can help learners modify and refine learning plan in an interactive way. That is, on demand the system detects and displays the incomplete definitions and potential conflicts between the scheduled actions in a plan. Learners, then, modify and refine the definition of the learning plan to resolve the detected conflicts. Even if a learning plan has not been defined completely or it contains conflicts, it can be executed. However, whenever a conflict is reached or necessary information is still missing during execution, the execution will pause. It can be resumed from the interrupted point after modifying or refining the plan. This subsection presents how the system detects incomplete definitions and possible conflict situations in PBL-plans.

4.6.4.1 Incomplete definition

Some values of attributes are assigned as default values such as the state of action. However, learners have to assign some values. Otherwise, the learning plan can not execute or will stop in the process of execution. The situations of incomplete definition and how these situations can be detected by predicates are listed below. These predicates are used when editing a plan finishes in order to detect the presence of incomplete definition.

1) No actor is arranged as the participant of an action.

$\{ a : Action \mid a \in actions \wedge \#(\textbf{ran} \{a\} \lhd actionParticipants) = 0\}$

2) No place is allocated for an action.

$\{ a : Action \mid a \in actions \wedge \#(\textbf{ran} \{a\} \lhd actionLocation) = 0\}$

3) How to start an action hasn't been specified.

$\{ a : Action \mid a \in actions \wedge$

$a.scheduledStartTime = \emptyset \wedge a.actionActiveCondition = \emptyset \wedge$
$startPointToAction \rhd \{a\} = planToAction \rhd \{a\} = actionSequence \rhd \{a\} = \emptyset \wedge$
$(\nexists\ aj : AndJoin \mid (aj \mapsto a) \in andJoinToAction) \wedge$
$(\nexists\ oj : OrJoin \mid (oj \mapsto a) \in orJoinToAction) \wedge$
$(\nexists\ as : AndSplit \mid (as \mapsto a) \in andSplitToAction) \wedge$
$(\nexists\ os : OrSplit \mid (os \mapsto a) \in orSplitToAction) \wedge$
$(\nexists\ artifact : Artifact \mid (artifact \mapsto a) \in artifactConsumedByAction)\}$

4) The estimated duration of an action hasn't been specified.

$\{ a : Action \mid a \in actions \wedge a.estimatedDuration = \emptyset\}$

5) The StartNode or EndNode of a plan is missing.

$\{ p : PBLPlan \mid p \in plans \wedge$
$(\#(\mathbf{ran}\ \{p\} \lhd planStartPoint) = 0 \vee \#(\mathbf{ran}\ \{p\} \lhd planEndPoint) = 0) \}$

6) The connection node lacks of connection.

$\{ s : StartPoint \mid s \in \mathbf{ran}\ planStartPoint \wedge$
$(\#(\mathbf{ran}\ \{s\} \lhd startPointToAction) + \#(\mathbf{ran}\ \{s\} \lhd startPointToPlan) +$
$\#(\mathbf{ran}\ \{s\} \lhd startPointToOrJoin) + \#(\mathbf{ran}\ \{s\} \lhd startPointToAndSplit) +$
$\#(\mathbf{ran}\ \{s\} \lhd startPointToOrSplit) = 0) \}$

$\{ e : EndPoint \mid e \in \mathbf{ran}\ planEndPoint \wedge$
$(\#(\mathbf{dom}\ actionToEndPoint \rhd \{e\}) + \#(\mathbf{dom}\ planToEndPoint \rhd \{e\}) +$
$\#(\mathbf{dom}\ andJoinToEndPoint \rhd \{e\}) + \#(\mathbf{dom}\ orJoinToEndPoint \rhd \{e\}) = 0)\}$

$\{ aj : AndJoin \mid aj \in \mathbf{ran}\ andJoinInPlan \wedge$
$((\#(\mathbf{ran}\ \{aj\} \lhd andJoinToAction) + \#(\mathbf{ran}\ \{aj\} \lhd andJoinToPlan) +$
$\#(\mathbf{ran}\ \{aj\} \lhd andJoinToEndPoint) = 0) \vee$
$(\#(\mathbf{dom}\ actionToAndJoin \rhd \{aj\}) + \#(\mathbf{dom}\ planToAndJoin \rhd \{aj\}) = 0))\}$

$\{ oj : OrJoin \mid oj \in \mathbf{ran}\ orJoinInPlan \wedge$
$((\#(\mathbf{ran}\ \{oj\} \lhd orJoinToAction) + \#(\mathbf{ran}\ \{oj\} \lhd orJoinToPlan) +$
$\#(\mathbf{ran}\ \{oj\} \lhd orJoinToEndPoint) = 0) \vee$
$(\#(\mathbf{dom}\ actionToOrJoin \rhd \{oj\}) + \#(\mathbf{dom}\ planToOrJoin \rhd \{oj\}) +$
$\#(\mathbf{dom}\ startPointToOrJoin \rhd \{oj\}) = 0))\}$

$\{ as : AndSplit \mid as \in \mathbf{ran}\ andSplitInPlan \wedge$
$((\#(\mathbf{ran}\ \{as\} \lhd andSplitToAction) + \#(\mathbf{ran}\ \{as\} \lhd andSplitToPlan) = 0) \vee$
$(\#(\mathbf{dom}\ actionToAndSplit \rhd \{as\}) + \#(\mathbf{dom}\ planToAndSplit \rhd \{as\}) +$

$\#(\textbf{dom}\ startPointToAndSplit \rhd \{as\}) = 0))\}$

$\{ os : OrSplit \mid os \in \textbf{ran}\ orSplitInPlan \wedge$

$((\#(\textbf{ran}\ \{os\} \lhd orSplitToAction) + \#(\textbf{ran}\ \{os\} \lhd orSplitToPlan) = 0) \vee$

$(\#(\textbf{dom}\ actionToOrJoin \rhd \{os\}) + \#(\textbf{dom}\ planToOrJoin \rhd \{os\}) +$

$\#(\textbf{dom}\ startPointToOrJoin \rhd \{os\}) = 0))\}$

7) Isolated artifact

$\{ a : Artifact \mid a \in artifacts \wedge$

$(\#(\textbf{dom}\ actionProduceArtifact \rhd \{a\}) + \#(\textbf{dom}\ actionSharedArtifact \rhd \{a\}) +$

$\#(\textbf{ran}\ \{a\} \lhd artifactConsumedByAction) = 0)\}$

4.6.4.2 Potential conflicts

The situations of conflicts are listed below:

1) An actor is assigned to two or more actions that overlap in time and are performed in the synchronous collaboration mode.

$\{ a : Actor \mid a \in actors \wedge$

$(\exists\ action_1, action_2 : Action \mid action_1 \neq action_2 \wedge$

$action_1 \in actions \wedge action_2 \in actions \wedge$

$action_1 \textbf{.}\ collaborationMode =$

$action_2 .\ CollaborationMode = synchronousSession \wedge$

$((action_1 \textbf{.}\ scheduledStartTime \leq action_2 \textbf{.}\ scheduledStartTime \leq$

$action_1 \textbf{.}\ scheduledStartTime + action_1 \textbf{.}\ estimatedDuration) \vee$

$(action_2 \textbf{.}\ scheduledStartTime \leq action_1 \textbf{.}\ scheduledStartTime \leq$

$action_2 \textbf{.}\ scheduledStartTime + action_2 \textbf{.}\ estimatedDuration)) \wedge$

$((actor\ a) \in \textbf{ran}\ \{action_1\} \lhd actionParticipants \vee$

$(\exists\ g : Group \mid a\ \_belongTo\_\ g \wedge$

$(group\ g) \in \textbf{ran}\ \{action_1\} \lhd actionParticipants)) \wedge$

$((actor\ a) \in \textbf{ran}\ \{action_2\} \lhd actionParticipants \vee$

$(\exists\ g : Group \mid a\ \_belongTo\_\ g \wedge$

$(group\ g) \in \textbf{ran}\ \{action_2\} \lhd actionParticipants)) \}$

Notes that ‘group’ is a function which turns a variable g into an agent. Same for ‘actor’ function. Both functions are defined as branches of free type Agent (see definition “Agent” in section 4.3.3.1).

2) A place is allocated for two or more actions that overlap in time.

$\{ p : \text{Place} \mid p \in \text{places} \wedge$

$(\exists\ \text{action}_1, \text{action}_2 : \text{Action} \mid \text{action}_1 \neq \text{action}_2 \wedge$

$\text{action}_1 \in \text{actions} \wedge \text{action}_2 \in \text{actions} \wedge$

$(\text{action}_1 \mapsto p) \in \text{actionLocation} \wedge (\text{action}_2 \mapsto p) \in \text{actionLocation} \wedge$

$(\ (\text{action}_1 .\ \text{scheduledStartTime} \leq \text{action}_2 .\ \text{scheduledStartTime} \leq$

$\text{action}_1 .\ \text{scheduledStartTime} + \text{action}_1 .\ \text{estimatedDuration}\ ) \vee$

$(\ \text{action}_2 .\ \text{scheduledStartTime} \leq \text{action}_1 .\ \text{scheduledStartTime} \leq$

$\text{action}_2 .\ \text{scheduledStartTime} + \text{action}_2 .\ \text{estimatedDuration}\ )\ )\ \}$

3) An action is scheduled to start before the preceding action doesn't start.

$\{ \text{action}_1, \text{action}_2 : \text{Action} \mid \text{action}_1 \neq \text{action}_2 \wedge$

$\text{action}_1 \in \text{actions} \wedge \text{action}_2 \in \text{actions} \wedge$

$((\text{action}_1 \mapsto \text{action}_2) \in \text{actionSequence} \vee$

$(\exists\ aj : \text{AndJoin} \mid (\text{action}_1 \mapsto aj) \in \text{actionToAndJoin} \wedge$
$(aj \mapsto \text{action}_2) \in \text{andJoinToAction}\ ) \vee$

$(\exists\ oj : \text{OrJoin} \mid (\text{action}_1 \mapsto oj) \in \text{actionToOrJoin} \wedge$
$(oj \mapsto \text{action}_2) \in \text{orJoinToAction}) \vee$

$(\exists\ as : \text{AndSplit} \mid (\text{action}_1 \mapsto as) \in \text{actionToAndSplit} \wedge$
$(as \mapsto \text{action}_2) \in \text{andSplitToAction}) \vee$

$(\exists\ os : \text{OrSplit} \mid (\text{action}_1 \mapsto os) \in \text{actionToOrSplit} \wedge$
$(os \mapsto \text{action}_2) \in \text{orSplitToAction})\ ) \wedge$

$(\text{action}_1.\text{scheduledStartTime} > \text{action}_2 .\ \text{scheduledStartTime} \bullet \text{action}_2 \}$

4) An action is scheduled to start before a consumed artifact can be produced.

$\{ \text{action}_1, \text{action}_2 : \text{Action} \mid \text{action}_1 \neq \text{action}_2 \wedge$

$\text{action}_1 \in \text{actions} \wedge \text{action}_2 \in \text{actions} \wedge$

$(\exists\ \text{artifact} : \text{Artifact} \mid (\text{action}_1 \mapsto \text{artifact}) \in \text{actionProduceArtifact} \wedge$
$(\text{artifact} \mapsto \text{action}_2) \in \text{artifactConsumedByAction} \wedge$

$(\text{action}_1.\text{scheduledStartTime} > \text{action}_2 .\ \text{scheduledStartTime} \bullet \text{action}_2 \}$

The above incomplete definition and potential conflicts can be detected by the system. Learners can define a learning plan interactively. That is, after modifying and refining the preliminary learning plan, they can require the system to display the incomplete definition and potential conflicts, and then modify and refine the learning plan again and again until a satisfied learning plan is produced. It is important to note that it is allowed to modify and refine a learning plan in this way when the learning plan is already executed. The next section presents the execution of a defined learning plan.

4.6.5 Execution of PBL-plans

In section 4.6.2, a process framework was presented, which defines the fundamental elements, relationships, constraints of a process, and related operations for constructing a valid learning process. In a virtual institute, multiple learning plans may exist and execute at a point in time, because it is allowed that multiple PBL groups carry out PBL activities concurrently. It is possible that a learner participates in more than one PBL activity and a place will be used by more than one PBL group. The system provides two ways for learners to execute learning plans and get information about learning plans. Firstly, learners can execute and monitor a learning process from the PBL-plan definition tool (see subsection 6.1.6) in which all actions, sub-plans, and more detailed information of the learning plan are organized and displayed as a hierarchical diagram. Secondly, there is a calendar in each home and each public room (see subsection 4.3.3.1). The calendar in a home lists all actions of which the owner of the home is a participant. The calendar in a public room lists all actions of which the public room is the location. The calendar can also be used to schedule isolated actions.

Normally, the organizer of a PBL activity defines a root PBL-plan and is responsible for executing the root learning plan. As a learning plan is executed, the state changes of the learning plan can be observed in the PBL-plan definition tool and in calendars. The documents defined as artifacts in the learning plan will be transferred from one place to another place by using message-boxes.

A PBL-plan can be executed by using the following operation. When a PBL-plan starts to execute, the plan and its sub-plans that are connected directly or indirectly to the StartPoint of the plan will change their state to “active”. All actions that are connected directly or indirectly to the StartPoint of the plan change their state to “enabled”.

```
┌─ StartPlanOK ─────────────────────────────────
│ Δ PlanBase
│ Δ Plans
│ Δ Actions
│ Ξ Artifacts
│ Ξ ConnectionNodes
│ Ξ TemporalRelations
│ plan? : Plan
├───────────────────────────────
│ plan? ∈ plans ∧ currentPlanState plan? = defined
│
│ plans’ = plans
│ currentPlanState’ = currentPlanState ⊕ {plan? ↦ active} ⊕ {
        ∀ p : plans | (p ↦ plan?) ∈ parentPlan ∧
        (((planStartPoint plan?) ↦ p) ∈ startPointToPlan ∨
        (∃ as : AndSplit | (oj ↦ plan?) ∈ andSplitInPlan ∧
                ((planStartPoint plan?) ↦ as) ∈ startPointToAndSplit ∧
                (as ↦ p) ∈ andSplitToPlan ) ∨
        (∃ os : OrSplit | (os ↦ plan?) ∈ orSplitInPlan ∧
                ((planStartPoint plan?) ↦ os) ∈ startPointToOrSplit ∧
                (os ↦ p) ∈ orSplitToPlan ) ∨
        (∃ oj : OrJoin | (oj ↦ plan?) ∈ orJoinInPlan ∧
                ((planStartPoint plan?) ↦ oj) ∈ startPointToOrJoin ∧
                (oj ↦ p) ∈ orJoinToPlan ) ) •
        p ↦ active}
actions' = actions
currentActionState' = currentActionState ⊕ { ∀ a : Action |
        a ∈ actions ∧ (a ↦ plan?) ∈ actionInPlan ∧
        ((planStartPoint plan?) ↦ a) ∈ startPointToAction ∨
        (∃ as : AndSplit | (oj ↦ plan?) ∈ andSplitInPlan ∧
                ((planStartPoint plan?) ↦ as) ∈ startPointToAndSplit ∧
                (as ↦ a) ∈ andSplitToAction ) ∨
        (∃ os : OrSplit | (os ↦ plan?) ∈ orSplitInPlan ∧
                ((planStartPoint plan?) ↦ os) ∈ startPointToOrSplit ∧
                (os ↦ a) ∈ orSplitToAction ) ∨
        (∃ oj : OrJoin | (oj ↦ plan?) ∈ orJoinInPlan Action
                ((planStartPoint plan?) ↦ oj) ∈ startPointToOrJoin ∧
                (oj ↦ a) ∈ orJoinToAction ) ) ∧
        (∀ subPlan : PBLPlan | subPlan ∈ { p : PBLPlan |
                (p ↦ plan?) ∈ parentPlan ∧
                (((planStartPoint plan?) ↦ p) ∈ startPointToPlan ∨
                (∃ as : AndSplit | (oj ↦ plan?) ∈ andSplitInPlan ∧
                        ((planStartPoint plan?) ↦ as) ∈ startPointToAndSplit ∧
                        (as ↦ p) ∈ andSplitToPlan ) ∨
                (∃ os : OrSplit | (os ↦ plan?) ∈ orSplitInPlan ∧
                        ((planStartPoint plan?) ↦ os) ∈ startPointToOrSplit ∧
                        (os ↦ p) ∈ orSplitToPlan ) ∨
                (∃ oj : OrJoin | (oj ↦ plan?) ∈ orJoinInPlan ∧
                        ((planStartPoint plan?) ↦ oj) ∈ startPointToOrJoin ∧
                        (oj ↦ p) ∈ orJoinToPlan ) ) } ∧
                (a ↦ subPlan) ∈ actionInPlan ∧
```

```
|               ((planStartPoint subPlan) ↦ a) ∈ startPointToAction ∨
|               (∃ as : AndSplit | (oj ↦ subPlan) ∈ andSplitInPlan ∧
|                       ((planStartPoint subPlan) ↦ as) ∈ startPointToAndSplit ∧
|                       (as ↦ a) ∈ andSplitToAction ) ∨
|               (∃ os : OrSplit | (os ↦ subPlan) ∈ orSplitInPlan ∧
|                       ((planStartPoint subPlan) ↦ os) ∈ startPointToOrSplit ∧
|                       (os ↦ a) ∈ orSplitToAction ) ∨
|               (∃ oj : OrJoin | (oj ↦ subPlan) ∈ orJoinInPlan Action
|                       ((planStartPoint subPlan) ↦ oj) ∈ startPointToOrJoin ∧
|                       (oj ↦ a) ∈ orJoinToAction ) ) ) •
|       a ↦ enabled}
| actionInPlan’ = actionInPlan
| actionParticipants’ = actionParticipants
| actionLocation’ = actionLocation
└──────────────────────────────────────────────────────────────
```

If the state of an action is ‘enabled’, learners can enact the action manually by performing the operation ‘start’. The state of the action turns into active or suspended according to the definition of the action. If the active-condition of the action is not met, the state of the action becomes suspended. For example, if the consumed artifacts haven’t been delivered or not all participants are presented in the location of the action. If the enacted action is connected by an OrSplit node that connects to some other actions, then the state of those actions turns into suspended. The OrSplit node is used in the situation that several scheduled actions can be performed to achieve the same goal. If one of them is performed in the execution process, it is not necessary to perform one of the other actions. When an action starts, the state of the produced artifacts of this action turns into inEditing.

```
┌─ StartActionOK──────────────────────────────────────────────
| Δ Actions
| Δ Artifacts
| a? : Action
├──────────────────────────────────
| a? ∈ actions ∧ currentActionState a? = enabled
|
| a?.actualStartTime = now
| actions’ = actions
| currentActionState’ = currentActionState ⊕ {
|       if a?.actionActiveCondition = true
|               then (a? ↦ active)
|               else (a? ↦ suspended)} ⊕ {
|       ∀ a : Action | a ∈ actions ∧ currentActionState a = enabled ∧
|       (∃ orSplit: OrSplit | (orSplit ↦ a?) ∈ OrSplitToAction ∧
|               (orSplit ↦ a) ∈ OrSplitToAction ) •
```

```
        a ↦ suspended}
actionInPlan' = actionInPlan
actionParticipants' = actionParticipants
actionLocation' = actionLocation
artifacts' = artifacts
artifactInPlan' = artifactInPlan
currentArtifactState' = currentArtifactState ⊕
        { ∀  a : Artifact | (a? ↦ a) ∈ actionProduceArtifact • a ↦ inEditing } ⊕
        { ∀  a : Artifact | (a? ↦ a) ∈ actionSharedArtifact ∧
         (a ↦ created) ∈ currentArtifactState  • a ↦ inEditing }
actionProduceArtifact' = actionProduceArtifact
artifactConsumedByAction' = artifactConsumedByAction
actionSharedArtifact' = actionSharedArtifact
```

For an action whose collaboration mode is a synchronous session, the action is enacted when all participants are presented in the location of the action. Meanwhile, the state of artifacts that are produced or shared by this action turns into inEditing.

```
── JoinActionOK ─────────────────────────────
Δ Actions
Δ Artifacts
Δ VirtualInstitute
actor? : Actor
action? : Action
──────────────────────────
action? ∈ actions ∧ currentActionState a? = suspended ∧
action?.collaborationMode = synchronousSession ∧
actor? ∈ actors ∧ ( actor ↦ actionLocation action?) ∉ actorLocation ∧
(∃ agent : Agent | (action? ↦ agent) : actionParticipants •
        actor actor? = agent ∨
        (∃ g : Group | actor? _belongTo_ g ∧ group g = agent))

actorLocation' = actorLocation ⊕ {actor ↦ actionLocation action?}
actions' = actions
currentActionState' = currentActionState ⊕  {
        if a?.actionActiveCondition = true
                then (action? ↦ active) }
actionInPlan' = actionInPlan
actionParticipants' = actionParticipants
actionLocation' = actionLocation
artifacts' = artifacts
artifactInPlan' = artifactInPlan
currentArtifactState' = currentArtifactState ⊕
        { ∀  a : Artifact | (action? ↦ a) ∈ actionProduceArtifact ∧
                action?.actionActiveCondition = true • a ↦ inEditing } ⊕
        { ∀  a : Artifact | (action? ↦ a) ∈ actionSharedArtifact ∧
                action?.actionActiveCondition = true ∧
                (a ↦ created) ∈ currentArtifactState  • a ↦ inEditing }
actionProduceArtifact’ = actionProduceArtifact
artifactConsumedByAction’ = artifactConsumedByAction
actionSharedArtifact’ = actionSharedArtifact
```

For an action in which some artifacts will be produced, delivering an artifact will trigger the actions that are waiting for consuming the artifact. If the terminated-condition of an action is true and all produced artifacts are delivered, the event of delivering artifacts will trigger the change of the state of the actions.

```
── ActionDeliverArtifactOK ──────────────────────
Δ Artifacts
Δ Actions
a? : Action
artifact? : Artifact
────────────────────────
a? ∈ actions ∧ currentActionState a? = active ∧
artifact? ∈ artifacts ∧ currentArtifactState artifact? = inEditing ∧
((a? ↦ artifact?) ∈ actionProduceArtifact ∨
(a? ↦ artifact?) ∈ actionSharedArtifact)

artifacts’ = artifacts
artifactInPlan’ = artifactInPlan
currentArtifactState’ = currentArtifactState ⊕ { artifact? ↦ finished }
actionProduceArtifact’ = actionProduceArtifact
artifactConsumedByAction’ = artifactConsumedByAction
actionSharedArtifact’ = actionSharedArtifact
actions’ = actions
currentActionState’ = currentActionState ⊕
      {∀ a : Action | a ∈ actions ∧ currentActionState a = suspended ∧
      (artifact? ↦ a) ∈ artifactConsumedByAction ∧
      a.actionActiveCondition = true •
      a? ↦ active }
actionInPlan’ = actionInPlan
actionParticipants’ = actionParticipants
actionLocation’ = actionLocation
──────────────────────────────────────────
```

```
── DeliverArtifactTerminateActionOK ──────────────────────
Δ Artifacts
Δ Actions
a? : Action
artifact? : Artifact
────────────────────────
```

```
a? ∈ actions ∧ currentActionState a? = active ∧
artifact? ∈ artifacts ∧ currentArtifactState artifact? = inEditing ∧
((a? ↦ artifact?) ∈ actionProduceArtifact ∨
(a? ↦ artifact?) ∈ actionSharedArtifact) ∧

artifacts' = artifacts
artifactInPlan' = artifactInPlan
currentArtifactState' = currentArtifactState ⊕ { artifact? ↦ finished }
actionProduceArtifact' = actionProduceArtifact
artifactConsumedByAction' = artifactConsumedByAction
actionSharedArtifact' = actionSharedArtifact
actions' = actions
currentActionState' = currentActionState
    (if a?.terminatedCondition = true then { a? ↦ finished }) ⊕ {
    ∀ a : Action | a ∈ actions ∧ currentActionState a = defined ∧
    a?.terminatedCondition = true ∧
    ((a?↦ a) ∈ actionSequence ∨
    (∃ andJoin : AndJoin | (a? ↦ andJoin) ∈ actionToAndJoin ∧
            (andJoin ↦ a) ∈ andJoinToAction ∧
            (∀ action : Action | a ∈ dom actionToAndJoin ▷ {andJoin}•
            currentActionState action = finished)) ∨
    (∃ orJoin : OrJoin | (a? ↦ orJoin) ∈ actionToOrJoin ∧
            (orJoin ↦ a) ∈ orJoinToAction ) ∨
    (∃ andSplit : AndSplit | (a? ↦ andSplit) ∈ actionToAndSplit ∧
            (andSplit ↦ a) ∈ andSplitToAction ) ∨
    (∃ orSplit : OrSplit | (a? ↦ orSplit) ∈ actionToOrSplit ∧
            (orSplit ↦ a) ∈ orSplitToAction ) ∨
    (∀ subPlan : PBLPlan |
             (a? ↦ subPlan) ∈ actionToPlan ∧
            ((planStartPoint subPlan) ↦ a) ∈ startPointToAction ∨
            (∃ as : AndSplit | (oj ↦ subPlan) ∈ andSplitInPlan ∧
                    ((planStartPoint subPlan) ↦ as) ∈ startPointToAndSplit ∧
                    (as ↦ a) ∈ andSplitToAction ) ∨
            (∃ os : OrSplit | (os ↦ subPlan) ∈ orSplitInPlan ∧
                    ((planStartPoint subPlan) ↦ os) ∈ startPointToOrSplit ∧
                    (os ↦ a) ∈ orSplitToAction ) ∨
            (∃ oj : OrJoin | (oj ↦ subPlan) ∈ orJoinInPlan Action
                    ((planStartPoint subPlan) ↦ oj) ∈ startPointToOrJoin ∧
                    (oj ↦ a) ∈ orJoinToAction ) ) ) •
    a ↦ enabled ⊕ {
    {∀ a : Action | a ∈ actions ∧ currentActionState a = suspended ∧
```

```
        (artifact? ↦ a) ∈ artifactConsumedByAction ∧
        a.actionActiveCondition = true •
        a ↦ active }
actionInPlan' = actionInPlan
actionParticipants' = actionParticipants
actionLocation' = actionLocation
```

Learners can terminate an action manually. If the terminated action is not the last action in the plan, the subsequent actions will be enacted according to the definition of the plan. Otherwise, the plan will be finished.

```
── TerminateActionOK ──────────────────────────
Δ Actions
a? : Action
───────────────────
a? ∈ actions ∧ currentActionState a? = active ∧
( ∄ endPoint : EndPoint |
        (a? ↦ endPoint) ∈ actionToEndPoint ) ∨
        ( ∄ andJoin : AndJoin |
                (a?↦ andJoin) ∈ actionToAndJoin ∧
                (andJoin ↦ endPoint) ∈ andJoinToEndPoint ∧
                (∀ a : Action | a ∈ actions ∧ a ≠ a? ∧
                        currentActionState a = finished ∧
                        (a ↦ andJoin) ∈ actionToAndJoin ) ∧
                (∀ p : PBLPlan | p ∈ plans ∧
                        currentPlanState p = finished ∧
                        (p ↦ andJoin) ∈ planToAndJoin )) ∨
        ( ∄ orJoin : OrJoin |
                (a?↦ orJoin) ∈ actionToOrJoin ∧
                (orJoin ↦ endPoint) ∈ orJoinToEndPoint ) )

actions' = actions
currentActionState' = currentActionState ⊕ { a? ↦ finished } ⊕
        { ∀ a : Action | a ∈ actions ∧ currentActionState a = defined ∧
        ((a?↦ a) ∈ actionSequence ∨
        (∃ andJoin : AndJoin | (a? ↦ andJoin) ∈ actionToAndJoin ∧
                (andJoin ↦ a) ∈ andJoinToAction ∧
                (∀ action : Action | a ∈ dom actionToAndJoin ▷ {andJoin}•
                currentActionState action = finished)) ∨
        (∃ orJoin : OrJoin | (a? ↦ orJoin) ∈ actionToOrJoin ∧
                (orJoin ↦ a) ∈ orJoinToAction ) ∨
        (∃ andSplit : AndSplit | (a? ↦ andSplit) ∈ actionToAndSplit ∧
│                (andSplit ↦ a) ∈ andSplitToAction ) ∨
│         (∃ orSplit : OrSplit | (a? ↦ orSplit) ∈ actionToOrSplit ∧
│                (orSplit ↦ a) ∈ orSplitToAction ) ∨
│         (∀ subPlan : PBLPlan |
│                (a? ↦ subPlan) ∈ actionToPlan ∧
│                ((planStartPoint subPlan) ↦ a) ∈ startPointToAction ∨
│                (∃ as : AndSplit | (oj ↦ subPlan) ∈ andSplitInPlan ∧
│                       ((planStartPoint subPlan) ↦ as) ∈ startPointToAndSplit ∧
│                       (as ↦ a) ∈ andSplitToAction ) ∨
│                (∃ os : OrSplit | (os ↦ subPlan) ∈ orSplitInPlan ∧
│                       ((planStartPoint subPlan) ↦ os) ∈ startPointToOrSplit ∧
│                       (os ↦ a) ∈ orSplitToAction ) ∨
│                (∃ oj : OrJoin | (oj ↦ subPlan) ∈ orJoinInPlan Action
│                       ((planStartPoint subPlan) ↦ oj) ∈ startPointToOrJoin ∧
│                       (oj ↦ a) ∈ orJoinToAction ) ) ) •
│         a ↦ enabled}
│ actionInPlan' = actionInPlan
│ actionParticipants' = actionParticipants
│ actionLocation' = actionLocation
└──────────────────────────────────────────────────────────

┌─ TerminateLastActionOfPlanOK──────────────────────────────
│ Δ Actions
│ Δ Plans
│ a? : Action
├──────────────────────────────────────
│ a? ∈ actions ∧ currentActionState a? = active ∧
│ (∃ endPoint : EndPoint |
│         (a? ↦ endPoint) ∈ actionToEndPoint ∨
│         (∃ andJoin : AndJoin |
│                (a?↦ andJoin) ∈ actionToAndJoin ∧
│                (andJoin ↦ endPoint) ∈ andJoinToEndPoint ∧
│                (∀ a : Action | a ∈ actions ∧ a ≠ a? ∧
│                       currentActionState a = finished ∧
│                       (a ↦ andJoin) ∈ actionToAndJoin ) ∧
│                (∀ p : PBLPlan | p ∈ plans ∧
│                       currentPlanState p = finished ∧
│                       (p ↦ andJoin) ∈ planToAndJoin )) ∨
│         (∃ orJoin : OrJoin |
│                (a?↦ orJoin) ∈ actionToOrJoin ∧
│                (orJoin ↦ endPoint) ∈ orJoinToEndPoint ) )
│
```

```
plans’ = plans
currentPlanState’ = currentPlanState ⊕ {actionInPlan a? ↦ finished}
actions’ = actions
currentActionState’ = currentActionState ⊕ { a? ↦ finished }
actionInPlan’ = actionInPlan
actionParticipants’ = actionParticipants
actionLocation’ = actionLocation
```

If time is specified as a factor of the active-condition or terminated-condition of an action in the definition of a plan, the event that the scheduled time is coming will trigger the change of the state of the actions and related artifacts.

```
── TimeTriggerActionActiveOK ──
Δ Actions
Δ Artifacts
a? : Action
────────
a? ∈ actions ∧ a?.scheduledStartTime = now

a?.actualStartTime = now
actions’ = actions
currentActionState’ = currentActionState ⊕ {
      if a?.actionActiveCondition = true
            then (a? ↦ active)
            else (a? ↦ suspended)} ⊕
      { ∀ a : Action | a ∈ actions ∧ currentActionState a = enabled ∧
      (∃ orSplit: OrSplit | (orSplit ↦ a?) ∈ OrSplitToAction ∧
            (orSplit ↦ a) ∈ OrSplitToAction ) •
      a ↦ suspended}
actionInPlan’ = actionInPlan
actionParticipants’ = actionParticipants
actionLocation’ = actionLocation
artifacts’ = artifacts
artifactInPlan’ = artifactInPlan
currentArtifactState’ = currentArtifactState ⊕
      { ∀ a : Artifact | (a? ↦ a) ∈ actionProduceArtifact • a ↦ inEditing } ⊕
      { ∀ a : Artifact | (a? ↦ a) ∈ actionSharedArtifact ∧
      (a ↦ created) ∈ currentArtifactState • a ↦ inEditing }
actionProduceArtifact’ = actionProduceArtifact
artifactConsumedByAction’ = artifactConsumedByAction
actionSharedArtifact’ = actionSharedArtifact
```

```
┌─ TimeTriggerActionTerminatedOK ─────────────────────────
│ Δ Actions
│ a? : Action
├──────────────────────────────────
│ a? ∈ actions ∧ currentActionState a? = active ∧
│ a?.actualStartTime + a?.estimatedDuration = now ∧
│ a?.actionTerminateCondition = true ∧
│ ( ∄ endPoint : EndPoint | (a? ↦ endPoint) ∈ actionToEndPoint ) ∧
│       ( ∄ andJoin : AndJoin |
│               (a?↦ andJoin) ∈ actionToAndJoin ∧
│               (andJoin ↦ endPoint) ∈ andJoinToEndPoint ∧
│               (∀ a : Action | a ∈ actions ∧ a ≠ a? ∧
│                       currentActionState a = finished ∧
│                       (a ↦ andJoin) ∈ actionToAndJoin ) ∧
│               (∀ p : PBLPlan | p ∈ plans ∧
│                       currentPlanState p = finished ∧
│                       (p ↦ andJoin) ∈ planToAndJoin )) ∨
│       ( ∄ orJoin : OrJoin |
│               (a?↦ orJoin) ∈ actionToOrJoin ∧
│               (orJoin ↦ endPoint) ∈ orJoinToEndPoint ) )
│
│ actions’ = actions
│ currentActionState’ = currentActionState ⊕ { a? ↦ finished } ⊕
│       { ∀ a : Action | a ∈ actions ∧ currentActionState a = defined ∧
│       ((a?↦ a) ∈ actionSequence ∨
│       (∃ andJoin : AndJoin | (a? ↦ andJoin) ∈ actionToAndJoin ∧
│               (andJoin ↦ a) ∈ andJoinToAction ∧
│               (∀ action : Action | a ∈ dom actionToAndJoin ▷ {andJoin}•
│               currentActionState action = finished)) ∨
│       (∃ orJoin : OrJoin | (a? ↦ orJoin) ∈ actionToOrJoin ∧
│               (orJoin ↦ a) ∈ orJoinToAction ) ∨
│       (∃ andSplit : AndSplit | (a? ↦ andSplit) ∈ actionToAndSplit ∧
│               (andSplit ↦ a) ∈ andSplitToAction ) ∨
│       (∃ orSplit : OrSplit | (a? ↦ orSplit) ∈ actionToOrSplit ∧
│               (orSplit ↦ a) ∈ orSplitToAction ) ∨
│       (∀ subPlan : PBLPlan |
│                (a? ↦ subPlan) ∈ actionToPlan ∧
│               ((planStartPoint subPlan) ↦ a) ∈ startPointToAction ∨
│               (∃ as : AndSplit | (oj ↦ subPlan) ∈ andSplitInPlan ∧
│                       ((planStartPoint subPlan) ↦ as) ∈ startPointToAndSplit ∧
│                       (as ↦ a) ∈ andSplitToAction ) ∨
|          (∃ os : OrSplit | (os ↦ subPlan) ∈ orSplitInPlan ∧
|                ((planStartPoint subPlan) ↦ os) ∈ startPointToOrSplit ∧
|                (os ↦ a) ∈ orSplitToAction ) ∨
|          (∃ oj : OrJoin | (oj ↦ subPlan) ∈ orJoinInPlan Action
|                ((planStartPoint subPlan) ↦ oj) ∈ startPointToOrJoin ∧
|                (oj ↦ a) ∈ orJoinToAction ) ) ) •
|    a ↦ enabled}
| actionInPlan' = actionInPlan
| actionParticipants' = actionParticipants
| actionLocation' = actionLocation
|______________________________________________________________
```

4.6.6 Related Work and Discussion

Workflow systems are a way of routing information objects among users, and to specify automatic actions to be taken in that routing typically according to certain process models [Winograd86] [Ellis94] [Abbott94] [Schael96]. Normally, in order to model the workflow in organizations, there are two dominating paradigms: The Customer-Supplier (CS) paradigm and the Input-Process-Output (IPO) paradigm.

The CS paradigm focuses on coordination among people. The typical workflow systems in this paradigm are Coordinator [Bullen91] and ActionWorkflow [Medina-Mora92]. This paradigm suits for explicitly modeling the chain of commitments that exists between people in order to satisfy the customer. However, the specific activities carried out in order to meet the contract are not modeled. In addition, the information produced and needed in each activity is not described. Thus, it is not suitable to model session-based collaboration processes.

In the IPO paradigm the workflow is regarded as a chain of actions that takes information as input and produces information as output. This approach was first applied to coordination problems in software engineering and office information automation. The approach takes the view that process descriptions should be thought of and implemented as software [Osterweil87]. The definition of the process (process model) includes the description of the resources used in the process (humans, tools, etc.), the policies followed in the process, the actions in which the process is structured, and any other information useful to characterize the process. The purposes of the process models are summarized in [Curtis92], ranging from understanding aids to automate execution support. A process model can be described by using a process modeling language as a network of asynchronous sub-processes or steps that are loosely coupled and which need from time to time to communicate between the group members engaged in the process. The overall pattern of these exchanges reflects the cooperative structure of the modeled process. Developing a process model involves decomposing cooperative work into activities and roles, and defining the dependencies between roles. Activities may be carried out either manually or by using software tools embedded in the support environment. A role refers to a logically coherent collection of obligations and responsibilities related to the achievement of a defined goal. The dependencies between roles are referred to as interactions. Such dependencies normally reflect the need of two roles to exchange information.

Many products of workflow management systems are already on the market with a different set of features and different degrees of support [Georgakopoulos95]. The Workflow Management Coalition (WfMC) was founded to define standards for terminology and interfaces of workflow management systems [WfMC]. The approach described in this chapter follows the framework proposed by the WfMC to support session-based collaborative processes. However, session-based collaborative processes have some distinguished characteristics that business processes have not. Therefore, existing workflow management products can not be exploited directly. This subsection compares the approach described in this section with existing workflow management products.

Firstly, in session-based collaborative processes, an action is carried out in a synchronous or asynchronous session that provides a shared workspace. The participants of the action collaboratively work to achieve the common goal by exploiting the tools and documents available in the shared workspace. Participants with the same role and even with different roles (e.g., teacher and learner) can perform the same action. In other workflow systems, an activity (or a work step) is defined as a process element that is performed by an individual performer manually or by means of a specific application tools. The shared workspace of an activity is not explicitly modeled. Instead of modeling a shared workspace, for each activity exactly one role is defined whose members can invoke a specific application tool to perform the tasks. Even if multiple performers with the same role are engaged in the same step, they deal with different work items individually.

Secondly, in session-based collaborative processes, the artifacts are defined for sharing. Artifacts produced jointly by participants in a session are maintained permanently within the shared workspace, or are transferred to other virtual places by using message-boxes. Furthermore, some artifacts can be viewed and edited synchronously by the people working on different actions. However, artifacts or documents in other workflow systems are designed for exchange. In the document-centric form of workflow, individual documents flow through a predefined network step by step. In each step the documents may be modified or transformed by one individual with a specific role working at this step. It is impossible that individuals working in different steps access or update the same document at the same point in time.

Thirdly, session-based collaborative processes are ill structured. The actions are enacted in various ways. Sometimes, an action is enacted when all preceding actions are terminated. Sometimes, it starts when necessary artifacts produced by other actions are delivered. Sometimes, it is enacted by the scheduled start time, and sometimes it starts when all participants join the action. In other workflow systems, activities are enacted by either temporal sequence or document routine.

Fourthly, in session-based collaborative processes, members of the group often work as a whole. The work procedure is defined collaboratively by the group and executed by the same group within the work processes. They frequently meet to exchange informal information, to carry out their substantive tasks, to develop or revise their work plan, to assign tasks to individuals or sub-groups, and to perform tasks individually or in sub-groups according to the defined work plan. In other workflow systems, the process models are normally developed by business process experts before the process models are initiated. An instance of a process model is executed by other people who may don't care and have no knowledge about the whole process model. Some workflow systems allow dynamic alterations to process definition from the run-time operational environment. Examples are EuroCoOp Task Manager [Hennessy92, Busbach93], Regatta [Swenson93], and TeamWARE Flow [TeamWARE], which allow dynamic modification of work plans by end-users on the fly. The term of *collaborative planning* has been used in Regatta [Swenson93]. It refers to means by which a computerized representation of a work procedure is defined and modified cooperatively by a group of people. The planning tool of Regatta supports some form of plan fragmentation where different capabilities can be assigned to different people for different fragments. Each plan fragment could be modified by the owner of the fragment or by delegates. However they can not modify a plan fragment jointly. Furthermore, at any given time workers engaged in a work process are often doing different things. In most cases, a worker always performs the same type of task (for dealing with different work items) one by one.

Fifthly, PBL-specific support (e.g., creating preliminary PBL-plan and interactive modifying and refining PBL-plan) is a distinct feature of the approach described in this thesis.

4.6.7 Summary

Based on the theory of self-directed learning and an analysis of the characteristics of PBL processes, an approach to support PBL processes at the action level was presented in this chapter. Through a comparison with other workflow management systems, the major characteristics of this approach can be summarized as following.

1) The idea of session-based collaborative processes is developed to capture processes that consist of a set of coordinated actions. Each action is executed on a shared workspace by a group of people employing a synchronous or asynchronous collaboration mode.
2) A visual process modeling language is developed for describing problem based learning processes. This visual process modeling language consists of the components of processes (e.g., nested processes, actions, and artifacts), and allows representing process properties, the relations between the components, and constrains.
3) Based on the visual process modeling language, a collaborative tool can be developed to support definition of problem based learning processes as a hypertext document (see chapter 5). In order to ease construction, some mechanisms such as creating a preliminary PBL-plan and interactive modifying and refining a PBL-plan are provided.
4) A cooperative environment is provided to execute these cooperative processes by a team. Learners can join an action by selecting an action item from calendars or from the hypertext document that represents the defined PBL-plan. Learners can also manipulate and monitor the state of actions by using calendars or the hypertext document. An action can be enacted by using a various ways. Multiple

participants can collaboratively perform an action by using shared tools and documents available in the shared workspace of the action. The artifacts can be transferred from one place to others by using message-boxes according to the definition of the PBL-plan.

5 Implementation

In chapter 4, an approach to design a collaborative PBL environment has been presented. This chapter describes how this approach is implemented. Section 5.1 describes the system architecture of the virtual PBL environment described in this thesis. Section 5.2 presents an abstract implementation model of the virtual PBL environment, which delineates architectural components and communication between these components, including software modules, users' data, and control information. Then how the abstract implementation model is mapped on the system architecture is described. Section 5.3 describes how to implement the system architecture. Section 5.4 presents how cooperative hypermedia technology is used to implement a prototype system.

5.1 System Architecture

A collaborative virtual PBL enrionment is a groupware application system that enables geographically separated and co-located people to conduct synchronous and asynchronous PBL activities. Such a system manages a collection of shared information objects and communication channels through which users can interact with each other. It support the real-time presentation and manipulation of shared information so that users can see other users' operations as reflected in changes to the shared information. We choose a client/server communication architecture, in which each application instance has a communication channel with a central server. Any update event to the shared information is transferred among application instances through the server. Comparison with the distributed communication architecture that provides one communication channel for every pair of application instances, the client/server communication architecture is easy to implement and manage. In addition, traffic on the network is lower. However, a disadvantage of client/server communication architecture is that the server is a bottleneck when the number of clients increases. It is not a serious problem for a collaborative virtual PBL environment, because most PBL programs use small group approach. As Woods noted: in the McMaster Medical School model of PBL, "the tutor is a facilitating presence used as needed by a group of 5 to 9 to 'solve the case'" [Woods96].

Figure 5.1 illustrates the system architecture. It consists of multiple clients and a server. A client consists of user interface and application functionality, local shared/replicated data, and communication management. The server consists of communication management, concurrency control, and shared data repository. Each client provides a user interface for users to interact with the application. A user's operation in the user interface results in an event. An event will be handled by the system calling a certain function that may update some shared data objects. Then an update message will be created and sent by the communication management module of the client and will be received by the communication management module of the server. The update message will be processed by the concurrency control module. If the update is allowed, the server will update the shared data objects in its repository. Meanwhile, the server broadcast the update message to all clients, which, in turn, will update their local shared/replicated data objects to keep consistency.

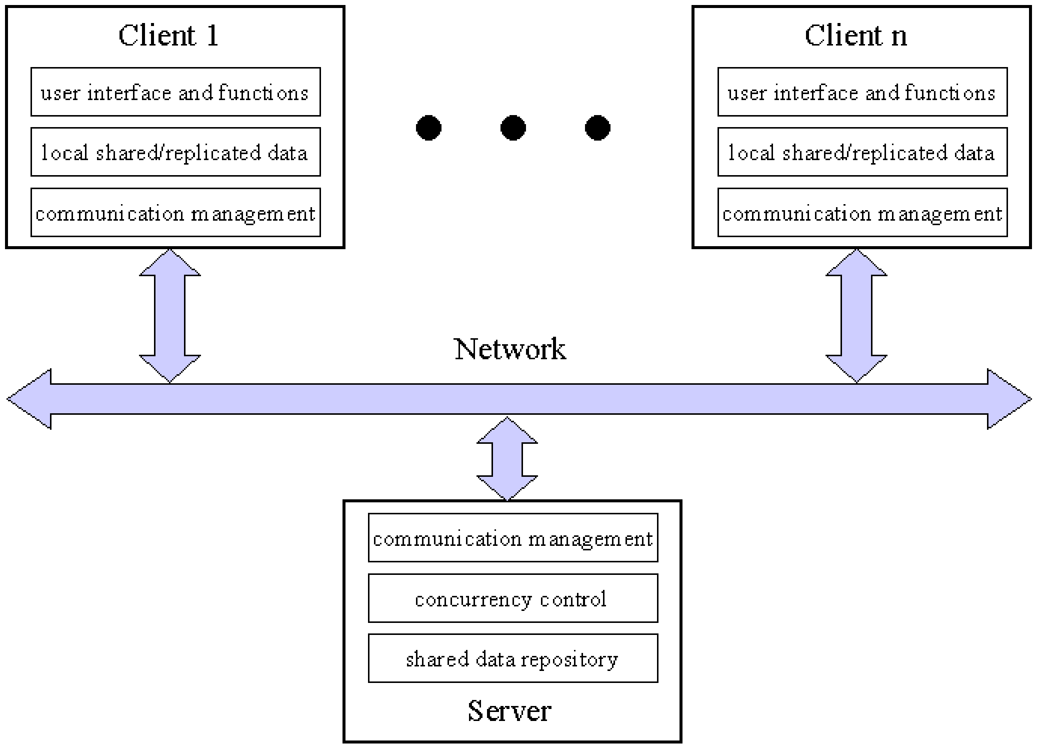


Figure 5.1: System Architecture

5.2 A Mapping from an Abstract Implementation Model to The System Architecture

This section describes the high-level implementation architecture of the virtual learning environment. And then it presents how the components of the abstract implementation model are mapped on the system architecture.

5.2.1 Abstract Implementation Model

The main functional components of the virtual learning environment are illustrated in figure 5.2. The abstract implementation model has three types of components. The first type of components is system definition and control data that are used by one or more software modules (shown in light gray). The second type of components are users' data that represents learning materials and learning recordings (shown unfilled). The third type of components are software modules that provide support for various functions within the virtual learning environment (shown in dark gray). The arrows in Figure 5.2 represent data flows. The roles of the major functional components within the virtual learning environment are described below.

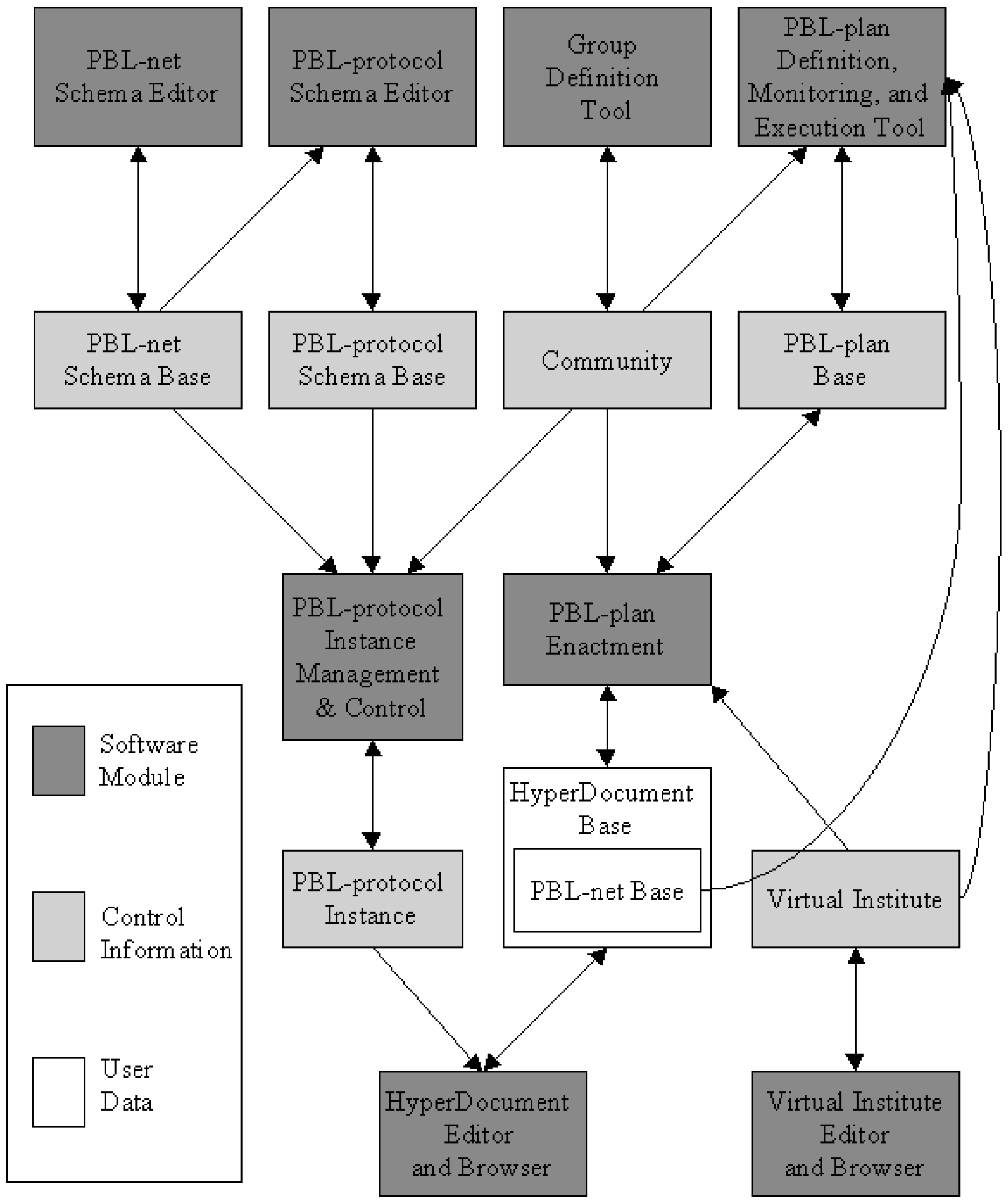


Figure 5.2: Abstract Implementation Model

5.2.1.1 System Definition and Control Data

In the virtual PBL environment, there are six modules of system definition and control data.

5.2.1.1.1 PBL-net Schema Base

The PBL-net schema base is a repository of the defined PBL-net schema (see subsection 4.4.4), which specifies a PBL-specific knowledge representation language.

In addition, further refined PBL-nets can also be stored in this base. Other knowledge representation languages for learning can be stored as schemas in this base as well. A schema contains a set of node types and link types.

5.2.1.1.2 PBL-protocol Schema Base

The PBL-protocol schema base contains all definitions of PBL-protocol schemas (see subsection 4.5.3), which can be instantiated in PBL processes. The important information contained in a PBL-protocol includes protocol states, state transitions, and bound behavior rules.

5.2.1.1.3 Community definition

The community definition component contains the information about all agents (actors and groups) and their relations, including information about the properties of agents as well (see subsection (4.3.3.1.1).

5.2.1.1.4 PBL-plan Base

The PBL-plan base contains all necessary information about PBL-plans in a virtual institute to enable them to be executed by the PBL-plan enactment software. This includes information about sub-plan, constituent actions, artifacts, connection nodes, and their relations. Important information about actions includes goal, participants, scheduled start time, duration, location, active-condition, and terminated-condition. The dynamic information such as the states of actions and current participants is included as well (see subsection 4.6.2).

5.2.1.1.5 PBL-protocol Instance

The PBL-protocol instance component contains information about all PBL-protocol instances (see subsection 4.5.2.2). Each PBL-protocol instance has information such as PBL-protocol, current state, and memberships of each protocol role (i.e., an agent having a specific role).

5.2.1.1.6 Virtual Institute definition

The virtual institute definition component contains information about all virtual institutes associated with all virtual places and their relations. Each place has information such as current actors, installed tools, available documents, and their relations (see subsection 4.3.3.1).

5.2.1.2 Users' Data

There is only one users' data module, which is the hyperdocument base (see below).

5.2.1.2.1 HyperDocument Base (Including PBL-net Base)

The hyperdocument base contains information about all documents created by users and the relationships between these documents. Although the documents in a virtual institute are distributed in different places, they are connected to each other to form hyperdocuments. Each document has information such as the title, owner, topic, state, and hyperlinks to other documents. Furthermore, some information contained in a document represents learning materials and learning recordings in the form of texts, tables, images, and graphics (see subsection 4.3.3.1.3). In addition, a PBL-net is created by a PBL group during a learning process. As a special kind of document, a PBL-net contains typed nodes that serve as hyperlinks to normal documents. A PBL-net and the documents directedly and indirectedly connected to the PBL-net form a hyperdocument, called a PBL hyperdocument (see subsection 4.4.4). All PBL hyperdocuments are also maintained in the hyperdocument base.

5.2.1.3 Software Modules

There are eight software modules.

5.2.1.3.1 PBL-net Schema Editor

The PBL-net schema editor is used to define and modify the PBL-net schemas (see subsection 4.4.3). Users can create and delete node types and link types of the PBL-net schema by using this tool. It is important to note that this tool can also be used to define other graphical knowledge representation schemas. Users can load and delete defined schemas as well by means of this tool.

5.2.1.3.2 PBL-protocol Schema Editor

The PBL-protocol schema editor is used to define PBL-protocol schemas and save them in the PBL-protocol schema base (see subsection4.5.2.1). Users can create and delete protocol states and transitions between protocol states. The behavior rules can be defined and bound to the states and state transitions by using this tool. When defining PBL-protocols, the PBL-net schemas will be used. Users can also load, modify, and delete the protocol schema stored in the PBL-protocol schema base by means of this tool.

5.2.1.3.3 Group Definition Tool

By using the group definition tool, users can define the organizational structure of the agents in a virtual institute (see subsection 4.3.3.2.3). Actors, groups, their individual properties, and the memberships of groups can be defined by using this tool. The definition of the agents and their relations can be stored in the community definition module and retrieved later on.

5.2.1.3.4 PBL-plan Definition, Monitoring, and Execution Tool

The PBL-plan definition, monitoring, and execution tool is primarily used to create the work process description in a computer tractable form as a PBL-plan (see subsection 4.6.2). The PBL-plan defined by using this tool can be stored in the PBL-plan base and be loaded later on. When defining a PBL-plan, information about the community (see the community definition module) and virtual institute (see the virtual institute definition) is used. In particular, information from a PBL-net can be used to create a preliminary PBL-plan. This tool also provides the functionality to guide users to modify and refine PBL-plans. Because each learning group has a unique PBL-plan that can not be reused by other learning groups, this tool doesn't distinguish the process model and process instance. That is, each PBL-plan is regarded as both a process model and its unique instance. Because this tool provides a graphic representation of executed learning plans, users can use it to monitor the state of executed PBL-plan. Some functions for users to execute a PBL-plan such as enact actions and terminate actions are also provided in this tool.

5.2.1.3.5 PBL-protocol Instance Management & Control

The PBL-protocol instance management & control software provides functions to initiate a PBL-protocol instance, to assign agents to protocol roles, to guide and control execution of the protocol instance, and to shift PBL-protocols (see subsection 4.5.2.2 and 4.5.2.3). The information stored in the PBL-net schema base, the PBL-protocol schema base, the community definition, and the PBL-protocol instance is used to determine the state change of protocol instances.

5.2.1.3.6 PBL-plan Enactment

The PBL-plan enactment software interprets and controls the PBL-plans. When an event occurs, for example, a user terminates an action, some changes will be triggered such as enacting a certain action or delivering artifact according to the definition of the executed learning plan (see subsection 4.6.5). Information about PBL-plan definition, community definition, virtual institute definition, and the state of related documents is used to change the state of the PBL-plans and the state of their constituent actions and artifacts.

5.2.1.3.7 Hyperdocument Editor and Browser

The hyperdocument editor and browser provide functions for users to edit information items and hyperlinks in the currently edited document (see subsection 4.3.3.2.5 and 4.4.4.2.1). User can also navigate around the hyperdocument by following the hyperlinks or by selecting an item in the navigation history queue. When a PBL-protocol is invoked, the currently edited document will be treated as a PBL-net. And then, the information about the PBL-protocol instance will be used to restrict users' behaviors to fit within the corresponding protocol roles (see subsection 4.5.3).

5.2.1.3.8 Virtual Institute Editor and Browser

The virtual institute editor and browser software enables users to navigate from one place to another in a virtual institute and to construct and customize learning contexts (see subsection 4.3.3.2).

In order to focus on the important components of the virtual learning environment, some tools (e.g., chatboard, audio tool, bookshelf, calendar, message box, and so on) are not drawn in Figure 5.2.

5.2.2 A Mapping from the Components to the System Architecture

Now we discuss how the components of the abstract implementation model are mapped on the system architecture.

Eight software modules (the PBL-net schema editor, the PBL-protocol schema editor, the group definition tool, the PBL-plan definition, monitoring, and execution tool, the PBL-protocol instance management and control software, the hyperdocument editor and browser, and the virtual institute editor and browser) are implemented in the 'user interface and functions' component of each client. The system definition and control data (the PBL-net schema base, the PBL-protocol schema base, the community definition, the PBL-plan base, the PBL-protocol instance, the virtual institute definition) and the users' data (the hyperdocument base including PBL-net base) are partially replicated in each client as 'local shared/replicated data', because only those shared data objects that interest a client are maintained by the client. However, the server maintains a complete shared data objects in the 'shared data repository' component.

5.3 Implementation of the System Architecture

This section describes how the system architecture is implemented. The prototype systems are developed based on COAST (stands for <u>co</u>operative <u>a</u>pplication <u>s</u>ystems <u>t</u>echnology). COAST is an object-oriented toolkit for the development of synchronous groupware. It supports the development of groupware by providing both generic components (e.g., session manager, replication manager, and transaction manager) and abstract classes that can be refined to implement a specific application [Schuckmann96]. COAST employs a distributed and replicated architecture. As shown in Figure 5.3, a complete architecture of COAST application consists of three kinds of components: client, mediator, and server [COAST manual]. The COAST clients provide a user interface to enable the end-user to actively manipulate application data. The COAST mediators deal with the synchronous sharing of application data. They maintain the primary copy of the data that is shared among their clients. The COAST servers retrieve data from a persistent storage and send it over the network to a mediator upon request. When this data is to be shared among several clients, the COAST server identifies/creates a COAST mediator that will then deal with sharing aspects. The COAST server and the COAST mediator are generic components that are independent from specific application data. In the figure 5.3, big boxes denote potentially different sites/processes in a networked computer environment. Arrows denote communication channels between processes. A small box represents a bundle of application data, where boxes with the same shading are different replicas of the same bundle of application data.

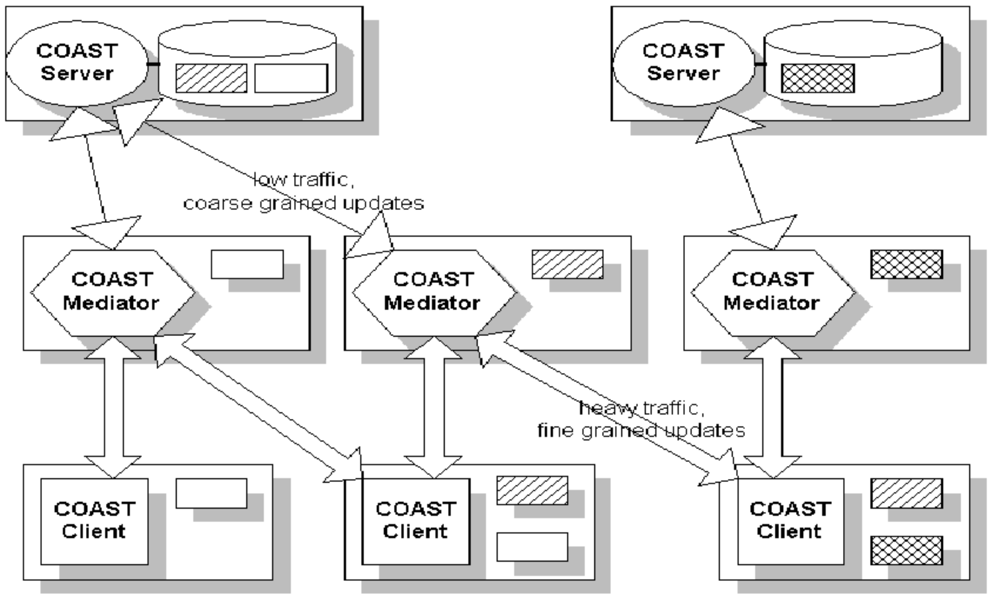


Figure 5.3: COAST Application Architecture (taken from [COAST manual])

Figure 5.4 illustrates the version supported by the first release of COAST [COAST manual], which is suitable for a medium-sized group of users. Our prototype systems are implemented by using this version, in which a mediator keeps shared application data consistent between clients. In fact, different COAST application architectures are transparent for the COAST application developers.

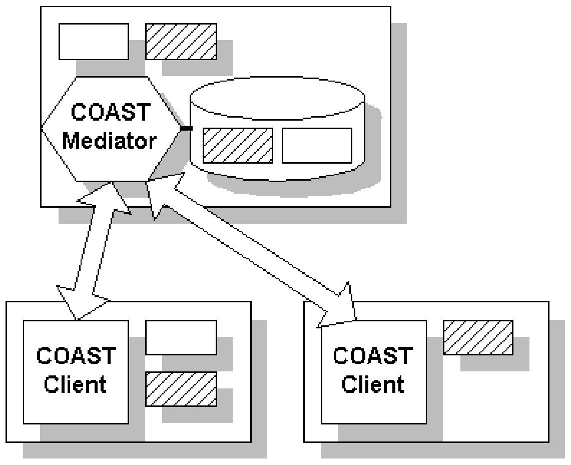


Figure 5.4: Application Architecture of
Current Version of COAST (taken from [COAST manual])

So far, two prototype systems have been developed. Firstly, the concept of 'collaboration protocol' and 'general plan' and their execution within session-based collaborative processes were tested successfully in SCOPE (for "session-based collaborative process-centered environment") [Miao98a] [Miao99a]. Based on these results, a collaborative virtual PBL environment, called CROCODILE (for "creative open cooperative distributed learning environment") [Pfister98] [Miao00e] has been implemented, which realizes the specification of the concepts of this thesis. Like the current version of COAST, these two systems are written in VisualWorks Smalltalk, Version 3.0, and can run on Window'95, Window'98, Window NT and Solaris.

Figure 5.5 illustrates the CROCODILE architecture, which is also a fully distributed and replicated architecture. Each user interacts with an individual instance of CROCODILE. As shown in Figure 5.5, a CROCODILE client has two layers. The top layer is the CROCODILE UI layer, through which users communicate (or interact) with CROCODILE. The next layer contains the data types and operations specified in the last chapter. The next layers belong to a COAST client. The first layer is the shared application framework layer, which provides mechanisms to define the shared data model. The second layer supports transaction management, shared data management, and communication with a COAST mediator. The CROCODILE server ins implemented by using the COAST mediator. The COAST mediator provides generic transaction, replication and storage management.

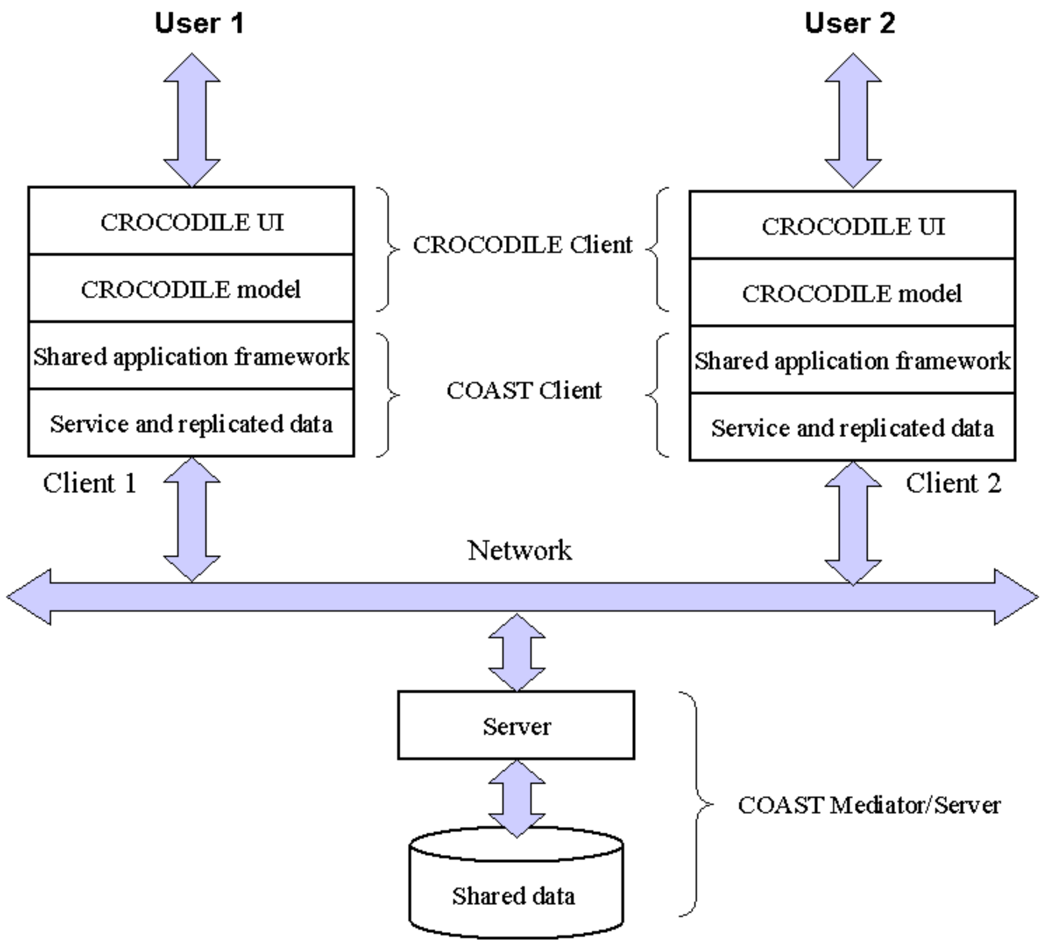


Figure 5.5: CROCODILE Architecture

By means of the COAST facilities, users of CROCODILE can cooperate with each other no matter whether they are geographically co-located or distributed. When a user performs an operation on shared objects (e.g., moving to another place through a door, modifying the statement of a typed node on a PBL-net, and so on) in the virtual institute, the system treats this operation as a transaction. The transaction is not only processed locally, but also propagated to other clients via the mediators to keep data consistent. Finally, all clients will update their user interfaces according to the up-to-date information (for details, see [Schuckmann96]).

5.4 A Cooperative Hypermedia Approach

This subsection describes our technological approach to implement the virtual problem based learning environment. It uses cooperative hypermedia technology to implement the virtual institute metaphor, PBL-net, PBL-protocol, and the PBL-plan as hyperdocuments in a unified style.

5.4.1 Implementation of the Virtual Institute Metaphor

Humans interact with real learning environments and with computer-based learning environments through their perceptions. A large part of the success of a system comes from the effectiveness of the user's experience interacting with the system. In order to enable users of the system to intuitively use their skills of social interactions in the virtual learning environment, firstly the elements of the interface are designed to correspond to the real world counterparts as perceptual metaphors that, in turn, are presented as graphical icons. Secondly, system functions are organized based on these metaphors. Another important design issue is to choose 2-D or 3-D user interface. 3-D user interfaces may be better for understanding the learning context. However, 2-D user interfaces need lower cost of computation, don't need complex input devices, and support easy navigation and manipulation of the learning environment. In particular, in this thesis, the emphasis is on the concepts of learning context, PBL-net, PBL-protocol, and PBL-plan. For the sake of simplicity, a 2-D user interfaces is adopted in this implementation. A place is visualized as a 2-D area wherein all objects in the place are visualized as icons. A visualization of a place is called a place page. Each concrete door will be visualized in the two places to be connected. Each virtual door is visualized only in the source place. The visualization of a door depends on (1) what types of places the door connects and (2) the status of the door. For example, if a concrete door connects a campus and an instructional building, this concrete door is visualized as an instructional building icon in the campus page, and as an exit door icon in the instructional building page. When the status (i.e. open or closed) of the door changes, the icon of the door will change correspondingly.

Actors, documents, and tools are represented in this hyperdocument as leaf nodes. A leaf node contains data whose internal structure is application dependent and is not part of the model. Navigating to such a node invokes an application according to the type of the nodes.

An actor is visualized in a place page as a picture of the actor (its human user). Navigating to an actor node leads to opening a menu. If the 'info' item is selected, a window will open, in which information about the actor (such as name, learning interests, expertise, email address, telephone number, etc) is displayed. A user can edit his/her personal information in the window, but can not edit others' information by using this window. If a user do not let others see her/him, s/he can hide him-/her-self by select the 'hide' item from the menu.

A document contained in a place is visualized in a place page as a document icon with the title of the document. Navigating to a document node leads to opening a document editor (see explanations below).

A tool is visualized as a tool icon. There are various types of tools such as document editor, bookshelf, message box, chatboard, phone, speaker, conversation tool, suitcase, calendar, and specific tool. Each type of tool is visualized as a distinct icon. The icon of a tool may change when the tool is used. Navigating to a tool node leads to invoking a corresponding application tool. The paragraphs below briefly describe some application tools used in the prototype system:

1) A document editor is an application tool that can be used to browse and edit the PBL hyperdocument Three types of tool icons can invoke document editors: whiteboard, computer, and private editor. Depending on the type of tool node the document editor might offer different capabilities. A whiteboard can be used by a group of users, who are located in the same place, while multiple users who may be located in different places can use computers. Only one user at a point in time can use a private editor. When one or more users use a tool, their pictures will be displayed on the tool icon.
2) A bookshelf is a kind of application tool that is used to store documents. Documents contained in a bookshelf are not visible in the place page. The act of navigation to a bookshelf will open an application window, in which all titles of the documents in the bookshelf are listed in a certain order (such as alphabetical, creation time, topic, and so on). Users may open a document from this list. Users can also put a document back to a bookshelf by dragging the document icon and dropping it on the bookshelf icon.
3) A message box is a kind of application tool that is used to transfer documents between places. Like the bookshelf, the act of navigation to a message box will open an application window, which give a list of titles of the documents in the message box. Users can take a document from this list and open it in the place. If a user want to send a document to someone or some place, s/he can drag the document icon and drop it on the message box icon. As a consequence, a window will popup, which displays a list of names of users and public places. After the user selects one or more items in the list, the system will distribute the document to the message boxes of the selected persons' homes and message boxes of the selected public places.
4) A calendar is a kind of application tool that is used to manage actions. Clicking on a calendar icon will open the corresponding application. Users can use it to schedule actions and monitor the states of actions.
5) A chatboard and a conversation tool are text-based communication tools. A chatboard is used by multiple users located in the same place, while a conversation tool supports a private conversation between two users. When two users are talking by using this tool, a link with a label (which shows the topic of

the conversation), which connects the pictures of two users, can be viewed in that place.

6) An audio tool supports oral communication. The act of navigation to two types of tool nodes can invoke this application tool: speaker and phone tool nodes. A speaker tool is used by multiple users located in the same place, while a phone tool is used by two users who are located in different places. When clicking a phone icon, a user can see that a window pops up, which shows the user's personal phone number list. If the user selects a number, the phone with this number will ring in the place where the phone is located. If a user in this place clicks on the phone icon, an audio communication channel will be established for them.
7) The group definition tool is used to define the structure of the community of the institute. A picture of a user with his/her name represents the user, and a group icon with a name represents a group. Clicking an icon results in opening a window, which can be used to edit information of an actor or a group depending on the type of the icon. An arrow between two icons represents a relationship.

So far, the primary elements of the institute hyperdocument model and their relations have been described. By different combinations of these elements, various learning contexts can be constructed. The paragraph below now describes the primary operations on the institute hyperdocument.

Places, tools and documents can be added/removed/moved in an institute hyperdocument and people can navigate within an institute hyperdocument. However, some operations depend on the place type. For example, it is allowed to create instructional buildings in a campus and to create public rooms in instructional buildings. If a new place is created in the current place, the new place becomes a place contained in the current place (i.e. a nesting relationship is created). Meanwhile, a concrete door between these two places will be created together. Users can also connect two existing places by creating a virtual door. Performing "open" and "close" operations can change the status of a door. Only the owner of the place has the right to open a door.

This hyperdocument model has a distinct feature. According to the categories of Conklin [7], a concrete door in this model can be regarded as an organizational link and a virtual door can be regarded as a referential link. However, the link in our model has an additional attribute - status. If an actor don't want to be disturbed by others when s/he is doing something in a place, s/he can simply close the door. When a door is closed, it is impossible to navigate through the door. A link with a status attribute provides a flexible navigation control mechanism.

Figure 5.6 shows an example of a virtual institute, which is represented as a hypermedia document. This hyperdocument consists of a set of nodes connected by a set of hyperlinks. The node types and link types are explained in the figure. In this virtual institute, three users are registered currently and two of them are working in a public room (Alice and David) and are editing a document by using a computer tool. The third user (John) is working at home and is editing the same document through a computer tool that is connected to the computer tool in the public room. That is, users can collaborate both within the same place and across places.

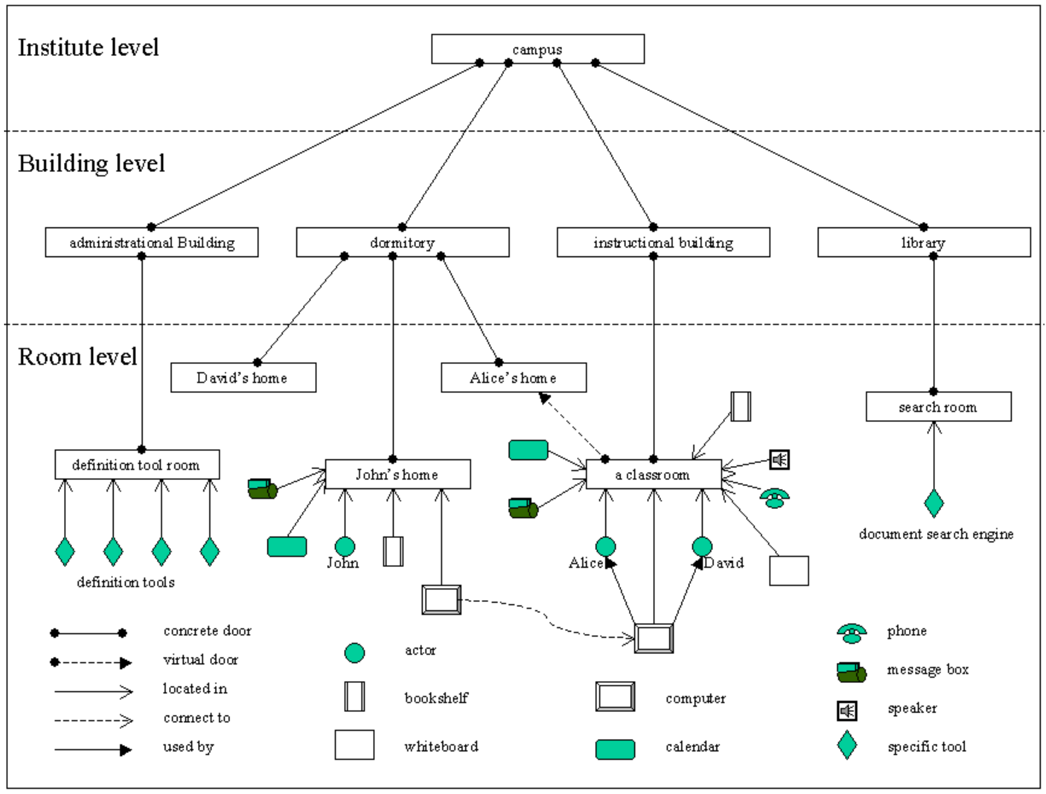


Figure 5.6: A Hyperdocument Representing a Virtual Institute

5.4.2 Implementation of the PBL-net

As discussed in the last chapter, a document editor tool can be used to construct shared knowledge. When users of a document editor use the PBL-net schema to facilitate the construction of shared knowledge, this tool can be regarded as a collaborative knowledge representation tool for PBL. This tool is implemented by using cooperative hypermedia technology. By means of this tool, participants of a PBL activity are able to collaboratively construct a particular knowledge representation as a hyperdocument, called as a *PBL hyperdocument*. The start node of a PBL hyperdocument is a special document node whose content contains a set of typed nodes and typed links between the typed nodes. This special document node is called a PBL-net node and its content is called a PBL-net. Each typed node and typed link contained in a PBL-net has a "type" attribute whose value will be a node type or a link type defined in the PBL-net schema. Each typed node in a PBL-net has its content page and "statements" attribute whose value is the title of the content page of the typed node. The content page of a typed node is a representation of the typed node that reflects the values of the typed node. The content page of a typed node contains detailed information about the typed node in the form of text, table, image, scribble, and even untyped node. An untyped node has the same set of attributes as a typed node except for the "type" attribute. The content page of an untyped node, in turn, contains detailed information and even other untyped nodes. Therefore, a PBL hyperdocument has two levels: PBL-net level and information level. There are some constraints in the PBL hyperdocument. For example, there is only one PBL-net node in a PBL hyperdocument. The structure of a PBL-net has to comply with the definition of the PBL-net schema. The PBL-net node can not be contained by any untyped node. Untyped nodes can not appear in the PBL-net.

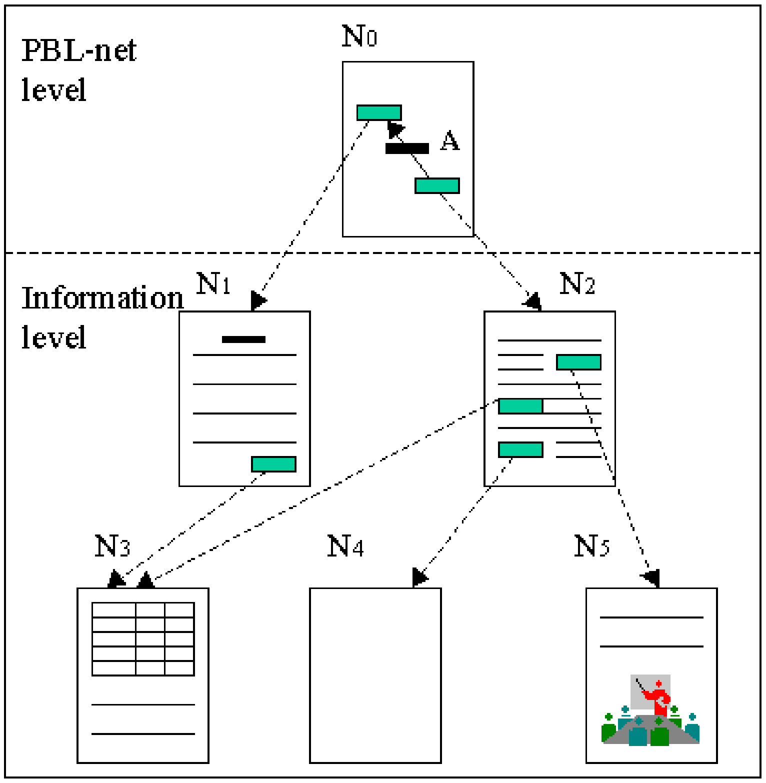


Figure 5.7: An Hyperdocument Representing a PBL-net

Figure 5.7 gives an example of a PBL hyperdocument. N0 is a PBL-net node and its content is a PBL-net. The PBL-net contains two typed nodes N1 (“problem” type), N2 (“solution” type) and a typed link A (“solve” type). N1 contains an untyped node N3. N2 contains untyped nodes N4, N5, and N3 as well.

A document editor can be used to browse and edit a PBL hyperdocument jointly. First of all, to start a PBL activity, users can create a PBL-net node by using the document editor. Then they can construct their own PBL-net by creating and manipulating typed nodes or typed links on the PBL-net. To create a typed node in a PBL-net, a user should assign values to the attributes of the node: node type and node statement. The node statement serves to briefly describe the content of the node and to publish a point of view to others. While a typed node is created, its content page will be created automatically. Typed links can be created to connect two existing typed nodes while complying with the definition of the PBL-net schema. Node type specific operations can be performed on the corresponding typed nodes. For example, on the nodes with “issue” type, learners can assign values such as “I know” and “I need to know” to the corresponding attributes.

Users can navigate to the content page of a typed node. When any user of the same shared document editor navigates to the content page of the typed node, all users of the document editor will go together. That is, all users of the same document editor always work on the same document page. By using the same document editor, users can edit information units in the form of text, table, image, scribble, and untyped node. Users can create a content page for an untyped node or connect it to an existing document page. By manipulating a shared PBL hyperdocument, the users can collaboratively construct a shared knowledge representation and interact with each other through the shared knowledge representation.

5.4.3 Implementation of the PBL-protocol

As mentioned in the last chapter, a PBL-protocol is defined as an extended, hierarchical state-transition diagram, which is represented as a hypertext document. The PBL-protocol schema editor is used to define PBL-protocols. A PBL-protocol consists of a set of nodes connected by a set of links. A node represents a protocol state. A node in a diagram may contain a sub diagram describing the state represented by that node in more detail, called a sub-PBL-protocol. That is, a node serves as a hyperlink connecting to a sub-PBL-protocol. A link in the diagram represents a protocol transition. Nodes are identified by a unique name within a PBL-protocol. A user of this tool can define a new protocol state by creating a node. Then, the user can click the created node and a dialog window will pop up, which allows the users to select one or more items from a list of behavior rules. In a similar way, a link can be defined. If the user make an 'open' operation on a node, the user will navigate to the sub-PBL-protocol that is nested in the node. S/he can edit this sub-PBL-protocol by manipulating the diagram of the sub-PBL-protocol.

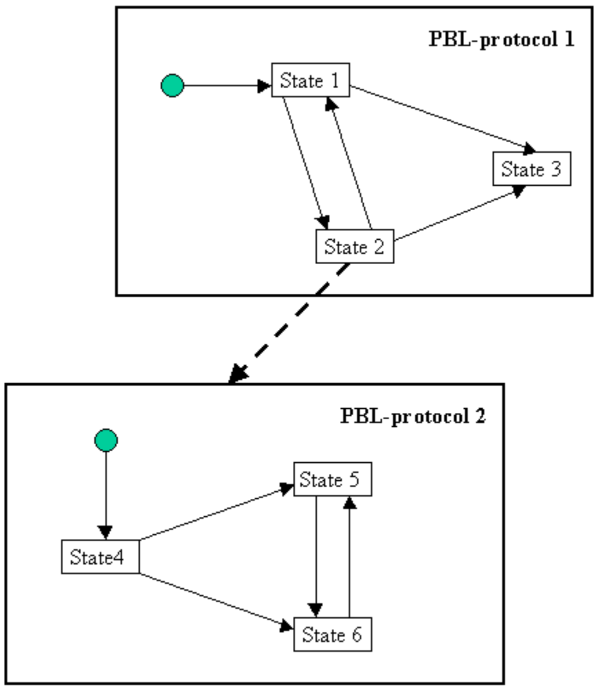


Figure 5.8: A Hyperdocument Representing two PBL-protocols

For executing a PBL-protocol instance, each user has to be assigned to a certain protocol role. When a user performs an operation on an object, the system will check whether this member is permitted to execute the operation according to the behavior rules defined for the currently executed protocol state. If this check fails, nothing happens except for displaying a warning message. If the check is successful, an update event will be propagated to all other clients to keep data consistence. If a user performs an operation following a behavior rule bound to a protocol transition link (i.e., changing a protocol state or proposing a solution), a state transition will be caused according to the definition of the used PBL-protocol.

Figure 5.8 illustrates two PBL-protocols. In this figure, each rectangle represents a protocol state and each arrow represents a protocol transition. Those protocol states which are linked to circles are start states (e.g., state 1 and state 4) of PBL-protocols. PBL-protocol 2 is a sub-PBL-protocol of PBL-protocol 1 in the state 2. That is, when an instance of PBL-protocol 1 is executed in state 2, an instance of PBL-protocol 2 will be initiated.

5.4.4 Implementation of the PBL-plan

The PBL-plan definition, monitoring, and execution tool is implemented by using collaborative hypertext technologies. This tool provides a visual process model language for the definition of PBL-plans. A defined PBL-plan contains information about all scheduled actions that constitute the plan, the values of attributes of each action, and the relationships among these actions. Like a PBL-protocol, a PBL-plan is described as a hypermedia document consisting of layered nodes, which are connected via links. A PBL-plan can potentially be decomposed into sub-PBL-plans. A sub-PBL-plan is represented visually by a node with the label “Process Node” and the name of the PBL-plan. The components and structure of a sub-PBL-plan are described on the content page of the node. An action is represented visually by a node, too, but it carries the label “Session Node” and the action name. A relationship among actions and sub-plans is represented as a link in the hypertext document. The tool provides six types of connection nodes: start point, end point, or-split, and-split, or-join, and and-join. They are used to specify temporal relationships between actions and sub-PBL-plans. Artifacts are also represented visually by nodes with the label “Artifact”, “Artifact Input”, and “Artifact Output”. “Artifact Input” and “Artifact Output” are used to transfer artifact across sub-PBL-plans, while “Artifact” nodes are used to transfer artifacts within a sub-PBL-plan. A temporal relationship is represented as a link as well.

Different colors are used to represent different states of actions and plans. When a PBL-plan is executed, users can monitor the changes of the states through viewing the changes of the colors of the nodes from the hyperdocument representing the PBL-plan. Operations for execution of a PBL-plan can be implemented by manipulating the hyperdocument representing the PBL-plan. For example, terminating an action can be done by clicking the session node representing the action and selecting the “terminate” item from the popup menu.

Figure 5.9 illustrates two PBL-plans. In this figure, each rectangle represents a session node or a process node. Each circle represents a connection node. Each arrow represents a temporal relation or an artifact relation. PBL-plan 2 is a sub-PBL-plan of PBL-plan 1, which is nested in the process node 2.

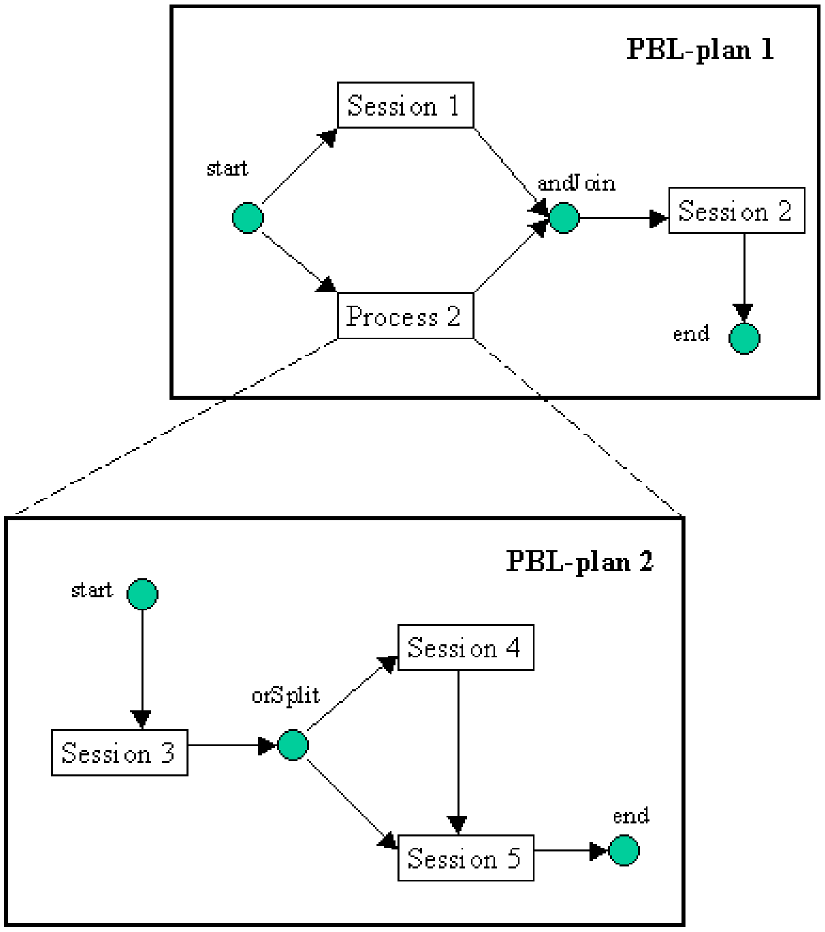


Figure 5.9: A Hyperdocument Representing two PBL-plans

6 A Usage Scenario and Experiences

This chapter describes a usage scenario of the prototype system and reports preliminary experiences with this approach.

6.1 A Usage Scenario

This section describes what the virtual learning environment looks like from the user's perspective and how the users interact with the virtual PBL environment in a usage scenario. In this scenario, a user navigates in the virtual institute and then participates in a collaborative PBL activity.

6.1.1 Virtual Institute Editor and Browser

When a user starts running the system, the start window of the system will open. The user can select a virtual institute and input the user's password. If the password of the user is correct, the virtual institute editor and browser window will open.

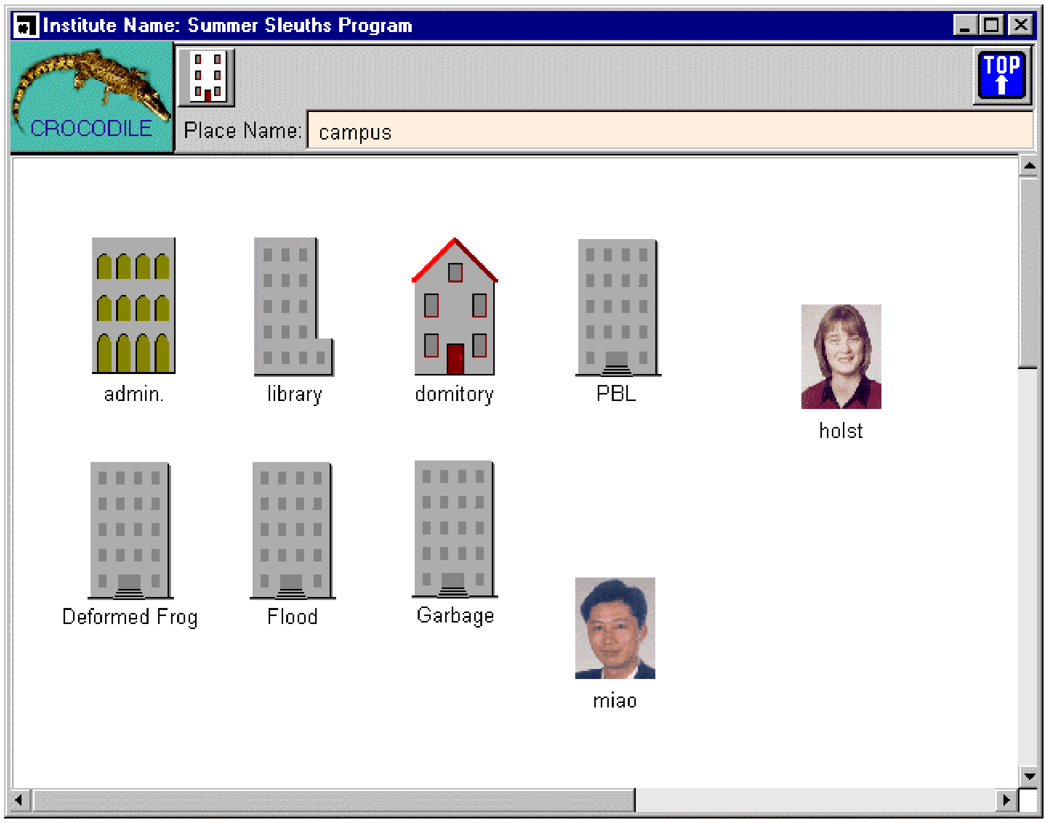


Figure 6.1: The Campus of the Virtual Institute

Figure 6.1 illustrates the virtual institute campus in the virtual institute editor and browser. The browser shows a window title bar showing the institute's name. The upper part of the window contains the system logo, a building button, a "TOP" button (for going to the campus as a shortcut), and a text field (for showing the name of the current place). Below, the window displaying the content of the place where the user is currently located is presented.

When the user enters a virtual institute, he is located in the campus of the institute. In the window, the user can see his picture and the pictures of other users who are currently located in the campus of the virtual institute. As the user's cursor moves in the window, his picture will follow the movement of the user's cursor (it functions like a tele-pointer). He can observe the movement of other users' pictures while their cursors move. As shown in Figure 6.1, in the campus, there is a set of different iconic presentations of buildings' metaphors, including an administrational building, a library, a dormitory, and several instructional building icons. The administrational building, the library, and the dormitory are constructed by the system when the institute is created. The user can create an instructional building by clicking the building button, typing a name of the instructional building, and anchoring it in the window. As mentioned in chapter 4 and chapter 5, the virtual institute is modeled and implemented as a hypermedia document that consists of a set of virtual places and connections between places. A building icon or a door icon in each place presents a hyperlink to the connected place. The tool supports navigation within the virtual institute by changing the currently displayed virtual place when the corresponding door icon or building icon is clicked. Therefore, the user can enter a building by clicking on the icon of the selected building.

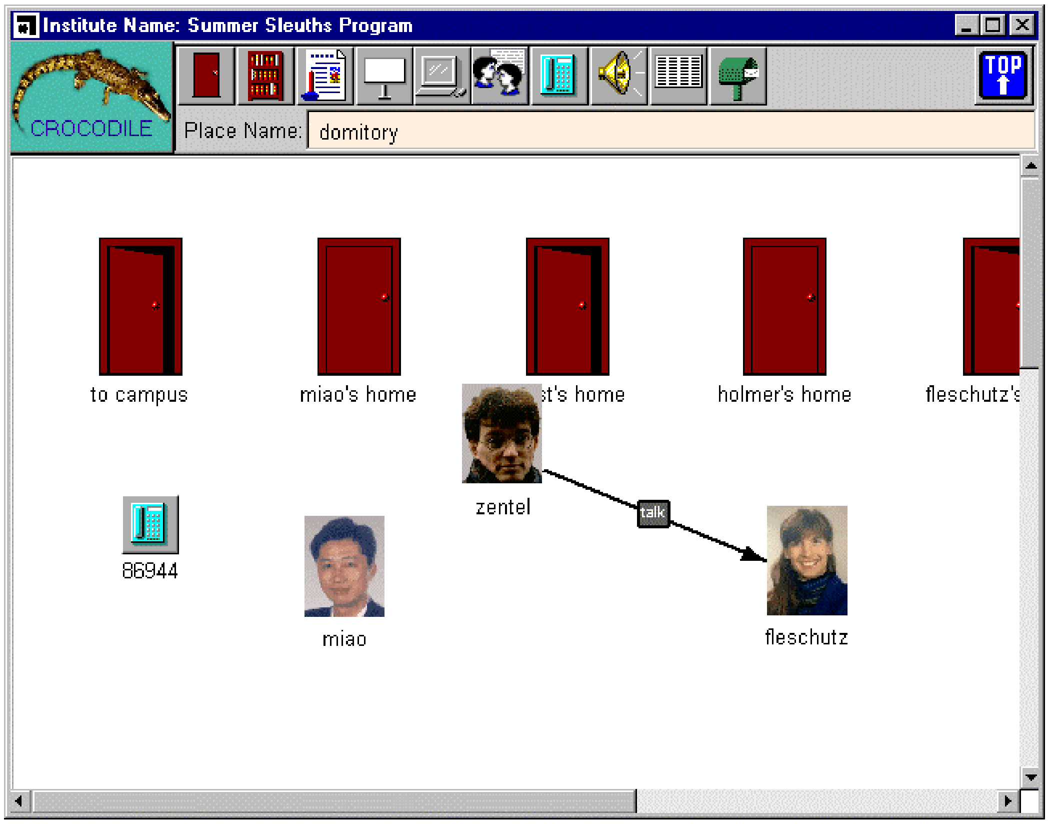


Figure 6.2: The Corridor of the Dormitory

When the user enters a building (e.g., the dormitory), the window content of the virtual institute editor and browser window will change to show the inside view of the building (see Figure 6.2). In addition, the place edit button bar will change. From left to right, the buttons in the button bar are door, bookshelf, private hyperdocument editor, whiteboard, computer, chatboard, phone, speaker, calendar, and message-box. These buttons enable users to customize a learning context by editing a place. The inside view of the building looks like a corridor with a set of door metaphors. As illustrated in Figure 6.2, a set of door icons represent an exit to the campus and entrances to all homes of users. In addition, tools may be available in the corridor (e.g., the telephone). The pictures of all users who are currently located in this place can be seen in the window as well. If the user wants to know information about a user in this place, he can click the picture of the user. A window will open in which the information about the user (e.g., email address, telephone number, expertise, learning interest, and so on) is displayed. They can talk with each other by using a conversation tool. As illustrated in Figure 6.2, when a pair of users is talking with each other, the user can observe an arrow that connects the two pictures of those users. He can join the conversation by clicking on the arrow. The user can enter his home by clicking on the icon of the home door. The user can open/close the door of his home, but can not open/close any home door of others. If the view of a room door is represented by a closed-door icon, he can not enter the place. This provides a flexible mechanism for access control. If a door is opened, the user can enter a place or go to the campus by clicking the corresponding door icon.

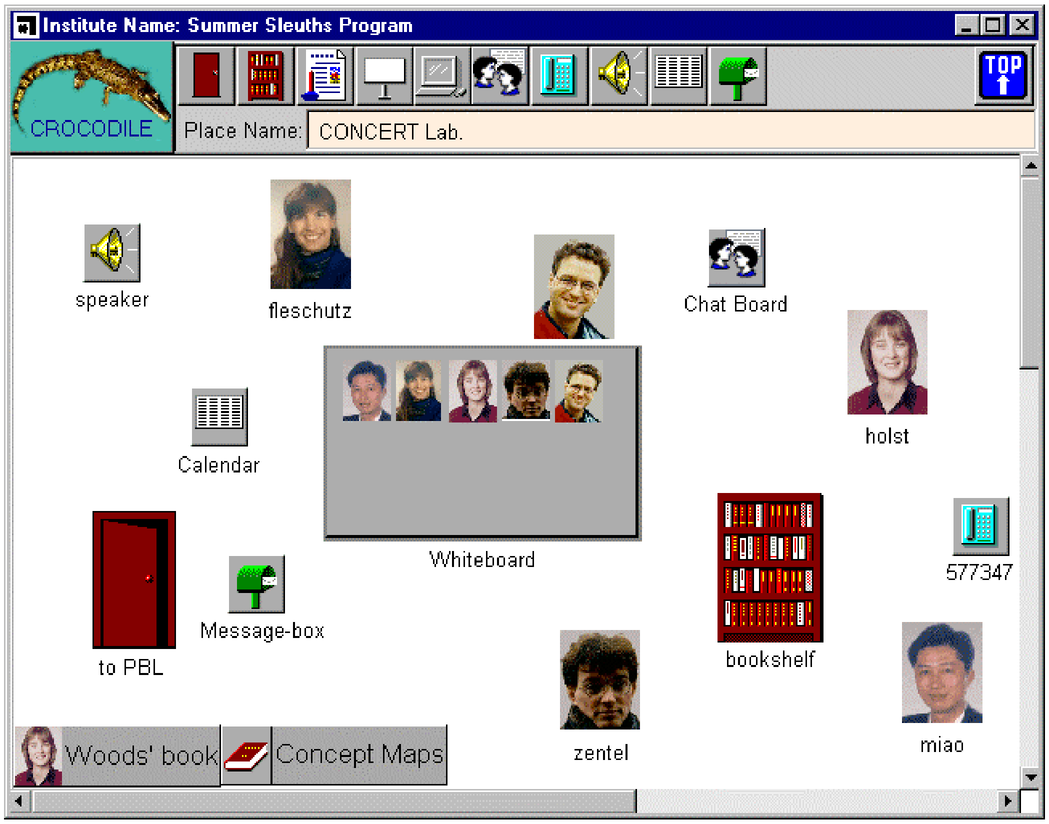
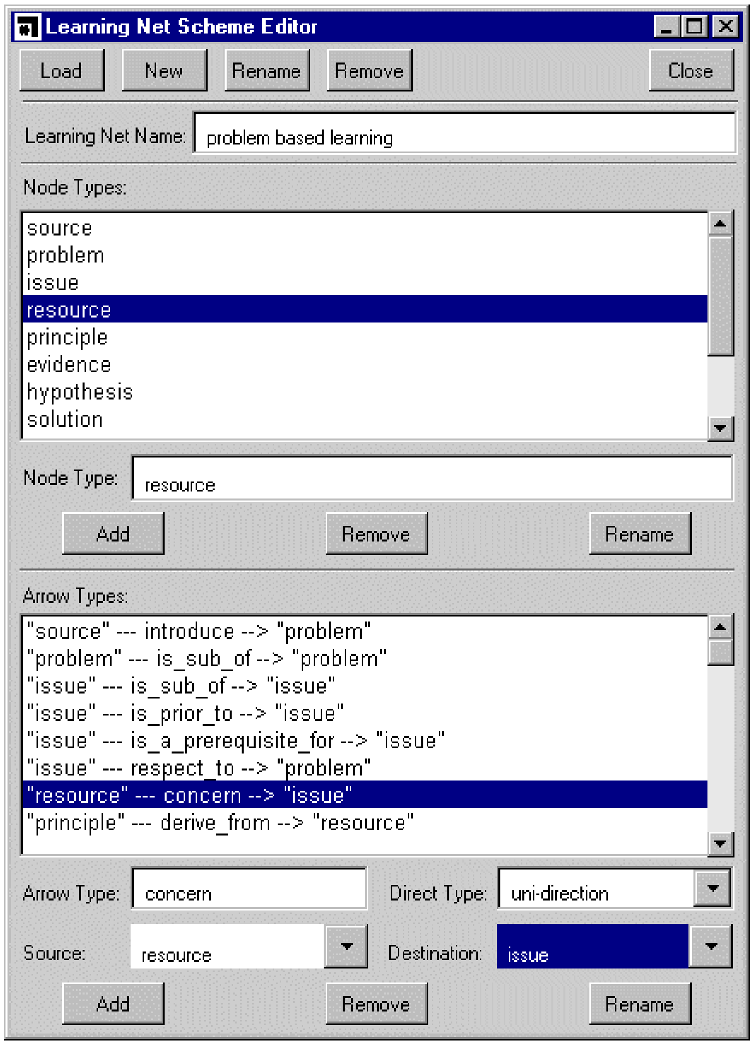


Figure 6.3: A Public Room

In this way, the user can navigate around the virtual institute. If an action is scheduled to be performed in a public room (the CONCERT Lab in this example), the user can move to this room on time. In this room, necessary tools and documents are available. The user can use them while carrying out the action. As illustrated in Figure 6.3, in this room there is a chatboard, a speaker, a calendar, a message-box, a bookshelf, a phone, and a whiteboard. When the user selects a tool by clicking on the icon of the tool, the corresponding tool window will open. For example, a chatboard is a text-based communication tool used by the users in this room. The speaker is an audio tool that enables the users in this room to speak and listen to each other. The calendar is used to arrange and monitor actions that are performed in this room. The phone can be used to talk with someone who is currently working in another room. The user can look up documents stored in the bookshelf or in the message-box by clicking the corresponding icon. The documents inside it will be listed in a pop-up window. The user can open a document by selecting it. In Figure 6.3, two documents (titled as "Woods' book" and "Concept Map") are put on the floor. As indicated by a picture of a user on one document icon, that user is reading this document. The other document is closed so that any user can open it. When a document is in use, other users can not open it. The user of a document can put it back to the bookshelf by dragging it and dropping it on the bookshelf icon. If the user drags the document icon and drops it to the message-box, the system will ask the user where to send this document. If the user wants to share this document with others, he can drag and drop it on a whiteboard icon. A shared hyperdocument editor that represents the whiteboard will treat this document as the currently edited document. When the user clicks on the icon of the whiteboard, the window of the shared hyperdocument editor will pop up on the screen. The pictures of users on the whiteboard indicate who is currently working on the whiteboard. The hyperdocument editor is an important cooperative learning tool that will be discussed in detail at the end of this section.

6.1.2 PBL-net Schema Editor

The user can go to the administration building to use the PBL-net schema editor. As illustrated in Figure 6.4, the user can create and delete node types and link types of the PBL-net schema by using this tool. The user can also load, create, rename, and delete schema by means of this tool if necessary.



Learning Net Name: problem based learning

Node Types:

source
problem
issue
resource
principle
evidence
hypothesis
solution

Node Type: resource

Arrow Types:

"source" --- introduce --> "problem"
"problem" --- is_sub_of --> "problem"
"issue" --- is_sub_of --> "issue"
"issue" --- is_prior_to --> "issue"
"issue" --- is_a_prerequisite_for --> "issue"
"issue" --- respect_to --> "problem"
"resource" --- concern --> "issue"
"principle" --- derive_from --> "resource"

Arrow Type: concern
Direct Type: uni-direction
Source: resource
Destination: issue

Figure 6.4: PBL-net Schema Editor

6.1.3 PBL-protocol Schema Editor

The PBL-protocol schema editor is located in the administration building. The user can use it to create and define a PBL-protocol schema by creating and deleting protocol states and state transitions in a diagram (see Figure 6.5). First, a learning net schema should be selected from the learning net schema base or a new one should be created. Secondly, the protocol roles should be defined as a list of names. Thirdly, a learning net schema should be selected from the learning-net schema base. A protocol schema is defined as a diagram on the window content of this editor window. While defining a state, the user clicks the “state” button and types the name of this protocol state in the pop-up dialog window, and then positions it on the window content. A transition is created by a gesture to drawing a line from the source state to the destination state node. When the user clicks a state or transition in the diagram, a window will pops up and the user can define a set associated behavior rules for the state or the transition. The user can define a sub-protocol of the currently edited protocol as well by selecting the name of a predefined sub-protocol and binding it to a state of the currently edited protocol. The user can also load, rename, and delete the protocol schema stored in the PBL-protocol schema base if necessary.

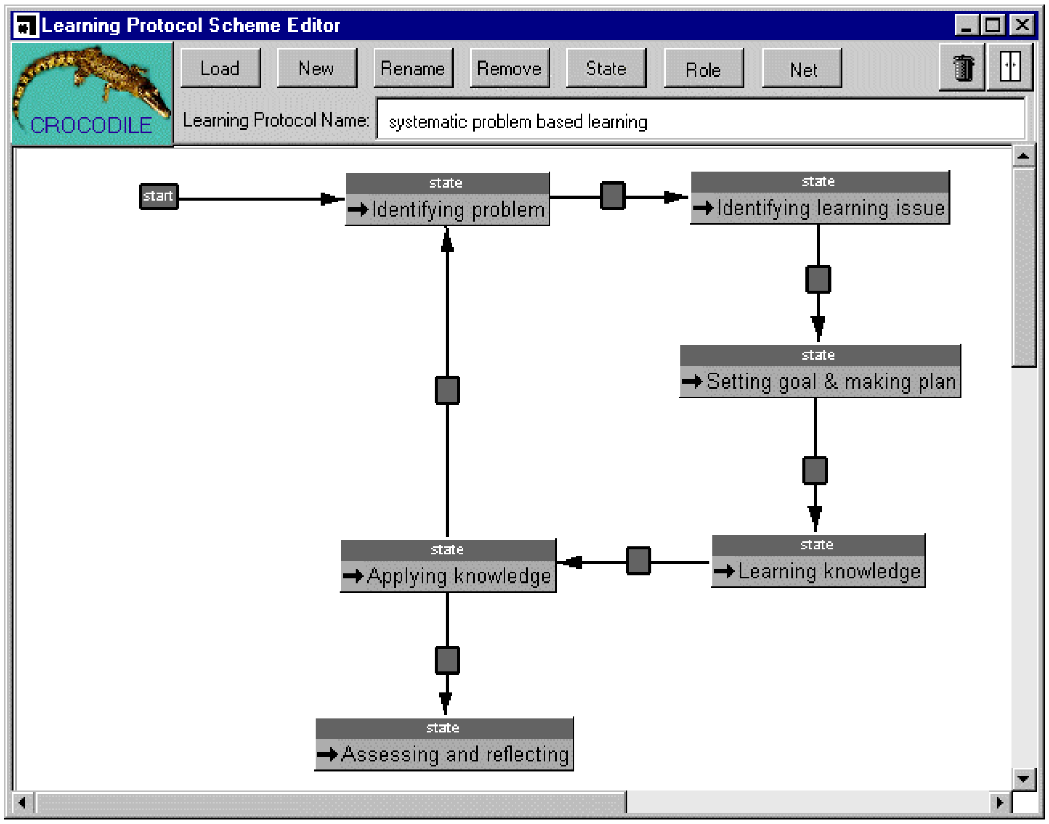


Figure 6.5: Learning Protocol Schema Editor

6.1.4 Group Definition Tool

The group definition tool is also located in the administration building. As illustrated in Figure 6.6, there are two agent icons (actor and group) in the button bar. The user can click on an agent icon, type the name of the agent, and anchor it in the content window. After clicking the picture of an actor in the content window, a window pops up and the profile of the actor can be edited by input information such as email address, telephone number, learning interests, expertise, and so on. In the same way, the properties of a group can be specified. By using group definition tool, the user can also define the organizational structure by creating arrows between agents.

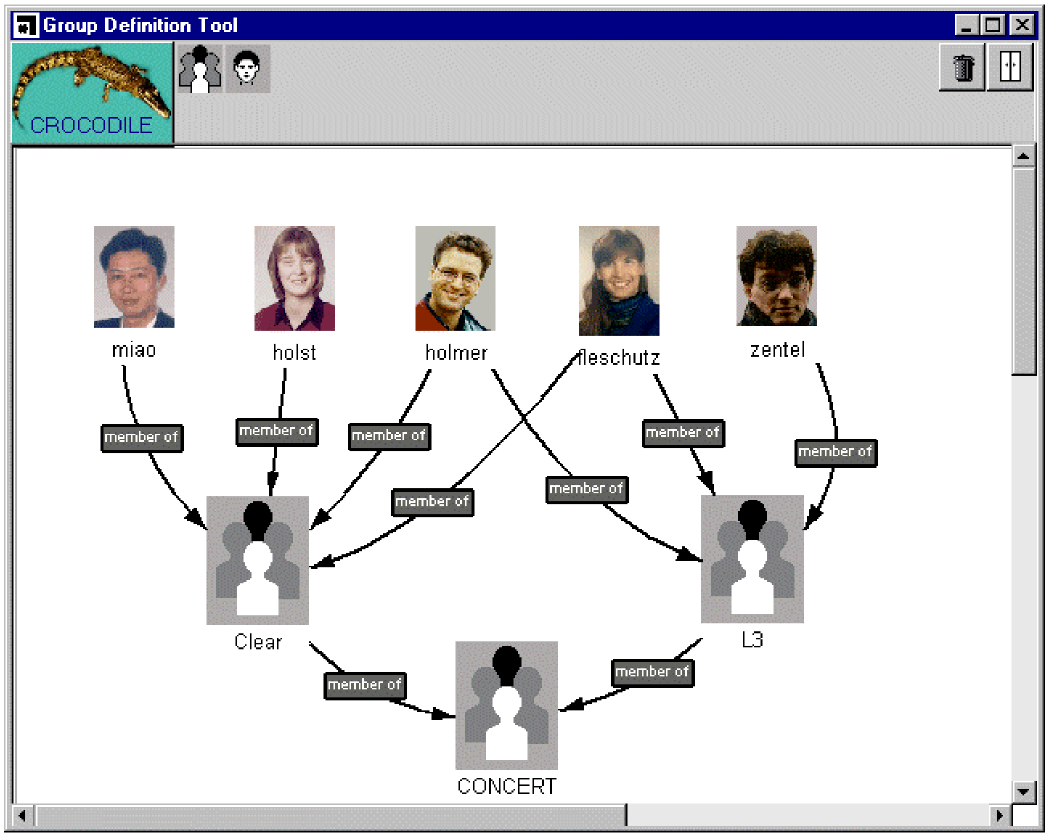
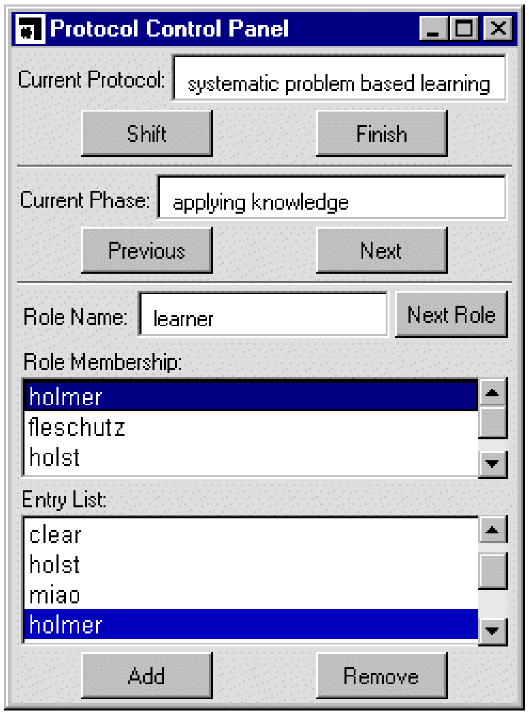


Figure 6.6: Group Definition Tool

6.1.5 Protocol Control Panel

When a group of learners construct a shared PBL-net by using a hyperdocument editor, they can initiate a PBL-protocol to guide them to carry out a PBL activity. After an instance of the selected PBL-protocol is created, a protocol control panel for this protocol will open (see Figure 6.7). The user can select one or more agents to take a protocol role from the entry list. From this panel the information about the currently used PBL-protocol schema and the current state of the protocol instance can be observed. The user can shift from currently used protocol to another protocol from the protocol family to which both protocols belong in the lifecycle of the protocol instance. At any point in time, only behaviors that are associated to the current state are allowed when manipulating the PBL-net by using the hyperdocument editor. The protocol control panel allows users to shift from current state of the protocol instance to another state by clicking the "previous" or "next" buttons. The state of the protocol instance will change according to the definition of the protocol schema, and the tool will provide guidance about how to perform this task and what contributions are expected to each protocol role.


Protocol Control Panel

Current Protocol: systematic problem based learning

Current Phase: applying knowledge

Role Name: learner

Role Membership: holmer, fleschutz, holst

Entry List: clear, holst, miao, holmer

Figure 6.7: Protocol Control Panel

6.1.6 PBL-plan Definition, Monitoring, and Execution Tool

As illustrated in Figure 6.8, by using the process definition tool, the user can create the work process description as a learning plan in a computer tractable form. This tool provides a visual process model language for process definition. A learning plan is described as a hypertext document (a hierarchical diagram) consisting of layered nodes, which are connected via hyperlinks. A process can potentially be decomposed into sub-processes that act as sub-plans. A learning plan or a sub-learning plan is represented visually by a node with the label "Process Node" and the name of the plan. The components and structure of a (sub-) learning plan are described on the content page of the node. An action is represented visually by a node too, but it carries the label "Session Node" and the name of the session. Session nodes are elements of processes and each session has a number of attributes, which have to be specified. A relationship among sessions and sub-processes is represented as a typed link in the diagram. There are two kinds of relationships: temporal relationship and artifact relationship. A temporal relationship represents the time dependence between sessions. If session A precedes a session B, it means that when A is finished, the active-condition of B is evaluated. This tool provides six types of checkpoints. They are start point, end point, or-split, and-split, or-join, and and-join. A temporal relationship is represented as a black arrow in the diagram. An artifact relationship is used to represent a kind of dependence of artifacts between sessions, such as transferring and sharing. An artifact transferring relationship denotes the situation when an output artifact of one session will be transferred into another session as an input artifact. An artifact sharing relationship means that an artifact can be viewed and manipulated by people working in different sessions running concurrently. An artifact relationship is represented in the process description as a blue arrow pointing to/from a named rectangle representing an artifact. This implementation meets the specification of PBL-plan (see section 4.6)

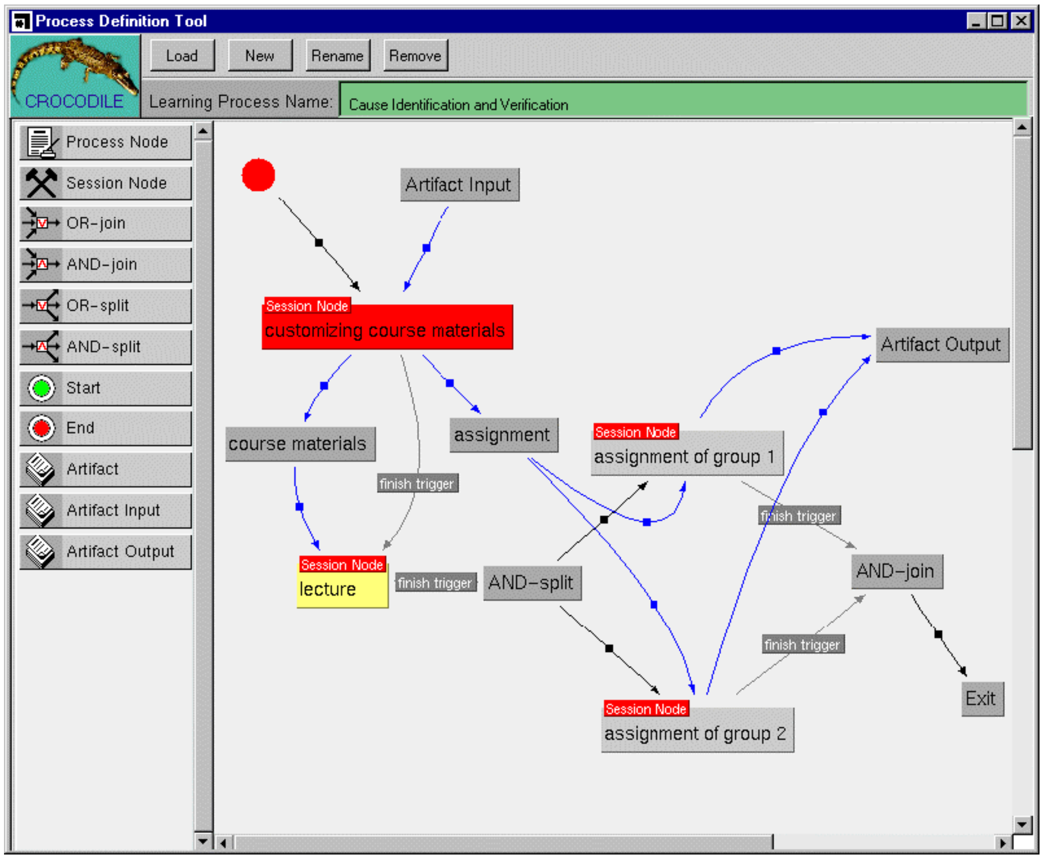


Figure 6.8: PBL-plan Definition, Monitoring, and Execution tool

Process execution is concerned with the enactment of a process following the defined work plan. When users start to execute the work plan, all sessions connected to the start node of the plan will be active. As the process executed, the state of the sessions described in the work plan will change. The different colors indicate the different states of sub-processes or sessions. Even when a work plan has already been executed, parts of the work plan could be modified. That is, the values of attributes of the corresponding sessions could be modified and the sub-plan could be altered.

6.1.7 HyperDocument Editor and Browser

Three tools including the private hyperdocument editor, the whiteboard, and the computer provide same functionality for users to edit hypermedia documents. The difference between these three tools is: the private hyperdocument editor is not a synchronously shared tool; the whiteboard can be used by multiple users who are located in the same place; and the computer tool can be used by multiple users who are located in the same place and in different places as well. Because their appearance is designed for different use situations, users interact differently with these tools.

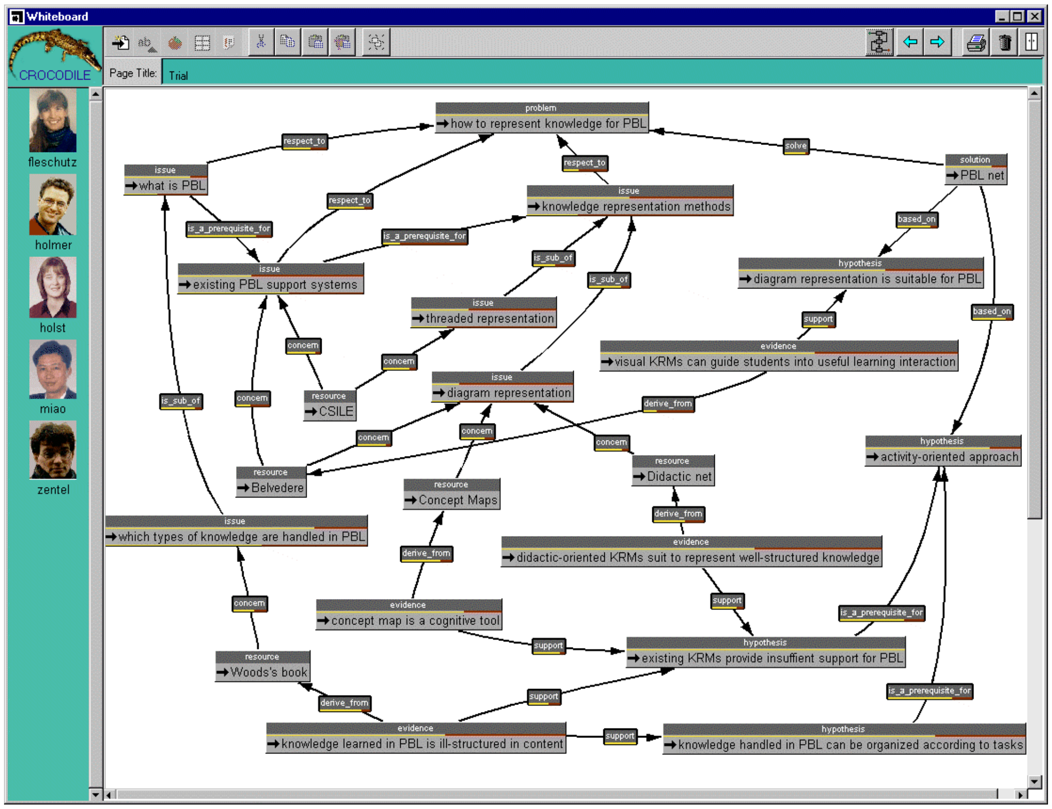
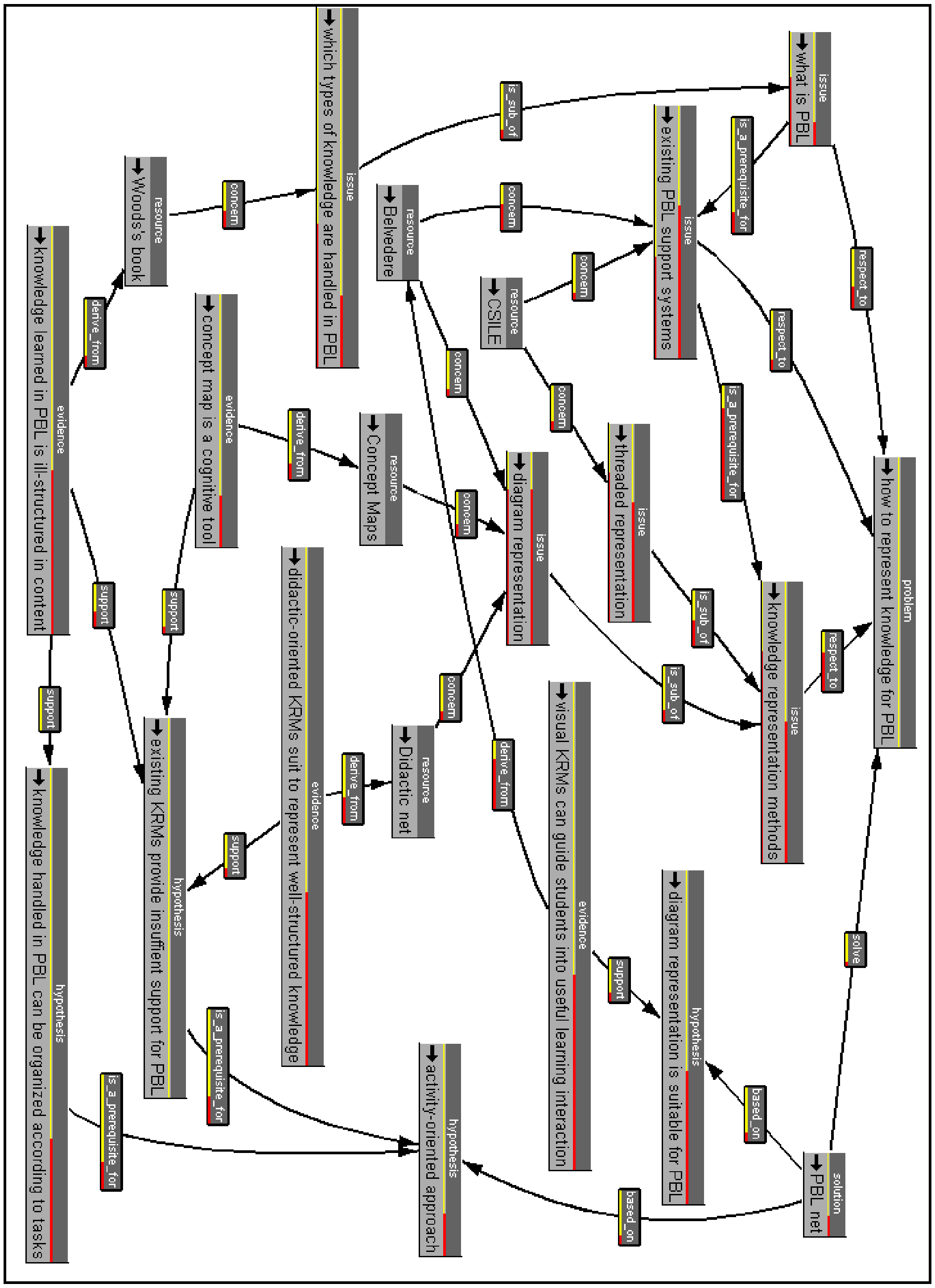


Figure 6.9: HyperDocument Editor and Browser

As shown in Figure 6.9, the hyperdocument editor and browser consists of a system logo, a user bar, an iconic edit button bar, a text field (showing the title of the currently edited document page), and the editing area (showing the content of the currently edited document node). The iconic edit button bar contains (from left to right) “node”, “text”, “image”, “table”, “annotation”, “cut”, “copy”, “paste”, “external text paste”, “merge”, “protocol”, “backward”, “forward”, “print”, “garbage”, and “close” buttons. By using this editor, a PBL hyperdocument can be collaboratively constructed. The window shown in Figure 6.9 corresponds to the opened whiteboard shown in the Figure 6.3. As mentioned before, the hyperdocument editor can be used to construct a PBL-net. When a PBL-protocol is initiated by clicking on the “protocol” button, the currently edited document will be treated as a PBL-net. And then, the information about the PBL-protocol instance will be used to restrict users’ behaviors to fit within the corresponding protocol roles. When the user clicks the whiteboard icon (mentioned above), the hyperdocument editor window that represents the whiteboard pops up. Including the user, five users whose pictures are listed in the user bar (left side of the window) are currently working on the PBL-net. In the upper part of window there is an icon bar that contains node button, text button, image button, table button, annotation button, cut button, copy button, paste button, external text paste button, merge button, protocol button, backward button, forward button, print button, garbage button, and close button. To create typed nodes, the user clicks on the node button (the left most icon in the icon bar), and drags it into the window content to position the node. If more than one node type is pre-defined by the schema, then a selection box appears, from which the user chooses from a series of text options describing the node types available. The user then types in the statement that describes the content of the node. The user creates links by a draw-line gesture going from the source to the destination node. A selection box once more allows them to define its link type, by choosing from a series of text options describing the link types available.



issue
what is PBL
issue
existing PBL support systems
issue
which types of knowledge are handled in PBL
resource
Woods's book
evidence
knowledge learned in PBL is ill-structured in content
evidence
concept map is a cognitive tool
resource
Belvedere
resource
CSILE
resource
Concept Maps
issue
diagram representation
issue
threaded representation
problem
how to represent knowledge for PBL
issue
knowledge representation methods
evidence
didactic-oriented KRMs suit to represent well-structured knowledge
resource
Didactic net
evidence
visual KRMs can guide students into useful learning interaction
hypothesis
knowledge handled in PBL can be organized according to tasks
hypothesis
existing KRMs provide insuffient support for PBL
hypothesis
activity-oriented approach
hypothesis
diagram representation is suitable for PBL
solution
PBL net
is_sub_of
concern
is_a_prerequisite_for
respect_to
derive_from
support
solve
based_on

Figure 6.10: An Example PBL-net

Information associated to negotiation of knowledge can be connected to each node using the pop-up menu of that node. The user expresses his perspective and confidence with the contents of a node by assigning values to the 'perspective' and the 'confidence' attributes from the pop-up window. The profession of learners in their knowledge about the content of a node with the type of "issue" is also expressed via the pop-up menu options, specifically 'I know or do not know' and 'I need to know or not'. The group perspective regarding a node or a link is represented in the net by a colored line, so does the profession. Each node in the net also serves as a hyperlink that connects to other document in which more information about this node is provided. The user can navigate in the hyperdocument by following the hyperlinks or selecting an item in the navigation history queue. Figure 6.10 shows the content of the example PBL-net in a more readable form.

6.2 Experience

The prototype system has been tested and used in our division. Five people used the problem-based learning approach that is supported in CROCODILE to tackle a research topic of interest in our research group. One person took the role of the tutor. This trial lasted two weeks, on average, one hour per working day. Sometimes they worked in a synchronous collaboration mode in our laboratory (because the quality of the audio tool is not good enough), and sometimes they worked in different office-rooms in an asynchronous collaboration mode.

The virtual institute created in this trial had seventeen places. The users were able to use the system functionality intuitively to navigate in the virtual institute, and to create new places and artifacts when they needed them. They found that the information about the local learning context is rich and is easy to be understood. They used their experience in the real world to choose tools available or create a tool, to handle documents, and to interact with each other in the virtual institute. In this trial, a PBL net was created and many documents were created and connected as a hyperdocument. The PBL hyperdocument contained about 90 nodes totally. The PBL net contained about 50 typed nodes and about 130 typed links, not counting typed nodes and links, which were removed during editing. The PBL net schema was tailored to each stage of the learning process, by making different node and link types available. The typed nodes and links supported them to construct shared knowledge corresponding to each stage of the PBL process. Although the users found the PBL net useful, they considered the restrictions of the computer screen width to be a difficulty. They, therefore, created separate whiteboards, which contained different sub nets as they moved through the stages of the learning process. They created one problem brainstorming net, containing mainly problem nodes, and networked them by using the "is_a_sub_of" link type. The second net was more varied, in which the users focussed specifically on the issues which they needed to learn about. They used this net to develop their learning plan and allocate tasks. They then used separate workspaces to collect information individually, but organized the results of the investigation using a third net, including the node types "resource", "evidence", "principle", "hypothesis", and "solution". Because the synchronous work was done primarily in a co-located mode, they rarely used temporary node types such as "question", "answer", and "hint".

Overall, our experience indicates that the system enables users to easily understand the local learning context, to intuitively navigate around, interact with, and tailor the virtual institute. The PBL net supports meaningful thinking and meaningful learning, and facilitates social interaction and social construction of knowledge. The trial also raised two questions. The first question is how to handle the situation, in which a user want to navigate a document following the structure of a PBL hyperdocument by using an document editor in place A, but the document is currently used by another user in place B. The second question is how to manage the size of the PBL-net that is created so as to maintain a good overview.

We currently work on a larger test use in winter, which will be conducted at Darmstadt University of Technology.

7 Conclusions and Future Work

This chapter concludes the research work described in this thesis. This research work is motivated by developments in the area of learning theories and learning methods and by the technological advances in hypertext/hypermedia, CSCW, and CSCL. The goal of this thesis work is to develop concepts and an approach for building collaborative learning environments for PBL, and to demonstrate the feasibility of the approach by describing a sample virtual PBL environment. This chapter highlights the main contributions of this thesis work by summarizing this thesis and comparing it to the state of art (with respect to approaches and prototype systems). Finally, it identifies a number of directions in which this research work might be extended.

7.1 Main Contributions of this Thesis

In this thesis a collaborative virtual PBL environment has been designed and implemented. This environment consists of four components: the virtual institute metaphor, the PBL-net, the PBL-protocol, and the PBL-plan. The virtual institute metaphor is designed and implemented to organize learning contexts, to support rich forms of social interaction, and to facilitate orientation in and tailoring of the virtual learning environment. The PBL-net provides a graphical knowledge representation language for PBL, which facilitates the pursuit of mutual understanding and the construction of shared knowledge. The PBL-protocol offers a role-based and state-dependent access control mechanism, which can support situated roles. The PBL-plan enables learning groups to define their own learning plans in a computational form. Such a learning plan can be automatically executed to coordinate actions. This section presents the major contributions of this research work.

7.1.1 A Conceptual Framework for the Development of Virtual PBL Environments

Based on activity theory, *a conceptual framework for building virtual learning environments* (see section 4.1) has been developed [Miao00e]. Through analyzing the characteristics of PBL processes based on a scenario and on literature, especially from the perspective of activity theory, the requirements to develop computer-based collaborative learning environments for PBL have been systematically identified. It is proposed that the roles of cultural and social mediation should be addressed to support PBL activities conducted in virtual learning environments. The conceptual framework consists of eight components: *agent, place, tool, language, document, action, work description, and behavior rule* (see subsection 4.1.3). It is suggested that these eight components should be modeled in virtual PBL environments appropriately. In addition, important design issues and possible design choices for developing virtual PBL environments are discussed. This conceptual framework can be used as a basis to analyze existing virtual PBL environments and as a guideline to design a virtual PBL environment. This conceptual framework has been applied to design a virtual PBL environment, in which cultural and social mediation in the PBL activity is reflected by four concepts: *the virtual institute metaphor, the PBL-net, the PBL-protocol, and the PBL-plan*.

7.1.2 A Context-based Approach to the Design of Collaborative Virtual Learning Environments: the Virtual Institute Metaphor

Based on the theory of situated learning, which emphasizes the importance of context and social interactions, the concept of *learning context* and an approach to develop *context-based virtual learning environments* (see section 4.3) have been developed [Miao99d]. The collaborative learning environments developed by adopting this approach enable the learners themselves to create and modify their learning environments. They therefore support customized learning contexts in which learning processes and interaction between learners can be situated. Comparison with *document-based approach, conferencing-based approach, and room-based approach,* (see subsection 4.3.4) the characteristics of *context-based approach* are the use of a set of perceptual metaphors, the flexible combination of these metaphors within the learning environment, and the support for awareness of the learning contexts and for the social interaction within it. This approach is used to design a hierarchically structured learning context, called as a *virtual institute metaphor* (see subsection 4.3.3). The virtual institute metaphor reflects parts of the culture used in learning environments without computers. A context-based virtual learning environment enables users to customize learning contexts at will, to intuitively navigate within the virtual learning environment, and to interact with learning contexts as they do in real world.

7.1.3 An Activity-oriented Approach towards a Graphical Knowledge Representation: the PBL-net

Based on the theory of constructivism and situated learning, an *abstract model of collaborative learning* (see subsection 4.4.1) is developed [Miao00a]. This model addresses the conflict at the individual memory level and at the group memory level. Considering the state-of-the-art in terms of graphical knowledge representation methods, it is found that support for the resolution of such conflict and support of negotiation during collaborative learning in virtual learning environments are both insufficient. It is also found that existing graphical knowledge representation approaches such as the *content-based approach* and *didactic-oriented approach* are not suitable to support the PBL activity. In this thesis, an *activity-oriented approach to visually represent structured knowledge* (see subsection 4.4.3) is developed [Miao00a]. By adopting this approach, a virtual learning environment can support the construction of graphical, shared knowledge in a shared information space. This approach is appropriate for supporting the representation, exploration and negotiation of shared knowledge in ill-structured knowledge domains. As an application of the activity-oriented approach, the *PBL-net schema* (see subsection 4.4.4), a graphical knowledge representation language for PBL is developed. By means of such a knowledge representation language, users can construct a *PBL-net*, representing their shared knowledge. The PBL-net mediates PBL processes by providing a metacognitive framework that facilitates and guides collaborative learning to pursue common understanding and to construct shared knowledge in the PBL processes.

7.1.4 An Approach to Guide and Control Social Interaction: the PBL-protocol

The concept of *multi-state collaborative process* (see subsection 4.5.2) has been developed and its characteristics have been identified. A multi-state collaborative process consists of multiple states with state transitions between them. Under each state, different behaviors of people with different roles are expected. As state changes, the expected behaviors of each role change as well. Based on schema theory, the concept of *collaboration protocol* and an *approach to model and execute collaboration protocol* (see subsection 4.5.2.1 and subsection 4.5.2.2) have been developed [Miao98b]. A collaboration protocol is a computerized script of a collaboration strategy, which explicitly specifies the expected behaviors of each role in the social interaction. The execution of a collaboration protocol supports role-based and state-dependent access control. A collaboration protocol can be refined by sub-protocols and can be initiated as protocol instances. An *algorithm to shift between collaboration protocols* (see subsection 4.5.2.3) is developed [Miao00c]. The approach is applied to develop *PBL-protocols* (see subsection 4.5.3) that support PBL groups within virtual learning environments [Miao00b]. The resulting PBL-protocols are used to guide and control construction of the shared PBL-nets by suggesting and restricting behaviors of roles in each state of the PBL process. In addition, the idea of sub-protocol is used to develop a *negotiation protocol* (see subsection 4.5.3.2), which can facilitate negotiation processes for the construction of shared knowledge. The PBL-protocol mediates PBL processes at the operation level and coordinates the contributions of people based on their different roles.

7.1.5 An Approach to Support Session-based Collaborative Processes: the PBL-plan

The concept of *session-based collaborative processes* (see subsection 4.6.2) has been developed to address supporting multi-phase synchronous collaboration processes [Miao98a] [Miao99a]. The term *session* is defined as that a process is executed in a synchronous or asynchronous collaboration mode on a shared workspace by a group of people to achieve a goal. The notion of a *session-based collaboration process* denotes the whole work process that consists of a set of coordinated sessions. An *approach to support the session-based collaboration processes* in computer-based collaboration environments is developed. Based on an analysis of the PBL process from the perspective of self-directed learning theory, this approach is adapted and applied to support PBL processes in virtual learning environments. By using this approach, a PBL process can be described in a computerized form, called a *PBL-plan* (see subsection 4.6.2) [Miao00d]. In addition, in order to ease making-plan, mechanisms to automatically create a preliminary PBL-plan and to support interactive modification and refinement of a PBL-plan are provided (see subsection 4.6.3 and subsection 4.6.4). When compared with existing workflow management systems, this approach has four major characteristics. Firstly, the idea of session-based collaborative processes is used to capture processes that consist of a set of coordinated actions. Each action is executed on a shared workspace by a group of people employing a synchronous or asynchronous collaboration mode. Secondly, a visual process modeling language can be used to describe a PBL process that consists of the components and relations (including temporal and artifact relations) between the components, and to represent process properties and constrains. Thirdly, a collaborative tool supports the definition of PBL processes as a hyperdocument. It provides mechanisms to create a preliminary PBL-plan and to enable interactive modification and refinement of a PBL-plan in order to ease the definition of PBL-plans. Fourthly, a cooperative environment is provided to execute the session-based collaborative processes. Users can join an action by selecting an action item from calendars or from the hyperdocument that represents the defined PBL-plan. Learners can also manipulate and monitor the state of actions by using calendars or the hyperdocument. An action can be triggered by using various ways. Multiple participants can collaboratively perform an action by using shared tools and documents available in the shared workspace of the action. The artifacts can be transferred from one place to others by using message-boxes according to the definition of the PBL-plan. The PBL-plan mediates PBL processes at the action level and coordinates the contributions of people made in different tasks.

7.1.6 Extension and Application of a Cooperative Hypermedia Model

When implementing the system, cooperative hypermedia technology is used to support representation and manipulation of the virtual institute metaphor, the PBL-net, the PBL-protocol, and the PBL-plan (see section 5.3).

The virtual institute is represented based on an *extended hypermedia model* [Miao01] that has two distinguished natures. Firstly, a place is modeled as a node and a door is modeled as a bi-directional hyperlink. This hyperlink has a status, which value is open or closed. When a door is closed on one side, people in both sides can not move from one side of the door to the other side of the door. Such a hyperlink provides a flexible navigation control mechanism. Secondly, most of metaphors and even activities are visually presented as graphical nodes and links of hypermedia documents. Therefore, users of the system can easily perceive learning environments and navigate within learning environments intuitively. They can use tools and documents by manipulating the corresponding graphical nodes directly and can interact with other people in the virtual learning environment. In addition, information in a virtual institute is structured and represented as hypermedia documents. A PBL-net is represented as a hypermedia document, from which other hypermedia documents can be reached by clicking on the typed nodes of the PBL-net. Therefore, a PBL-net and the associated documents can be manipulated as a logical whole by multiple users.

The PBL-protocol and the PBL-plan are described as hypertext documents. The process structure and process properties are represented by means of hypertext nodes and links and their attributes. For example, the set of behavior rules bound to a protocol state is represented as an attribute of a node that represents the protocol state. The processes on different abstract levels and different segments of a process are represented as individual nodes, but the hyperlinks are used to represent decomposition relations (such as sub-protocol and sub-plan) between them. The enactment mechanisms of processes are built on the properties of the hypertext document.

It is important to note that the concepts and approaches described above can be applied not only to PBL, but also to other domains. In fact, some concepts and approaches (such as collaboration protocol and session-based collaborative process) were originally developed for supporting collaborative design [Miao98a] [Miao99a]. These approachs can be used to support any collaborative process that has the identified characteristics (those of the multi-state collaborative process or the session-based collaborative process).

7.2 Comparison with other PBL Support Systems

This section compares CROCODILE with the PBL support systems discussed in chapter 3 based on the requirements identified in chapter 2. Generally speaking, other PBL support systems focus on supporting collaborative PBL between homogenous learners of high schools or of universities. These systems focus mainly on supporting science inquiry. In addition, most of them are implemented based on the Web, so that a large number of users can conduct PBL primarily in an asynchronous collaboration mode. However, the potential users of CROCODILE are adult learners who come from different background and want to improve their competence in their professional career. The problems used to drive collaborative learning are usually more authentic, complex, and related to their professional work. The PBL processes carried out by using CROCODILE are usually arranged and scheduled and are performed primarily in synchronous collaboration mode. In addition, other systems often emphasize support of one or a limited number of aspects of the PBL process (i.e., text-based communication, synchronous collaboration, identification of problems and learning issues, and so on). CROCODILE is developed based on a conceptual framework and intends to provides a systematical, complete support for PBL.

CROCODILE enables users to construct and customize their virtual institute and to navigate within it. Any user can socially present himself/herself by the position and movement of the user's picture. Detailed information about the user is available via the user's picture, which can be retrieved by others when clicking the picture. The user can interact with other users who are located in the same place or in different places by using tools such as conversation tools, message-boxes, and phones. Multiple users can construct shared knowledge synchronously or asynchronously by using private hypermedia editors, computers, and whiteboards. Other PBL support systems are developed as document-based systems (CSILE [Scardamalia92, Scardamalia96], CALE [Mahling95], CNB [Edelson95], McBAGEL [Narayanan95, Guzdial96], and Web-SMILE [Guzdial97, Kolodner98]) or conferencing-based systems (CCL [Koschmann90, Koschmann92] and Belvedere [Suthers97, Suthers99b]. In these systems, the collaborative PBL activity is mainly mediated by the shared information spaces. Some systems such as CSILE and Belvedere support communication by providing a chatboard that is used separately from the shared information spaces. All these systems do not address the problems of social orientation, group awareness, and customizing learning contexts.

CROCODILE enables users to construct a shared PBL-net that consists of typed nodes and typed links. The types indicate the users' intention when creating the piece of information. More detailed or related information can be accessed by following the hyperlinks. The negotiation of different perspectives about a statement is supported by the declaration function and the negotiation protocol. All other PBL support systems except for CCL enable users to categorize their ideas more or less and to provide detailed information and connect it to their ideas. In these systems, different perspectives are represented by separate statements so that the inconsistent knowledge has to be detected by carefully reading the separate statements. These systems support users to organize their ideas by using discussion threads, except for Belvedere that supports graphical knowledge representation. However, Belvedere only supports science inquiry activity, which is a simple form of PBL. For example, Belvedere does not support collaborative generation of a solution to solve the problem under study.

CROCODILE uses PBL-protocols to coordinate group interaction at the operation level. A PBL-protocol defines a number of possible states. In each state, the responsibilities and obligations of each role are clearly defined as behavior rules. At different states, behavior rules associated to a certain role may be different. The PBL-protocol not only guides social interaction by suggesting appropriate behaviors for each role, but also controls social interaction by restricting unexpected behaviors. In order to support a collaboration process, a family of protocols can be developed for supporting different collaboration strategies. Users can use one of them on demand and exploit sub-protocols to get more fine support. It is possible to change protocols in the course of using a protocol. In comparison with other PBL support systems, only Belvedere and Web-SMILE provide the definition of distinct states and transitions as a diagram. They also provide guidance for users to perform the focal task at each state. However, these systems do not attempt to support a synchronized collaborative activity. That is, users can work under different states at a point in time. They can not navigate together as collaboration state changes. In addition, the behavior rules associated with different roles can not be defined and bound to collaboration states in these two systems. Therefore, these two systems can not control social interaction to avoid unexpected behaviors. Furthermore, it is impossible in these systems to support change of collaboration strategies.

CROCODILE uses the PBL-plan to coordinate actions. Each action is performed in a place individually or as a team to achieve a sub-goal. Users can use the PBL-plan definition tool to describe their work process by decomposing goals, arranging actions to achieve each sub-goal, dividing labor, allocating place and resources, determining the conditions to start and terminate each action, and specifying the relations between actions. The system also supports the execution and monitoring of the PBL-plan. Furthermore, mechanisms to create a preliminary PBL-plan and to support interactive modification and refinement of a PBL-plan are provided in order to release some users' burden to define a PBL-plan. Some PBL support systems described in chapter 3 enable users to define actions. However, these actions are represented as a list of isolated commitments. Most of the information necessary to enact actions of a process is missing in these systems. Therefore, these systems can not support the coordination of actions by executing and monitoring the defined learning plans. Except for CALE, which provides primitive support for defining actions, no system supports the creation of a preliminary learning plan and the detection of conflicting allocation of shared resources and participants on demand. The following table summarizes the discussion above.

	CCL	CSILE	CALE	CNB	Belvedere	McBAGEL	Web-SMILE	CROCODILE
R1.1	∅	∅	∅	∅	∅	∅	∅	+
R1.2	∅	∅	∅	∅	∅	∅	∅	+
R1.3	-	-	-	-	-	-	-	+
R1.4	∅	∅	∅	∅	∅	∅	∅	+
R2.1	∅	+	+	+	+	+	+	+
R2.2	∅	∅	∅	∅	-	∅	∅	+
R2.3	∅	+	+	+	∅	∅	+	+
R2.4	∅	-	-	-	-	-	-	+
R3.1	∅	+	+	∅	∅	∅	∅	+
R3.2	∅	∅	∅	∅	+	-	+	+
R3.3	∅	∅	∅	∅	∅	∅	∅	+
R3.4	∅	∅	∅	∅	∅	∅	∅	+
R4.1	∅	∅	-	-	∅	-	∅	+
R4.2	∅	∅	-	∅	∅	∅	∅	+
R4.3	∅	∅	-	∅	∅	∅	∅	+
R4.4	∅	∅	∅	∅	∅	∅	∅	+

Table 7.1: Comparison of CROCODILE with Other PBL Support Systems

Table 7.1 summarizes to what degree systems fulfill the requirements presented in the chapter 3. Like Table 3.1, the notion "∅" indicates "no support". The notion "+" indicates "full support". The notion "-" means "partial fulfill" or "weak support". The requirements identified in chapter 2 are listed below:

(R1.1): support social orientation
(R1.2): support group awareness
(R1.3): support rich forms of social interaction
(R1.4): support customization of learning environments

(R2.1): support representation of various types of ideas
(R2.2): support representation of relations between ideas
(R2.3): support provision of referential information
(R2.4): support negotiation of shared knowledge

(R3.1): support definition of roles
(R3.2): provision of guidance according to PBL strategies
(R3.3): support synchronization of collaborative activities
(R3.4): support shifting between PBL strategies

(R4.1): support definition of action plans
(R4.2): support allocation of resources.
(R4.3): release users' burden to make action plans
(R4.4): support execution of action plans

7.3 Open Issues

While this research work provides a foundation for the development of comprehensive virtual PBL environments, a number of open issues are raised by this research. The purpose of this section is to suggest directions for future research based on this work.

7.3.1 Short-term Directions

From this research, several issues arose regarding conceptual framework, the virtual institute metaphor, the PBL-net, the PBL-protocol, and the PBL-plan. This subsection discusses these issues in detail.

Conceptual framework

The conceptual framework developed based on activity theory is still abstract, because activity theory provides a very abstract model of an activity. In order to support PBL, more PBL-specific cultural and social characteristics should be identified from the perspective of activity theory. For example, traditional methods for evaluating students' learning such as test sheet with standard answers are not suitable for PBL. The PBL-specific culture and social relations are used during evaluating students' learning. How to design computational mechanisms to support PBL-specific characteristics is a research issue. The design issues and the possible choices for these design issues should be more systematically taken into account. Especially, which kind of combination of choices for every design issue is appropriate for supporting a given form of PBL?

Virtual institute metaphor

Usually a virtual learning environment is developed as a database system, a meeting or conference system, or a room-based system. Information in such systems is structured and distributed according to a unique criterion such as information topic, a class, or a room. Users retrieve information using the same criterion. In a context-based system information is structured and distributed according to multiple co-existing criteria such as document owner, document relationships in contents or in references, where documents are stored or handled in tools, and where they are located. On the one hand, a context-based system enables users to retrieve information in multiple and flexible ways; On the other hand, some situation will confuse users. For example, when a user navigates a document following the structure of a PBL hyperdocument by using an document editor in place A, the document may be currently used by another user in place B. They may work on the same document, but they are not at the same place. Such situations can not happen in the real world. How to handle such a problem is a research issue.

The PBL-net

The usefulness of defined node types and link types should be evaluated. The result of such an evaluation may suggest adding new types or removing defined types. The current version of PBL-net schema is a generic knowledge representation language for PBL. There are different forms of PBL such as project-based learning, science inquiry, and so on. For supporting each concrete form of PBL, more refined PBL-net schemas should be developed. In addition, the patterns of conflict perspectives such as *flat conflict*, *single point conflict*, and *multiple-point conflict* should be defined and identified carefully in order to support learners to resolve conflicts. Another research issue is the choice between a plain PBL-net or a nested PBL-net. A plain PBL-net offers a better overview and direct operation on any information node. However, when the number of nodes and links in a PBL-net increases, the PBL-net editing area seems too crowded because of the limitation of the editing area in space. In unlimited space, orientation becomes difficult, too. A nested PBL-net solves the problem existing in the plain PBL-net, but users can not get a whole picture of a PBL-net at one sight.

PBL-protocol

Learning groups are different in group size, group structure, and geographical distribution. Group members vary in age, sex, knowledge, skills, interests, learning style, and so on. A hypothesis is that for a certain group with some characteristics or in some special situations there exists one or more PBL-protocols, which are suitable for such a group. How to define and refine PBL-protocols for different kinds of learning groups is a research issue. Furthermore, is it possible that the virtual learning environment actively suggests one or some PBL-protocols for a learning group according to the execution situation? More negotiation protocols should be researched in order to help learners to resolve different patterns of conflict perspectives described above. How can the system automatically initiate an appropriate negotiation protocol according to the confidences of learners' perspectives, the confict degree and the conflict pattern of their perspectives?

PBL-plan

A friendlier user interface for the definition of active-/terminated-condictions should be provided, so that the users who are not familiar with predicate logic can express active-/terminated-condictions.

7.3.2 Long-term Directions

This subsection identifies long-term research directions based on this research work. Generally speaking, the system should be extended to support learning at work and learning for work. Initial research work in this direction was presented in [Miao99c]. Such a system would support a group of people who work together on the same project and are committed to continuous improvement of their work processes. More work can be done in this general direction:

First of all, the system can be extended to provide integrated support for problem-based learning and problem solving. Problem solving is an activity that has many commonalities with problem-based learning activity. The main goal of problem solving is to propose effective solutions to a problem and solve it. Learning activities may occur within the problem solving processes, but learning is just a means for achieving the goal. The main goal of problem-based learning is not to solve a problem, but to acquire knowledge surrounding the problem and to improve the abilities and skills through solving a problem. However, the tasks performed and information types handled in both activities are almost the same. These two activities are normally carried out by a group of people. In some cases, the subject and object of these two activities may be the same, and these two processes intertwine. The problem raised in the real work can be used as the starting point of a PBL process. Such a problem provides rich authentic and social contexts that are good for PBL. The knowledge and skills acquired in the learning processes can then be applied immediately to work.

Project-based learning is a form of complex, *learning-by-doing* PBL. This learning method is typically applied in learning domain-specific knowledge, such as software engineering and civil engineering. The domain-specific knowledge models and task-specific tools will be used in the learning processes. This learning method can be used in professional training centers and at the work settings. When conducting project-based learning in a work setting, the collaborative learning process and the collaborative work process are intertwined. In this situation, many questions arise. Whether the virtual institute can be extended by adding components such as virtual company and virtual office to form a virtual world? How to model the community? The workers are also learners and the learning groups and working groups are intertwined. How to manage documents so that work documents and learning documents can be shared in both processes securely? How can the PBL-net schema be extended into domain-specific instantiations of this representation? What protocols should be developed? How to integrate a session-based collaborative process support system with business process support systems (e.g., workflow systems)? How to implement a scaleable and longitudinal collaborative system?

8 References

[The Activity System] The Activity System. Available in URL: http://www.helsinki.fi/~jengestr/activity/6b.htm

[Abbott94] Abbott, K., and Sarin, S. (1994). Experiences with workflow management: Issues for the next generation. In *Proceedings of the Conference on CSCW'*94, Chapel Hill, USA. ACM, pp. 113-120.

[Aspy93] Aspy, D.N., Aspy, C. B., and Quimby, P.M. (1993). What doctors can teach teachers about problem-based learning. *Educational Leadership*, Vol. 50, No. 7, pp. 22-24.

[Baloian95] Baloian, N., Hoppe, U., and Kling, U. (1995). Structured Authoring and Cooperatrive Use of Instructional Multimedia for a Computer-integrated Classroom. In: *Proceedings of ED-MEDIA'95*.

[Barden94] Barden, R., Stepney, S., and Cooper, D. (1994). Z in Practice. BCS Practitioner Series. Prentice-Hall, 1994.

[Barrows80] Barrows, H.S. and Tamblyn, R.M. (1980). Problem-Based Learning. An Approach to Medical Education. Springer Publishing Company : New York.

[Belvedere Webpages] Available in URL: http://advlearn.lrdc.pitt.edu/belvedere/tours/quad.html

[Boud91] Boud, D., and Feletti, G. (1991). The Challenge of Problem-Based Learning, London: Kogan Page.

[Bridges92] Bridges, E. M. (1992). Problem based learning for administrators. Eugene, OR: ERIC Clearinghouse on Educational Management. (ERIC Document Reproduction Service No. ED pp. 347, 617)

[Brockett91] Brockett, R G, Hiemstra, R (1991). Self-direction in Learning: Perspectives in Theory, Research, and Practice. Routledge, London, UK

[Brooks93] Brooks, J.G., & Brooks, M.G. (1993). In search of understanding: The case for constructivist classrooms. Alexandria, VA: Association for Supervision and Curriculum.

[Brown89] Brown, J.S., Collins, A. and Duguid, S. (1989). Situated cognition and the culture of learning. *Educational Researcher*, Vol. 18, No. 1, pp.32-42. Also available in URL: http://www.ilt.columbia.edu/ilt/papers/JohnBrown.html

[Brown92] Brown, G. and M. Pendleberry (1992). Assessing Active Learning. Parts 1 and 2, CVCP Universities' Staff Development and Training Unit, UK.

[Brundage80] Brundage, D. and MacKeracher, D. (1980) Adult Learning Principles and Their Application to Program Planning. Toronto: The Minister of Education.

[Bullen91] Bullen, C., and Bennett, J. (1991). Groupware in Practice: An Interpretation of Woek Experiences. In C. Dunlop and R. Kling (eds.), *Computerization and Controversy*, Academic Press, 1991, pp. 257-287.

[Burtis97] Burtis, J. (1997). Sociocognitive Design Issues for Interactive Learning Environments Across Diverse Knowledge Building Communities. In: *Proceedings of The Annual Meeting of the American Educational Association*, Chicago, March 24, 1997.

[Busbach93] Busbach, U. and Kreifelts, T. (1993). Support for Meeting using the EuroCoOp Task Manager. S. Srivener (Hrsg) CSCW: *The Multimedia and Networking Paradigm*, Avebury Technical, Ashgate Publ. Ltd, 1993, p.149-170.

[Cameron99] Cameron, T., Barrows, H. S. and Crooks, S. M. (1999). Distributed Problem-Based Learning at Southern Illinois University School of Medicine. In *proceedings of Computer Support for Collaborative Learning'99*. December 12-15, 1999. Palo Alto, California.

[Camp96] Camp G. Problem-based learning: A paradigm shift or a passing fad? Med Educ Online 1996, Vol. 1, No. 2, 1996. Available in URL: http://www.med-ed-online.org/f0000003.htm

[COAST manual] (1996). Available in URL:
http://www.darmstadt.gmd.de/publish/ocean/software/coast/doku/COAST-manual-3.PDF

[Cognition and Technology Group at Vanderbilt 91] (1991). Some thoughts about constructivism and instructional design. *Educational Technology*, Vol. 39, No. 9, pp. 16-18.

[Collins89] Collins, A., Brown, J. S., and Newman, S. (1989). Cognitive Apprenticeship: Teachning the Crafts of Reading, Writing and Mathematics. In Lauren B. Resnick (Ed.), *Knowing, Learning, and Instruction*. Essays in Honor of Robert Glaser (pp. 453 - 494), Hillsdale, NJ.: Erlbaum.

[Conklin87a] Conklin, J. and Begeman, M. L. (1987). gIBIS: A hypertext tool for team design deliberation. In: *Proceedings of Hypertext'87 Proceedings*, Chapel Hill, NC, pp. 247-252. New York: ACM.

[Conklin87b] Conklin, J., (1987). Hypertext: An Introduction and Survey. *Computer*, Vol. 20, No. 9, pp17-41, 1987.

[Course Material] Available in URL:
http://www.rcc.ryerson.ca/learnontaria/idnm/mod2/mod2-5/mod2-5.htm

[CoVis Webpages] Available in URL: http://www.covis.nwu.edu/

[CSILE's Webpages] Available in URL: http://csile.oise.utoronto.ca/

[Cunningham93] Cunningham, D. (1993). Assessing Constructions and Constructing Assessments. In Duffy, T. and Jonassen, D. (Eds.), *Constructivism and the technology of instruction: A conversation*. Hillsdale, NJ: Lawrence Erlbaum.

[Curtis92] Curtis, B., Kellner, M.I., and Over, J. (1992). Process Modelling. *communications of the ACM*, Vol. 35, No. 9, pp.75-90. Sep., 1992.

[Dewey38a] Dewey, J. (1938) Experience and Education. New York: Collier and Kappa Delta Pi.

[Dewey38b] Dewey, J. (1938). Logic: The Theory of Inquiry, New York: Holt and Co.

[Dolmans94] Dolmans, D., Schmidt, H. and Gijselaers, W. (1994) The relationship between student-generated learning issues and self-study in problem-based learning. *Instructional Science*, Vol. 22, pp.251–267, 1994.

[Duffy96] Duffy, T. M., and Cunningham, D. J. (1996). Constructivism: Implications for the design and delivery of instruction. In D. H. Jonassen (Ed.), *Handbook of Research for Educational Communications and Technology* (pp. 170-198). New York: Simon & Schuster Macmillan.

[Duncan98] Duncan-Hewitt W. and Mount D. (1998). Problem-based learning by Grian. Available in URL: http://www.grian.com/pblpage/pbl1.html

[Edelson94] Edelson, D.C., and O'Neill, D.K. (1994). The CoVis Collaboratory Notebook: Computer Support for Scientific Inquiry. Presented at the *Annual Meeting of the American Educational Research Association*, New Orleans, LA, April 1994, as part of a symposium entitled Computer-Supported Collaboration for Scientific Inquiry: Bringing Science Learning Closer to Science Practice.

[Edelson95] Edelson, D., O'Neill, K., Gomez, L., D'Amico, L. (1995). A design for effective support of inquiry and collaboration. In J. Schnase and E. Cunnius (Eds*.), Proceedings of the Conference on Computer Support for Collaborative Learning*, pp. 107-111. Mahwah, NJ: Lawrence Erlbaum.

[Edelson96a] Edelson, D., Pea, R., and Gomez, L. (1996). Constructivism in the Collaboratory. In B. G. Wilson (Ed.), *Constructivist learning environments: Case studies in instructional design,* (pp. 151-164). Englewood Cliffs, NJ: Educational Technology Publications.

[Edelson96b] Edelson, D., Pea, R., and Gomez, L. (1996). The Collaboratory Notebook. *Communications of the ACM*, Vol. 39, No. 4, pp. 32-33.

[Education by Design] Available in URL:
http://www.edbydesign.org/foundations/ebd_theory.htm

[Ehrlich87] Ehrlich, S. F. (1987). Strategies for encouraging successful adoption of office communication systems. *ACM Transactions on Office Information Systems*. Vol. 5, pp.340-357, 1987.

[Ellis94] Ellis, C. and Wainer, J. (1994). A Conceptual Model of Groupware. In: *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'94).* Chapel Hill. N. C., USA (October 22-26, 1994) ACM Press pp.79-88, 1994.

[Engestroem87] Engestroem, Y. (1987). Learning by expanding: An activity-theoretical approach to developmental research. Helsinki: Orienta-Konsultit.

[Ertmer93] Ertmer, P. A. and Newby, T. J. (1993). Behaviorism, cognitivism, constructivism: comparing critical features from an instructional design perspective. Performance Improvement Quarterly, Vol. 6, No. 4, pp. 50-72.

[Felder88] Felder, R.M. and L. Silverman (1988) Learning and teaching styles in engineering education. *Eng. Ed.*, Vol. 78, No. 7, pp.674-681.

[Georgakopoulos95] Georgakopoulos, D., Hornick, M., and Sheth, A. (1995). An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, Vol. 3, No. 2, 1995.

[Greeno98] Greeno, J. G., and the Middle School Mathematics Through Applications Projects Group. (1998). The situativity of knowing, learning, and research. *American Psychologist*, Vol. 53, No. 1, pp. 5-26.

[Grabe98] Grabe, M. and Grabe C. (1998). Integrating Technology for Meaningful Learning 2e. Houghton Mifflin.

[Grayson74] Grayson, L.P., and J.M. Biedenbach (1974). Individualized Instruction in Engineering Education. ASEE, Washington, DC.

[Grudin94] Grudin, J. (1994). Eight Challenges for Developers. *Communications of the ACM*, Vol. 37, No. 1, Jan., 1994.

[Guzdial96] Guzdial, M., Kolodner J.L., Hmelo, C, Narayanan, H., Carlson, D., Rappin, N., Huebscher, R., Turns, J., and Newstetter, W. (1997). Computer Support for Learning through Complex Problem Solving. *Communication of the ACM*, Vol. 39, No. 4, pp. 43-45, April 1996.

[Guzdial97] Guzdial, M., Hmelo, C, Hübscher, R., Nagel, K., Newstetter, W., Puntembakar, S., Shabo, A., Turns, J., and Kolodner J.L., (1997). Integrating and Guiding Collaboration: Lessons learned in computer-supported collaboration learning research at Georgia Tech. In R. Hall, N. Miyake, & N. Enyedy (Eds.), *Proceedings of Computer-Supported Collaborative Learning '97*, pp. 91-100. Toronto, Ontario, Canada.

[Haake92] Haake, J. M., and Wilson, B. (1992) Supporting Collaborative Writing of Hyperdocuments in SEPIA. In: *Proceedings of ACM CSCW'92*, pp. 138-146.

[Hennessy92] Hennessy, P., Kreifelts, T., and Ehrlich, U. (1992). Distributed work management: activity coordination within the EuroCoOp project. *Computer Communications,* Vol. 15, No. 8, October 1992, p.477-488.

[Hiemstra90] Hiemstra, R., and Sisco, B. (1990). Individualizing instruction for adult learners: Making learning personal, powerful, and successful. San Francisco: Jossey-Bass.

[Hiemstra94] Hiemstra, R. (1994). Self-directed learning. In T. Husen & T. N. Postlethwaite (Eds.), The International Encyclopedia of Education (second edition), Oxford: Pergamon Press. Available in URL: http://home.twcny.rr.com/hiemstra/sdlhdbk.html

[Hmelo95] Hmelo, C. E., Vanegas, J. A., Realff, M., Bras, B., Mulholland, J., Shikano, T., and Guzdial, M. (1995). Technology support for collaboration in a problem-based curriculum for sustainable technology. In J. L. Schnase & E. L. Cunnius (Eds.), *Computer Support for Collaborative Learning (CSCL '95)*, pp. 169-172.

[Huebscher96] Huebscher, R., Hmelo, C., Narayanan, N., Guzdial, M., and Kolodner, J. (1996). McBAGEL: A Shared and Structured Electronic Workspace for Problem-Based Learning. *Proceedings of the Second International Conference on the Learning Sciences*, Evanston, Illinois.

[Johnson82] Johnson, D.W. and Johnson, F.P. (1982) Joining Together. 2nd edition, Prentice Hall, Englewood Cliffs, NJ.

[Johnson91] Johnson, D.W., Johnson, R.T. and Smith, K.A. (1991) Active learning: cooperation in the college classroom. Interaction Books, Edina, MN.

[Jonassen91] Jonassen, D. (1991). Thinking technology: Context is everything. *Educational Technology*, Vol. 31, No. 6, pp. 35-37.

[Jonassen93] Jonassen, D.H., Beissner, K., and Yacci, M. (1993). Structural knowledge: Techniques for representing, conveying, and acquiring structural knowledge. Hillsdale: Erlbaum.

[Jones94] Jones, B., Valdez, G., Norakowski, J., & Rasmussen, C. (1994), Designing Learning and Technology for Educational Reform. North Central Regional Educational Laboratory.

[Keller68] Keller, F.S. (1968) Good-bye, Teacher. *Journal of Applied Behaviour Analysis*, Vol.1, pp.79-89.

[Kibler74] Kibler, R.J., D.J. Cegala, D.T. Miles and L.L. Barker (1974) Objectives for Instruction and Evaluation.

[Knowles75] Knowles, M. S. (1975) Self-Directed Learning: A Guide for Learners and Teachers. New York: Association Press, 1975.

[Knowles80] Knowles, M. S. (1980). The modern practice of adult education (revised and updated). Chicago: Follett Publishing Company.

[Knowles84] Knowles M.S. and Associates (1984) Andragogy in Action: Applying Modern Principles of Adult Learning. San Francisco: Jossey-Bass, Inc.

[Knowles86] Knowles, M. S. (1986). Using learning contracts. San Francisco: Jossey-Bass, Inc.

[Kolodner93] Kolodner, J. (1993). Case Based Reasoning. San Mateo, CA: Morgan Kaufmann Publishers.

[Kolodner98] Kolodner, J. L., Crismond, D., Gray, J., Holbrook, J., Puntambekar, S. (1998). Learning by Design from Theory to Practice. In A. Bruckman, M. Gudial, J. Kolodner, & A.

Ram (eds.), In: *Proceedings of International Conference of the Learning Sciences 1998,* pp. 16-22. Atlanta, Georgia.

[Koschmann90] Koschmann, T.D., Feltovich, P.S., Myers, A.M. and Barrows, H.S. Designing communication protocols for a computer-mediated tutorial laboratory for problem-based learning. *Proceedings of the 14th Annual Symposium on Computer Applications in Medical Care*, Los Alamitos, CA: IEEE Computer Society Press, 1990, pp. 464-468

[Koschmann92] Koschmann, T.D.,Feltovich, P.S., Myers, A.M. and Barrows, H.S. (1992), Implications of CSCL for Problem-Based Learning. *Barrows in the proceedings of the Spring '92 ACM Conference on Computer Supported Collaborative Learning*, Vol. 21, No.3, ACM Press.

[Koschmann97] Koschmann, T.D., Glenn, P., Conlee, M. (1997). Analyzing the Emergence of a Learning Issue in a Problem-Based Learning Meeting. *Med Educ Online*, Vol. 2, No. 2, 1997.

[Larkin87] Larkin, J.H. and Simon, H. A.. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science* Vol. 11, No. 1, pp. 69-99. 1987.

[Lave91a] Lave, J. and Wenger, E. (1991). Situated Learning: Legitimate Peripheral Participation. Cambridge, UK: *Cambridge University Press*.

[Lave91b] Lave, J. (1991). Situated learning in communities of practice. In L.B. Resnick, J.M. Levine, & S.D. Teasley (Eds). *Perspectives on socially shared cognition* (pp. 63-82). Washington, DC: American Pscyhological Association.

[LEARNING CONTRACTS] available in URL:
http://home.twcny.rr.com/hiemstra/contract.html

[Lemke97] Lemke, J. L. (1997). Cognition, context, and learning: A social semiotic perspective. In D. Kirschner & J. A. Whitson (Eds.), *Situated cognition: Social, semiotic and psychological perspectives* (pp. 37-55). Mahwah NJ: Erlbaum.

[Leontiev47] Leontiev, A.N. (1947). Problems of the Development of Mind. English translation, Moscow, 1981, Progress Press. (Russian original published 1947).

[Mager62] Mager, R.F. (1962). Preparing educational objectives. Fearon Publishers, San Francisco, CA.

[Mahling95] Mahling, D. E., Sorrows, B. B., and Skogseid, I. (1995). A Collaborative Environment for Semi-Structured Medical Problem Based Learning. In: *Proceedings of CSCL'95*.

[Mayo93] Mayo, P., Donnelly, M. B., Nash, P. P., & Schwartz, R. W. (1993). Student Perceptions of Tutor Effectiveness in problem based surgery clerkship. *Teaching and Learning in Medicine*. Vol. 5, No. 4, pp. 227-233.

[McDermott99] McDermott, R. (1999) Learning Across Teams: The Role of Communities of Practice in Team Organizations. *Knowledge Management Review*, May/June, 1999. Available in URL: http://www.co-i-l.com/coil/knowledge-garden/cop/learning.shtml

[Miao98a] Miao, Y. and Haake, J. M. (1998). Supporting Concurrent Design by Integrating Information Sharing and Activity Synchronization. In *Proceedings of the 5th ISPE International Conference on Concurrent Engineering Research and Applications (CE 98)*, pp. 165-174, Tokyo, Japan, July 15-17, 1998.

[Miao98b] Miao, Y. and Haake, J. M. (1998). Flexible Support for Group Interactions in Collaborative Design. In *Proceedings of the Third International Workshop on CSCW in Design (CSCWID 98)*, Tokyo, Japan, July 15-18, 1998.

[Miao99a] Miao, Y. and Haake, J. M. (1999). Supporting Concurrent Design in SCOPE. in *The International Journal of Concurrent Engineering Research and Applications* Vol. 7, No. 1, pp.55-66, March 1999.

[Miao99b] Miao, Y., Pfister, H. R., and Wessner, M. (1999). Combining the Metaphors of an Institute and of Networked Computers for Building Collaborative Learning Environments. Poster published in *Proceedings of the 4th Annual ACM SIGCSE/SIGCUE ITiCSE 99*, pp. 188, Cracow, Poland, June 27 - July 1, 1999.

[Miao99c] Miao, Y., Pfister, H. R., Wessner, M., and Haake, J. M. (1999). SCOPE: An Environment for Continuous Improvement Teams in Virtual Corporations. In *Proceedings of ED-MEDIA 99 - World Conference on Educational Multimedia, Hypermedia & Telecommunications*, pp. 957-962, June 19-24, 1999, Seattle, Washington, U.S.A.

[Miao99d] Miao, Y., Fleschutz, J.M., and Zentel, P. (1999). Enriching Learning Contexts to Support Communities of Practice. In: *Proceedings of the Computer Support for Collaborative Learning (CSCL'99)*, pp. 391-397, December 12-15, 1999. Palo Alto, California, U.S.A.

[Miao00a] Miao, Y., Holst, S., Holmer, T., Fleschutz, J.M., and Zentel, P. (2000). An Activity-Oriented Approach to Visually Structured Knowledge Representation for Problem-Based Learning in Virtual Learning Environments. In: *Proceedings of the Fourth International Conference on the Design of Cooperative Systems (COOP'2000)*, pp. 303-318. May 23-26, 2000. Sophia Antipolis, France.

[Miao00b] Miao, Y., Holst, S., Haake, J.M., and Steinmetz, R. (2000). PBL-protocols : Guiding and Controlling Problem Based Learning Processes in Virtual Learning Environments. In: *Proceedings of the Fourth International Conference on the Learning Sciences (ICLS'2000)*, pp. 232-237. June 14-17, 2000, Ann Arbor, U.S.A.

[Miao00c] Miao, Y., Haake, J.M., and Steinmetz, R. (2000). A Rule-based Method to Shift between Learning Protocols. In: *Proceedings of the ED-MEDIA 2000 - World Conference on Educational Multimedia, Hypermedia & Telecommunications.* June 26 - July 1, 2000, Montréal, Canada.

[Miao00d] Miao, Y. (2000). Supporting Self-directed Learning Processes in a Virtual Collaborative Problem Based Learning Environment. In: *Proceedings of the 2000 Americas Conference on Information Systems (AMCIS'2000)*, pp. 1784-1790, August 10-13, 2000, Long Beach, California, U.S.A.

[Miao00e] Miao, Y. (2000). An Activity Theoretical Approach to A Virtual Problem Based Learning Environment. In: *Proceedings of the 2000 International Conference on Information*

Society in the 21 Century: Emerging Technologies and New Challenges, pp. 647-654, November 5 - 8, 2000, Aizu, Japan.

[Miao01] Miao, Y. and Haake, J. M. (in press). Supporting Problem Based Learning by a Collaborative Virtual Environment: A Cooperative Hypermedia Approach. Accepted by *the 34th Hawaii International Conference on System Sciences (HICSS-34)*, January 3-6, 2001, Hawaii, U.S.A.

[Narayanan95] Narayanan, H., Hmelo, C., Petrushin, V., Newstetter, W., Guzdial, M., and Kolodner J.L., (1995). *Computer Support for Learning through Generative Problem Solving. In proceedings of Computer Support for Collaborative Learning'95.*

[Neal98] Neal, L. (1998). Tutorial on Distance Learning, at GMD-IPSI, on Sept 1, 1998.

[Norman92] Norman, G.R. and Schmidt, H.G. (1992). The psychological basis of problem-based learning: a review of the evidence, Academic Medicine, Vol. 67, pp. 557-565.

[Norman96] Norman, D. A. and Spohrer, J. C. (1996). Learner-Centered Education. *Communication of ACM*, Vol. 39, No. 4, pp.24-27, 1996.

[Novak84] Novak, J.D. and Gowin, D.B. (1984). Learning how to learn. New York: Cambrifge University Press.

[Novak89] Novak, J. (1989). Helping students learn how to learn: a view from a teacher-researcher. *Third Congress of Research and Teaching in Science and Mathematics*, Santiago de Compostela, Spain, Sept. reviewed in PS News 69.

[O'Neill94] O'Neill, D. K. (1994). The Collaboratory Notebook: A Networked Knowledge-Building Environment for Project Learning. In T. Ottmann & I. Tomek (Eds.), *Educational Multimedia and Hypermedia*, pp. 416-423. 1994. Charlottesville, VA: AACE.

[Oshima96] Oshima, J., and Scardamalia, M. (1996). Knowledge-building and conceptual change: An inquiry into student-directed construction of scientific explanations. In D. C. Edelson & E. A. Domeshek (Eds.), *Proceedings of the Second International Conference on the Learning Sciences*, Northwestern Univ., Evanston, IL.

[Osterweil87] Osterweil, L. (1987). Software processes are software too. in *Proc. 9th Int'l Conf. Software Eng.* , pp. 2-13, ACM Press, New York, 1987.

[Pea93] Pea, R. (1993). The collaborative visualization project*.* *Communications of the ACM*, Vol. 36, No. 5, pp60-63, May 1993.

[Perret93] Perret-Clermont, A. N. (1993). What is it that develops? Cognition and Instruction, 11, pp.197-205.

[Pfister98a] Pfister, H. R., Schuckmann, C., Beck-Wilson, J., and Wessner, M. (1998). The metaphor of virtual rooms in the cooperative learning environment CLear. Streitz, N., Konomi, S., and Burkhardt, H.J. (Eds.), *Cooperative Buildings. Integrating Information, Organization and Architecture*. pp. 107-113. Berlin: Springer.

[Pfister98b] Pfister, H. R., Wessner, M., Beck-Wilson, J., Miao, Y., and Steinmetz, R. (1998). Rooms, protocols, and nets: metaphors for computer-supported cooperative learning of distributed groups. In: *Proceedings of the Third International Conference on the Learning Sciences (ICLS-98)*, pp. 242-248, Dec. 16-19, 1998. Georgia Tech, Atlanta.

[Pfister99] Pfister, H. R., Wessner, M., Holmer, and Steinmetz, R. (1999). Negotiating about Shared Knowledge in a Cooperative Learning Environment. In: *Proceedings of Computer Support for Collaborative Learning (CSCL'99)*, pp. 454-457. December 12-15, 1999. Palo Alto, California.

[Piaget73] Piaget, Jean. (1973). To Understand is to Invent. New York: Grossman, 1973.

[Piaget77] Piaget, J. (1977). The development of thought: Equilibrium of cognitive structures. New York: Viking Press.

[Popham70] Popham, W.J. and E.L. Baker (1970). Establishing instructional goals. Prentice Hall, Englewood Cliffs, NJ.

[Pross99] Pross, H. (1999). Problem-Based Learning Handbook. Available in URL: http://meds.queensu.ca/medicine/pbl/pblprint.htm

[Puntambekar98] Puntambekar, S. and Kolodner, J. L. (1998). The design diary: development of a tool to support students learn science by design. *Proceedings of the Third International Conference on the Learning Sciences (ICLS'98)*, pp. 230-236. Dec. 16-19, 1998. Georgia Tech, Atlanta.

[Rhem98] Rhem, J. (1998). Problem-Based Learning: An Introduction. *The National Teaching & Learning Forum*, Vol. 8, No. 1, pp.1-4, December, 1998.

[Roseman96] Roseman, M. and Greenberg, S. (1996). TeamRooms: Network places for collaboration. In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'96),* pp. 325-333, November 16-20, 1996, Boston, Mass.

[Roschelle92] Roschelle, J. (1992). Reflections on Dewey and Technology for Situated Learning. Paper presented at *annual meeting of the American Educational Research Association*, San Francisco, CA.

[Roschelle95] Roschelle, J. & Teasley, S. (1995). The construction of shared knowledge in collabrative problem solving. In C.E. O'Malley (Ed) *Computer-supported collaborative learning*. Heidelberg: Springer-Verlag.

[Roth92] Roth, W.M. and Roychoudhury, A. (1992). The social construction of scientific concepts or the concept map as conscription device and tool for social thinking in high school science. *Science Education*, Vol. 76, No. 5, pp. 531-557, 1992.

[Savery95] Savery, J. R. and Duffy, T. M. (1995). Problem based learning: An instructional model and its constructivist framework. *Educational Technology*, Vol. 35, No. 5, pp. 31-38, 1995.

[Scardamalia89] Scardamalia, M., Bereiter, R. S., Swallow, M. J., and Woodruff (1989). Computer-supported intentional learning environment. *Journal of Educational Computing Research*, 5, 51-68.

[Scardamalia92] Scardamalia, M., Bereiter, C., Brett, C., Burtis, P. J., Calhoun, C., and Lea, N. S.(1992). Educational applications of a networked communal database. *Interactive Learning Environments*, 2, 45-71.

[Scardamalia94] Scardamalia, M., Bereiter, C., and Lamon, M. (1994). The CSILE project: Trying to bring the classroom into World 3. In K. McGilly (ed.), *Classroom lessons - Integrating cognitive theory and classroom practice* (pp. 201-228). Cambridge, MA: MIT Press.

[Scardamalia96] Scardamalia, M., and Bereiter, C. (1996). Student communities for the advancement of knowledge. *Communications of the ACM*, Vol. 39, No. 4, pp. 36-37, 1996.

[Schael96] Schael, T. (1996): Workflow Management Systems for Process Organisations. Berlin: Springer Verlag.

[Schank77] Schank, R. C., and Abelson, R. P. (1977). Scripts, plans, goals, and understanding. Hillsdale, NJ: Erlbaum.

[Schank82] Schank, R. C. (1982). Dynamic memory. Hillsdale, NJ: Erlbaum.

[Schuckmann96] Schuckmann, C., Kircher, L., Schuemmer, J., Haake, J. (1996). Designing Object-Oriented Synchronous Groupware With COAST. In *Proceedings of ACM CSCW'96*, pp. 30-38, 1996.

[Schmidt92] Schmidt, H. G., Henny, P. A., and de Vries, M. (1992). Comparing problem-based with conventional education: A review of the University of Limburg medical school experiment. Annals of Community-Oriented Education, 5, pp. 193-198.

[Script theory] Available in URL: http://www.gwu.edu/~tip/schank.html

[Soloway94] Soloway, E., Guzdial, M., Hay, K. (1994) Learner-centered design: The next challenge for HCI. *ACM Interactions*. Vol. 1, No. 2, pp. 36-48.

[Spivey89] Spivey, J. M. (1989). The Z Notation: A reference Manual. 2nd edition, Prentice Hall International Series in Computer Science, 1992.

[Stahl99] Stahl, G., Herrmann, T. (1999). Intertwining Perspectives and Negotiation. *In proceedings of ECSCW'99.*

[Stefik86] Stefik, M., et al. (1986). WYSIWIS Revised: Early Experiences with Multiuser Interfaces. In: *Proceedings of the Conference on Computer-supported Cooperative Work (CSCW'86)*, pp. 276-290, Dec. 3-5, 1986.

[Stepien93a] Stepien, W., and Gallagher, S. (1993). Problem-Based Learning: As authentic as it gets. *Educational Leadership*, pp. 25-28.

[Stepien93b] Stepien, W. J., Gallagher, S. A., and Workman, D. (1993). Problem-Based Learning for traditional and interdisciplinary classrooms. *Journal for the Education of the Gifted*, Vol. 4, pp. 338-345.

[Streiz89] Streitz, N.A., Hannemann, J., and Thuering, M. (1989). From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces. In: *Proceedings of the 2nd ACM Conference on Hypertext (Hypertext '89),* Pittsburgh, PA, November 5-8, 1989, pp. 343-364, 1989.

[Streiz92] Streitz, N., Haake, J., Hannemann, J., Lemke, A., Schuett, H., Schuler, W., and M. Thuering (1992). SEPIA: A cooperative hypermedia authoring environment. In: *Proc. of ACM Conference on Hypertext ECHT'92*, pp.11-22.

[Streitz94] Streitz, N. A., Geissler, J., Haake, J. and Hol, J. (1994). DOLPHIN: Integrated Meeting Support across LiveBoards, Local and Remote Desktop Environments. In: *Proceedings of ACM 1994 Conference on Computer-Supported Cooperative Work (CSCW'94),* pp. 345-358. Chapel Hill. N. C. , U.S.A., October 22-26, 1994, ACM Press.

[Suchman87] Suchman, L.A. (1987). Plans and Situated Actions: The problem of human-machine communication. Cambridge: Cambridge University Press, 1987.

[Summer Sleuths Program] Available in URL:
http://www.imsa.edu/team/cpbl/ipbln/sleuths/problems/frogs/index.html

[Summers99] Summers, B. *Official Microsoft® NetMeeting™Book*,
http://mspress.microsoft.com/prod/books/1546.htm

[Suthers95] Suthers, D. and Weiner, A. (1995). Groupware for developing critical discussion skills. In: *Proceedings of Computer Supported Cooperative Learning '95*, Bloomington, Indiana, October 17-20, 1995.

[Suthers97] Suthers, D., Toth, E., and Weiner, A. (1997). An Integrated Approach to Implementing Collaborative Inquiry in the Classroom. In R.Hall, N.Miyake, & N. Enyedy (Eds.), *Proceedings of Computer-Supported Collaborative Learning '97*, pp. 272-279, December 10-14, 1997. Toronto, Ontario, Canada.

[Suthers99a] Suthers, D. (1999). Representational Support for Collaborative Inquiry. In: *Proceedings of the Hawaii International Conference on System Sciences*, January 5-8, 1999.

[Suthers99b] Suthers, D. D. (1999). Effects of Alternate Representations of Evidential Relations on Collaborative Learning Discourse. In: *Proceedings of Conference on Computer Supported Collaborative Learning (CSCL99),* pp.611-621. Stanford, December 11-15, 1999.

[Swenson93] Swenson, K. D. (1993). Visual Support for Reengineering Work Processes. In: *Proceedings of the Conference on Organisational Computing Systems*, Nov. 1993.

[Tam00] Tam, M. (2000). Constructivism, Instructional Design, and Technology: Implications for Transforming Distance Learning. *Educational Technology & Society,* Vol. 3, No. 2, 2000. Available in URL: http://ifets.ieee.org/periodical/vol_2_2000/tam.html.

[TeamWARE] TeamWARE Flow. Available in URL:

http://www.teamware.com/homepage.htm

[Trigg83] Trigg, R. (1983). A Network-based Approach to Text Handling for the Online Scientific Community. PhD sissertation. Department of Computer Science, University of Maryland.

[Trilling99] Trilling B. and Hood P. (1999) Learning, Technology, and Education Reform in the Knowledge Age or "We're Wired, Webbed, and Windowed, Now What?". *EDUCATIONAL TECHNOLOGY*, May-June 1999. Available in URL: http://www.sasked.gov.sk.ca/~parkland/webbed.htm

[VonGlaserfeld84] Von Glaserfeld, E. (1984). Radical constructivism. In Watzlawick, P. (Ed.) *The invented reality*, Cambridge, MA: Harvard University Press, pp. 17-40.

[VonGlaserfeld89] Von Glaserfeld, E. (1989). Cognition, Construction of Knowledge, and Teaching, Synthese, 80, pp. 121-140.

[Vygotsky78] Vygotsky, L.S. (1978). Mind in society: the development of higher psychological processes. Cambridge: Harvard University Press.

[Wan94a] Wan, D. and Johnson, P. M. (1994). Computer Supported Collaborative Learning Using CLARE: The Approach and Experimental Findings. In: *Proceedings of the ACM CSCW94,* pp. 187-198, Oct. 22-26, 1994, Chapel Hill, NC.

[Wan94b] Wan, D. and Johnson, P. M. (1994). Experiences with CLARE: a Computer-Supported Collaborative Learning Environment. *International Journal of Human-Computer Studies*, October 1994.

[Wenger98] Wenger, E. (1998). Communities of Practice: Learning as a Social System, *The Systems Thinker*, Vol. 9, No. 5, June/July 1998. Pegasus Communications Inc., Waltham, MA, USA.

[Wessner99] Wessner, M., Pfister, H. R., and Miao, Y., (1999). Using Learning Protocols to Structure Computer-Supported Cooperative Learning. In: *Proceedings of the ED-MEDIA'99*, pp. 471-476, Seattle, Washington, June 19-24, 1999.

[WfMC] Workflow Management Coalition. Available in URL: http://www.aiim.org/wfmc/

[Winograd86] Winograd, T. and Flores, F. (1986). Understanding Computers and Cognition: A New Foundation for Design. Norwood, New Jersey: Ablex Publishing Corp.

[Wolfson] Available in URL:
http://www.knowarch.com/index/front_office/evaluation/year_three.html

[Woods85] Woods, D.R. (1985). Total Quality Management. McMaster University, Hamilton ON.

[Woods85] Woods, D.R. (1993) New Approaches for developing problem solving skills, J. College Science Teaching, Vol. 23, pp. 157-158.

[Woods96] Woods, D. R. (1996). Problem Based Learning: How to Get the Most from PBL, McMaster University, 3rd edition, March 1996.

[Zhao94] Zhao, J. and Hoppe, U. (1994). Supporting Flexible Communication in Heterogeneous Multi-User Environments. In: *Proc. of the 14th International Conf. on Distributed Computing Systems*, IEEE Computer Society Press. pp. 442-449, June, 1994.

[Zhao95] Zhao, J. and Hoppe, U. (1995). Getting Serious About Flexible UI Coupling. *Proc. of the International Workshop on the Design of Cooperative Systems*, pp. 507-515, France, January, 1995.

Appendix A: List of Definitions

A

Action 158
Actions 159
Actor 63
Agent 63
Artifact 159
Artifacts 159

B

Behavior rule 125
Bookshelf 68

C

Calendar 69
Collaboration Protocol 126
Community 63
Conflict Degree 116
Connection Nodes 160

D

Declaration 109
Document 67
Document Editor 68
Document Reference 67
Door 65
Door View 65

G

Group 63
Group Perspective 116

H

Hyperdocument Base 68

I

Institute Space 65

L

Label Relation Graph 139
Learning Context 71
Link Type 103

M

Message Box 68
Mutual Understanding 115

N

Net 104
Net Base 105
Net Schema 103
Node Type 103

O

Object 125
Operation 125

P

PBL-net 109
PBL-net Base 110
PBL-net Schema 107
PBL-plan 157
PBL-protocol 144
PBL-protocol Family 144
Place 64
Plan Base 164
Plans 158
Profession 177
Profession Issues 177
Protocol Base 126
Protocol Family 126
Protocol Instance 132
Protocol Instance Base 132
Protocol Role 125
Protocol State 125
Protocol Transition 126

S

Specific Tool 69

T

Temporal Relations 160
Tool Base 70
Typed Link 104
Typed Node 104

V

Virtual Institute 70

Appendix B: List of Figures

Figure 3.1: An Example WebCSILE Public Forum 28
Figure 3.2: Progressive Inquiring by Using Theory-building Scaffold...................... 29
Figure 3.3: The Table of Contents of a Notebook and the Content of a Page 31
Figure 3.4: Belvedere Inquiry Diagram.. 33
Figure 3.5: The McBAGEL's Whiteboard with Four Sub-spaces 34
Figure 3.6: The Problem-solving Flowchart in Web-SMILE 36
Figure 3.7: Web-SMILE's Whiteboard 37
Figure 3.8: Guidance Information for a Stage of the Process 38
Figure 4.1: The Structure of Human Activity 44
Figure 4.2: The Hierarchical Structure of Activity 44
Figure 4.3: The Conceptual Framework 46
Figure 4.4: The Conceptual Architecture of a Virtual Learning Environment........... 48
Figure 4.5: The Conceptual Architecture of a Virtual Institute 62
Figure 4.6: Conceptual Model of Collaborative Learning Between Two Learners.... 99
Figure 4.7: Conceptual Architecture of the Shared Artifact 103
Figure 4.8: PBL-Net Schema 109
Figure 4.9: A Typical Restaurant Script 121
Figure 4.10: Conceptual Architecture of Collaboration Protocols and
Protocol Instances 124
Figure 4.11: A PBL-protocol Diagram.. 145
Figure 4.12: A Sub-PBL-Protocol Diagram 149
Figure 4.13: Conceptual Architecture of the PBL-plan.. 157
Figure 4.14: The Sub-net of a PBL-net 179
Figure 4.15: Creating a Sub-plan of the Overall PBL-plan 180
Figure 4.16: Creating a Sub-plan of Another Sub-plan.. 180
Figure 4.17: Creating Actions in Sub-plans... 181
Figure 4.18: Creating Actions in the Overall PBL-plan 181
Figure 4.19: Creating Temporal Relationships 182
Figure 4.20: Creating an AndJoin Node and Temporal Relationships.................... 183
Figure 4.21: Creating an AndSplit Node and Temporal Relationships.................... 183
Figure 4.22: Creating a Artifact and Two Artifact Relationships 184
Figure 4.23: The Example Preliminary Learning Plan 185
Figure 5.1: System Architecture.. 206
Figure 5.2: Abstract Implementation Model.. 207
Figure 5.3: COAST Application Architecture 212
Figure 5.4: Application Architecture of Current Version of COAST 212
Figure 5.5: CROCODILE Architecture... 213
Figure 5.6: A Hyperdocument Representing a Virtual Institute 217
Figure 5.7: An Hyperdocument Representing a PBL-net....................................... 218
Figure 5.8: A Hyperdocument Representing two PBL-protocols 219
Figure 5.9: A Hyperdocument Representing two PBL-plans 221
Figure 6.1: The Campus of the Virtual Institute 223
Figure 6.2: The Corridor of the Dormitory.. 224
Figure 6.3: A Public Room 225
Figure 6.4: PBL-net Schema Editor 227
Figure 6.5: Learning Protocol Schema Editor.. 228
Figure 6.6: Group Definition Tool 229

Figure 6.7: Protocol Control Panel 230
Figure 6.8: PBL-plan Definition, Monitoring, and Execution tool 231
Figure 6.9: HyperDocument Editor and Browser 232
Figure 6.10: An Example PBL-net 233

Appendix C: List of Tables

Table 3.1: Comparison of Existing PBL Support Systems ………………………… 41
Table 7.1: Comparison CROCODILE with Other PBL Support Systems ……….. 243