REMEMEX: A SOFTWARE TOOL FOR KNOWLEDGE CONSTRUCTION AND ORGANIZATION

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION SCIENCE AND THE INSTUTE OF ENGINEERIG AND SCIENCE

OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGRE OF
MASTER OF SCIENCE

By

Özden Ermek Erarslan

November, 1999

QA
76.9
.S88
E73
1999

REMEMEX: A SOFTWARE TOOL FOR KNOWLEDGE CONSTRUCTION AND ORGANIZATION

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Özden Emek Erarslan

November, 1999

QA
76.9
.S88
E73
1999
B051144

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. David Davenport (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Uğur Doğrusöz

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Veysi İşler

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

ReMemex: A SOFTWARE TOOL FOR
KNOWLEDGE CONSTRUCTION and
ORGANIZATION

Özden Emek Erarslan
M.S. in Computer Engineering and Information Science
Supervisor: Asst. Prof. David Davenport
November, 1999

Citizens of the Knowledge Age have to be skilled in the construction of meaningful knowledge; in the collecting, analyzing, synthesizing, and organizing of information. These demands are driving educational technology research to find new theory and practices. Constructivism, an emerging theory of learning, offers new perspectives and is expected to have a major influence on education in the near future. Yet today, despite increasing efforts, considerable research is still needed to determine how best to apply constructivist ideas to educational software and to prove or disprove its effectiveness. This thesis presents the ReMemex system, a software tool, which both aids knowledge workers to collect and organize information, and provides a basis for research into the application of constructivist learning theory in educational software. ReMemex is a framework for a knowledge construction and organization environment, where learners create information landscapes to visualize the inter-relationships between the concepts and/or objects the domain is composed of. Features such as importing files and data from other applications as nodes, abstraction of the details by grouping, multiple view support for maps and nodes, and style support make ReMemex more powerful than traditional tools. The base implementation of ReMemex has been developed in Java as a flexible and extensible Object Oriented API.

Keywords: Constructivism, Knowledge Organization, Educational Software, Cognitive Tools, Semantic Networks

ÖZET

ReMemex: BİLGİ İNŞAASI ve DÜZENLEMESİ İÇİN BİR YAZILIM ARACI

Özden Emek Erarslan
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yard. Doç. Dr. David Davenport
Kasım, 1999

Bilgi Çağı insanları, bilginin inşaasında; toplamada, çözümlemede, sentezlemede ve düzenlemede vasıflı olmalıdır. Bu vasfa sahip olmaya yönelik talepler eğitimsel teknoloji araştırmalarını yeni teoriler ve uygulamalar bulmaya yönlendirmektedir. Gelişen bir öğrenme teorisi olarak Konstrüktivizm (constructivism), yeni bakış açıları önermekte ve bunun yakın gelecekte eğitim üzerinde büyük etkisi olması beklenmektedir. Bugün, artan çabalara rağmen, konstrüktivist fikirlerin eğitim yazılımlarına en iyi ne şekilde uygulanacağına karar vermek ve etkisinin olup olmadığını ispatlamak için yeterince araştırmaya ihtiyaç hala vardır. Bu tez, bilgi işçilerinin bilgiyi toplama ve düzenlemesine yardım eden ve konstrüktivist öğrenme teorisinin eğitimsel yazılımlara uygulanması konusunda araştırmalar için bir taban teşkil eden ReMemex sistemini, bir yazılım aracını sunmaktadır. ReMemex, öğrenen konumundaki kişilerin kavramlar ve/veya nesneler arasındaki ilişkileri görselleştirmek için bilgi peyzajı oluşturdukları bilgi inşaa ve düzenleme ortamları için bir çatıdır. Diğer uygulamalardan aktarılan kütük ve verilerin düğümler şeklinde gösterilmesi, detayların gruplama yoluyla soyutlanması, haritalar ve düğümler için çoklu-görünüş desteği ve biçem desteği gibi özelliklerin tamamı ReMemex sistemini geleneksel araçlardan daha güçlü kılmaktadır. ReMemex sisteminin temel gerçekleştirimi Java kullanılarak, esnek ve geliştirmeye açık, Nesneye Yönelik Uygulama Programı Arayüzü (API) olarak yapılmıştır.

Anahtar sözcükler: Konstrüktivizm, Bilgi Düzenleme, Eğitimsel Yazılım, Anlamsal Ağlar, Bilişsel Araçlar

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest thanks and gratitude to my advisor Asst. Prof. David Davenport for his patient supervision of this thesis.

I am grateful to Uğur Doğrusöz and Veysi İşler for reading the thesis and for their constructive comments.

My special thanks to İlker and Murat for their great support and help; and to Ali, Yeşim, and Zülal for their friendship.

Finally, I also would like to thank to my parents and my sister who are always behind me.

To my parents and my sister

Contents

1 Introduction **1**

2 Educational Background **4**

2.1 Objectivism 4

2.2 Cognitive Psychology 5

2.3 Constructivism 5

2.3.1 Learners are Active 7

2.3.2 Multiple Perspectives 7

2.3.3 Collaborative Learning 8

2.3.4 Authentic Context 9

2.4 Discussion 10

2.4.1 Application of Constructivism 11

2.4.2 Constructivism and Current Situation 16

2.4.3 Looking Ahead 20

3 Educational Software and ReMemex **21**

3.1 Educational Software 21

3.1.1 Intelligent Tutoring Systems 22

3.1.2 Web-based Instructional Tools 24

3.1.3 Web-based teaching and learning and the future 27

3.1.4 The World Wide Web and Constructivism 29

3.1.5 Cognitive Tools 30

3.1.6 Knowledge Construction and Representation Tools 32

3.1.7 Constructivism and Concept Maps 34

3.1.8 Applications of Concept Maps 34

3.1.9 The Way towards ReMemex 36

3.1.10 Overview of ReMemex and Related Applications 38

4 Design **41**

4.1 Requirements 41

4.2 Design Overview 47

4.2.1 Design Patterns 47

4.2.2 High-level Design 48

4.2.3 Detailed Design 50

5 Implementation **58**

5.1 Why Java? 58

5.2 Implementation Flavors 60

5.2.1 Actions and Commands 61

5.2.2 Icons and Shapes 68

5.2.3 Data Types 69

6 Conclusion **70**

List of Figures

3.1 A concept map of concept map 33

4.1 The layered architecture of ReMemex 49

4.2 Graph Structure Layer 51

4.3 MVC Base Layer 52

4.4 The Concrete Visualization Layer 56

4.5 Application Layer 57

5.1 A screen dump of ReMemex: Sample map about Design Patterns 62

5.2 The modified version of the sample map 63

5.3 Sample action: Select All 65

5.4 Sample action: Clear All 66

5.5 Method itemSelectionChanged 66

5.6 RemoveItemCommand class 67

List of Tables

2.1 Industrial Age vs. Knowledge Age learning practice 18

Chapter 1

Introduction

Today, there is a shared belief that we are living the shift from Industrial Age towards Information Age, or so called Knowledge Age. For example, in US, year 1991 is assumed to be the Year One of the Knowledge Age, when spending for Industrial Age capital goods, a total of $107 billion, was exceeded by the spending for information technology, which was about $122 billion [TH99]. The turning point is not so sharp, but it is obvious that some transformations are already on the way.

This implies that education, the main source of preparing people for life and work in the society, should also revise itself according to the new dynamics. Westera identifies three major factors that force today's education for innovations [Wes99]: (1) the convergence of classroom teaching and distance learning; (2) the effective technology-push for addressing new ways of collaborative learning; and (3) changing student-tutor relationships. Usually, utilization of the latest technology into education is considered as an innovation, however, the technology alone is not the reason that causes educational innovation; nor does it improve teaching and learning automatically. In other words technological potentials do not transfer into direct educational benefits easily. Sometimes new models of teaching and learning are essential to address the new directions and practises.

Constructivism, an alternative theory of learning, is the latest area that

scientists apply to describe the transformations. Compared to the traditional beliefs of teaching and learning such as objectivist pedagogy, constructivism offers more active learners who are creators of individually-meaningful knowledge, and who are responsible for their own learning. Advocates of constructivism assume that the internal mental models in the minds of individuals are likely to be different; and since learning is based on prior knowledge and experiences, that is, on mental models in the minds, it is an expected case that individuals form different understandings from the same knowledge domain. The learning environments based on constructivist views should be designed properly to *engage* learners in a collaborative learning process, rather than to *teach* them in a transmission mode of learning where the learning processes are totally directed by teachers, or computers. In this sense, computer programs for drill and practice as well as most intelligent tutoring systems represent traditional instructivist teaching models. Although constructivists value the Web as an open information source and a powerful communications medium, they do not classify the integrated Web based teaching and learning packages as constructivist tools. "Even an advanced application like the virtual classroom represents both socially and functionally, a traditional classroom" ([Wes99], p17).

The *cognitive tools* approach defines constructivist views of educational software. Basically, cognitive software tools are software applications that enhance the higher-order thinking and reasoning skills by providing powerful mechanisms to manipulate information. Generally, cognitive tools are assumed to be rather unintelligent tools relying on the learner to provide intelligence. They assist the learners in operations such as storing, retrieving, filtering, and visualizing information, while learners construct knowledge by conceptualizing, analyzing, applying, and evaluating information.

This thesis presents a framework and initial developments of a software system, ReMemex, which is based on the cognitive tools approach. ReMemex is a collaborative knowledge construction environment where users can visualize and organize information landscapes. The visualization is based on the concept mapping or cognitive mapping technique which has been proven as a powerful cognitive tool. Concept mapping is closely related with the semantic

network theory and schemata theory, theories that try to model the internal structures and processes of human minds. Simply, a cognitive map based on these theories is composed of nodes that represent domain concepts and links that represent the interrelationships between those concepts. ReMemex adds some more capabilities to this approach such as grouping related nodes into a single node, associating nodes with external data, and mechanisms to filtering and exporting the maps. In short, by creating a map of a particular information landscape learners are engaged in an active knowledge construction process while organizing the scattered pieces of information into a meaningful semantic structure that closely reflects the actual structure in their minds.

The oranization of the thesis is as follows. Chapter 2, first, introduces the constructivist learning theory with the main principles it is based on. Then, it presents some examples of the learning environments that apply constructivism and finally, dicuss the results and investigates why constructivism has not been implemented fully.

Chapter 3 presents the background research about the educational software applications and explores the characterictics of the main systems encountered in this field from a constructivist point of view. At the end of the chapter the ReMemex system is introduced and related applications are given. This chapter can also be viewed as the story of how the idea of ReMemex is devised.

In Chapter 4, after the specification of the requirements of the ReMemex system, we discuss some design strategies and criterias, and then present the high level and low level designs of the system.

Chapter 5 starts with the discussion of the choice of the language, Java, and then gives some implementation details of the main parts of ReMemex.

The thesis finishes by concluding the studies in the last chapter, Chapter6, where we also discuss the directions for possible future work.

Chapter 2

Educational Background

2.1 Objectivism

Today most of the instructional system design models take their groundings from the objectivist views of learning. Objectivism is based on the behavioral psychology, a field advocating that the focus of learning is in shaping learner's responses. It is usually described in terms of the "stimulus-response" principle [WS96]. Given a certain stimulus, it is expected that the learner is conditioned to respond in a certain way. The final goal is not that the learner will learn x or understand y. Instead the outcomes are all described in terms of behaviors, that is the learner will be able to do z. Little consideration is given to the individual learners, their internal thought processes, and the differences in prior knowledge and motivation that each brings to the instruction, assuming that all meet a preset list of entry behaviors [McM].

Under objectivist pedagogy the instructor, as a master, identifies the goals in terms of terminal behaviors, determines the minimum level of expected skills prior coming to instruction, creates the learning environment in terms of selecting media, developing strategies and production of instructional materials, and transfers his knowledge to learners. Although the student interacts with the learning environments she is thought of as being basically passive, and either

intrinsically or extrinsically motivated to learn the behavior set in the instructional goal. Assessing the student progress is relatively simple; can the learner exhibit the required behavior? Objectivist instructional systems are appropriate if the knowledge is procedural and can be exhibited, such as in teaching surgery, piloting etc. The aeroplane simulators that are used to train pilots are good examples of teaching and learning environments that incorporate behaviorist thinking. But if the instruction deals with declarative knowledge, or more importantly higher levels of thinking and learning processes (i.e. analysis, synthesis, problem solving, experimentation, creativity, and examination of topics from multiple perspectives) the objectivist model is ineffective [McM].

2.2 Cognitive Psychology

The difficulties of applying behaviorism to learning lead to the rise of cognitive psychology. The research on memory and mental imaginary indicated that psychological processes and prior knowledge intervene between the stimulus and the response, making the latter less predictable by behavioral theory [WS96]. The focus of cognitive psychology is the mental representation and mental processes that lie between stimulus and response. Basically, cognitivism implies that in order to understand learning we should understand how our mind manages the input, processing, storage and retrieval of information. With increasing efforts many teaching and learning theories are derived that have close relationships with cognitive psychology. Today, constructivist learning theory seems the most popular one appearing in much of the recent educational research, theory and policy [DC96].

2.3 Constructivism

There are many reasons that attract attention to constructivism. Greening [Gre98], for example, argues that the need meta-learning, lifelong learning and "just-in-time" learning requires emphasis to be placed on student-centered learning, which is a leap to a constructivist approach. Duchastel [Duc97], takes

the developments in Web technologies as a reason to devise new models of university instruction. Although he does not state any idea based on constructivism, the model he proposes has great similarities with constructivist learning models. Some research fellow [LAB+] focus on the use of the Web in distance learning, which they think necessitates a shift in pedagogy and teaching/learning styles towards constructivism. According to Westera [Wes99] the new powerful software collaboration tools enable effective collaborative learning, which in turn, starts a changing period in student-teacher relationships. He thinks that constructivism offers a close match with this change. After analyzing the needs of the new Knowledge Age, Trilling and Hood [TH99] find that fortunately the modern constructivism theory of teaching and learning fits those needs (and that current educational practice should be revised to match the theory).

Whatever the reasons are, most educational scientists agree that constructivist learning theory will have a major influence on the future of education. We, therefore, have tried to follow the constructivist principles in our work. Constructivism is concerned with how we construct knowledge, while the emphasis in objectivism is on the `object` of our knowing [CJLM98]. Constructivism emphasizes the mind's process of "meaning-making" from external input; objectivism emphasizes the content of that external input and assumes that it is "placed" in the learner's mind as it was presented. Constructivists claim that new knowledge is built upon prior knowledge based on past experiences. Therefore, it is expected that individuals may come up with different achievements or different views of the same knowledge domain. In constructivist learning environments (CLE), learning is considered as an active process of constructing rather than acquiring knowledge, and instruction is a process of supporting that construction rather than communicating knowledge [DC96]. The common grounding of constructivism could be summarized by von Glasersfeld's statement: "Instead of presupposing knowledge is a representation of what exists, knowledge is mapping in the light of human experience of what is feasible".

Given this brief overview the following sections introduce (summarize) main characteristics of CLEs.

2.3.1 Learners are Active

Student ownership of the learning tasks must surely be regarded as a basic principle of constructivist pedagogy [Gre98]. In CLEs knowledge does not exist "out there" in an objective reality. Each individual learner experiences the same knowledge (problem) domain and seeks explanations that are usable in the world as he/she understands it, rather than seeking the "truth" by correspondence of the real world [DC96]. This kind of activities can be achieved by defining ill-structured or ill-defined problems within criss-crossed landscapes with its suggestion of a nonlinear and multidimensional traversal of a complex subject matter [McM]. Learners devise their own problems, define goals and examine the domain from this point of view and construct own solutions and views. This means that they also learn how to learn, and probably, how to teach.

After reading the paragraph above people might think that educators in CLEs are lazy, or unimportant. However, teachers do have critical roles in CLEs. First of all they should provide interesting, relevant and engaging problems to solve, and open learning environments to explore. They should guide and help the students rather than directly telling them what to do. They may also participate in the learning process as "senior partners". To quote Cunningham and Duffy: "We no longer teach, but rather coach - we have moved from the sage on the stage to the guide on the side"([DC96], p184). In CLEs perhaps the hardest task for teachers is the evaluation of students. It is clear that CLEs require alternative approaches to traditional testing procedures (i.e. to tests and exams), which are not straightforward to develop.

2.3.2 Multiple Perspectives

In CLEs multiple perspectives are valued and necessary [CJLM98]. Instead of encouraging acceptance and closure of a specific idea people are asked to present and debate their own ideas. We might be confused and fail in finding reasonable explanations that fit our past experiences and memory. When failure happens, we are ready to hear about other people's explanations, and

enhance our viewpoint. By this way, we are aware of multiple perspectives and continually expand our web of understanding. Thus, confusion followed by explanation leads to learning [Sch98]. This again, is one of the areas that the traditional education does not apply, a possible reason why it does not work very well. No confusion is allowed (or assumed) as the learners are taught the objective reality directly.

Learning is best performed when participants gather multiple views and synthesize them into an integrated one. In CLEs, people are also encouraged to utilize knowledge construction and modeling tools, collectively called cognitive tools, to represent knowledge in different forms and to provide examples in several kinds of media [Gra96]. As they are closely related to our project work, cognitive tools are discussed in detail in Section 3.1.5.

2.3.3 Collaborative Learning

Sharing multiple perspectives implies collaboration. In fact, learning itself, is an inherently social process; it most naturally occurs not in isolation but in teams of people working together to solve problems [DC96] [Jon98]. One may claim that constructivism is not consistent and contains the paradox of encouraging both student-centered (individualized) learning and group learning. However, it should be recognized that learning remains a strictly individual process, actually located in the brain of the person involved. From this point of view, the primary focus of cooperative learning is to optimize the conditions for each members' learning process, that is, the conditions for individualized learning [Wes99]. In other words, knowledge is individually constructed and socially co-constructed.

Collaboration and sharing enable learners to make clear knowledge/ideas which may be internally fuzzy [CJLM98]. Additionally, students are more willing to take on the extra risk required to tackle complex, ill-structured problems when they have the support of others in the cooperative group [Gra96]. In collaborative environments they also learn solving problems collectively, playing multiple roles, and confronting ineffective strategies and misconceptions. Students working together are responsible for each other's learning as well as their

own. They are expected to support one another and to provide scaffolding to those who are unfamiliar with the domain. In this way they also gain teaching skills. Research (given in Section 2.4.1) has shown that collaborative learning is likely to be more successful than traditional methods when implemented properly.

2.3.4 Authentic Context

Traditionally, content was thought of as being decomposable into components. It was considered as being separate from the individual and therefore manipulatable independently of the individual [DC96]. This is the objectivist view of content. However, today most educationalists believe that people can not transfer knowledge from one context to another easily [Gra96]. Knowledge that is taken out of context during instruction does not have much meaning to the learner. Learners are assumed to know something if they act effectively in the relevant context. Thus, they need both content and context learning. In constructivist views context is more dynamic including the individual and socio historical context; and plays a very significant part in learning. The demand for more "authentic" learning tasks that match real-world conditions comes directly from these findings [TH99].

Authentic means that learners should engage in activities which present the same type of cognitive challenges as those in the real world [Jon98]. Or simply, it can mean personally relevant or interesting to learner. In CLEs, problems must be authentic because it is difficult to create artificial ones that maintain the complexity and dimensions of actual problems. An authentic problem, activity or goal provides learning experiences as realistic as possible. Realistic problems hold more relevance to students' needs and experiences as they narrow the gap between the artificial world of schools and real-life society. This helps in building stronger and richer internal connections, and learners are more successful when they re-apply (transfer) what they have learned to novel real-world situations [CJLM98].

2.4 Discussion

Reasons for applying constructivist approaches in education follow from the characteristics described above. It appears to describe the natural process of learning, that is, we believe that something like constructivism occurs during learning (though its truth or falsity is not provable) [CJLM98]. It accepts the diversity of thinking, understanding and building mental models among individuals, which is the case in real world. So, it makes little sense to develop learning and assessment activities that contradict the knowledge construction that is "naturally" occurring.

In CLEs we may expect that learners are highly motivated. They participate in creating learning goals, they devise their own problems from ill-structured problems and follow the paths of experience and learning they think best for them. Briefly, what they do is inherently more interesting, a factor that increases motivation. Also, the authenticity in CLEs offer learners more realistic experiences and learners are less likely to loose their interest as they closely mimic what they see in their everyday life.

One of the major skills that learners gain with constructivism is higher order thinking and reasoning, the areas in which today's students are not particularly strong. "In conventional kinds of schooling learners tend to organize their mental activities around topics rather than goals. Usually, they focus on surface features and do not examine a topic in depth. They work straight ahead, that is they tend to work until a task is finished. They do not take time to examine the quality of their work or thinking. Finally, they think of learning in additive fashion rather than transforming and enriching their existing knowledge structures"([Gra96] p671). "Students treat new information as facts to be memorized and recited back rather than as tools to solve problems relevant to their own needs."([Gra96] p666).

These behaviors prevent them from transferring their knowledge to new problems. They may even fail when they face problems that are similar compared to the ones they encountered before (but which require different viewpoints). On the other hand, learners of CLEs do not treat knowledge "as an

end". They develop strong connections with prior knowledge, identify and eliminate unclear points and examine the problem from multiple perspectives. Thus, they also learn how to approach and analyze problems. Obviously they should be more successful in adopting and solving new problems.

2.4.1 Application of Constructivism

The benefits discussed above are all valid from the theoretical point of view. Whether constructivism really works is a question that needs to be discussed in detail, but here we give an overview of some research findings.

Virtual learning environments (VLE) are frequently applied to increase the authenticity in education. Most research on VLEs is based on creating virtual 3D worlds and simulations using virtual reality (VR). Especially in science education interactive simulations are potentially very useful and relatively easy to build as the behavior of objects and their interactions can be well defined. Besides simulating the real world, VLEs also offer interactions that are not otherwise possible to experience in real world [MCKE98].

Dede and his colleagues [DSL96] state that VLEs has the potential to complement existing approaches to science instruction through creating immersive inquiry environments for learners' knowledge construction. In their work they set up and evaluated three virtual worlds, namely NewtonWorld, MaxwellWorld, and PaulingWorld in which students could explore the kinematics and dynamics of motion, electrostatic forces, and the structure of small and large molecules in a number of single or mixed representations. In NewtonWorld, for example, students configure the balls and examine their movements and collisions within a special corridor from various viewpoints such as from a camera attached to the center-of-mass of the balls. Formative evaluation studies of these virtual worlds has been conducted with respect to their usability and learnability. These studies show that learners enjoyed their learning and found the activities interesting, engaging and more effective than either textbooks or lectures. Limitations and discomfort caused by the current VR head-mounted displays hindered usability and learning. On the other hand, manipulating the

field in 3D, multisensory cues, and the introduction of multiple new representations is believed to have helped students develop correct mental models of the abstract material.

Researchers at the Multimedia and VR Lab at Ioannina University found similar results [MCKE98] in evaluating their VLE designed to investigate the phenomenon of eutrophication in lakes, a topic in environmental education. They also emphasize the positive acceptance of VR in the educational process. Importantly, half of the students declared immersion experiences although the study does not support immersion. And finally they also observed the two common results; that VLEs offer good motivational support, and that there are some usability problems in VLEs.

The NICE (Narrative-based, Immersive, Constructivist/Collaborative Environments) project [RJM+99] focuses on the use of the VLEs in children education and claimed to be the first immersive, $multi-user$ learning environment. NICE implements a persistent virtual garden in which children may collaboratively plant and harvest fruits and vegetables, cull weeds, and position light and water sources to differentially affect the growth rate of plants. There are two constructivist artifacts in NICE which are the garden and the stories formed by students using a shared story writing workspace. In addition to planting and story writing, as a constructivist approach, students also gradually discover the relationships between the characteristics of plants and the amount of water and sunlight they need. There is also a 2D applet version of the garden that is accessible via the Web and is capable of working simultaneously with the VR system. Lastly, all these activities are done collaboratively including a text based chat facility for communication. In the evaluation phase it was observed that most kids felt immersed attempting to *touch* the virtual objects by moving and clasping their hands in the air. Their attitude towards the workspace was highly positive. Most (73%) of the children answered "nothing" to the question "what did you dislike the most?". The intend during these studies was to have only one child in each group control the wand (the VR system). However, this greatly affected to the learning process. Approximately 17 children (35%) were able to understand the NICE models and improved their gardening knowledge as desired. Thirteen of them took control of the system.

As a result, immersive VR permits concepts and symbols to be separated. Constructivists note that this has consequences for the positivist information processing view of cognition, which emphasizes the manipulation of symbols, and is therefore unable to properly address the first-person direct experience (which often has an association with tacit knowledge) [Gre98]. Constructivism recognizes this tacit knowledge as the result of reconstructions in response to direct experience, that is incapable of being demonstrably shared.

Besides VLEs, there are other environments that can be classified as virtual, but that do not rely on VR. In the Learning Research and Development Center at University of Pittsburgh researchers set up a problem based environment consisting of authentic scientific problems, Web based resources and a tool, called Belvedere, that provides visual and collaborative construction of hypotheses by utilizing inquiry diagrams [STW97]. The chosen problems such as finding the cause of a strange disease in a Pacific island allow development of multiple possible hypotheses. The Web based curriculum materials include background information, simplified versions of articles on scientists' hypotheses, methodology and field reports and a link to experiments involving both hands-on manipulatives and computer simulations. Students are encouraged to use five phases of inquiry and provided suggestions on how to conduct scientific inquiry and how to use the Belvedere software in this process. Evaluation results show that students appeared to be engaged and on task. They report that working with Belvedere makes it easier for them to organize and review the arguments for and against a specific scientific hypotheses. Interestingly, the evaluators observed that the classroom changed from a traditional format, with students doing work at their desks in rows, to a group-centered organization, in which students were gathered around computers or hands-on activities "like campfires" and engaged in active discussing one of the few examples against traditional beliefs of education. Some other knowledge organization and construction tools similar to Belvedere are examined in Section 3.1.6.

The Epistemology and Learning Group at MIT leads many innovative projects, especially focusing on children education. Their work is based on *constructionism*, a theory of learning developed by Seymour Papert. It is an extension of constructivism, accepts the constructivist principles and adds the idea that people

construct new knowledge with particular effectiveness when they are engaged in constructing personally-meaningful artifacts such as LEGO machines, animations or computer programs [Res98]. Therefore, in all of the learning environments they build children are required to create such a product. For example, in the PetPark, a graphical virtual world, kids can teach their pets to dance, greet visitors, or even tell jokes using a special kid-friendly programming language called YoYo. Another project, Programmable Bricks, enables children to program their LEGO constructions. A child writes a Logo program on a personal computer, then downloads it to the Programmable Brick called Cricket, after which the child can take or put it anywhere. Each Cricket has output ports for controlling motors and lights, and input ports for receiving information from light, touch, and temperature sensors. They can also communicate with one another. Now commercial versions of Programmable Bricks are available from the LEGO company. Children use them to create autonomous robots, active rooms etc. The researchers also try to explore children's learning from the digital versions of the traditional toys such as beads, balls and blocks that have added programming capabilities. Crickets are also used in another research initiative called Beyond Black Boxes that aims to develop computational tools and project materials that allow children to create their own scientific instruments. For example, 11 year old Jenny, built a new bird feeder capable of taking photographs of birds, a facility which her old bird feeder could not do [Res98]. She used various tools such as a touch sensor, a Cricket, a camera, bricks and did some programming. These are all good examples demonstrating authentic contexts. By involving in such activities students not only become more motivated, but also develop deeper understandings due to the strong connections between these activities and their real-world experiences.

There are many instructional applications commonly characterized as constructivist or, at least, they reflect some characteristics of constructivism, if designed correctly. Some examples are problem based learning [DC96], project based learning [LTMW98] [WT], learning through goal based scenarios [Sch98], and reciprocal teaching [Gra96]. In all these models collaboration, teacher guidance, and students' responsibility for learning activities are required.

The Project Based Learning Support System (PBLSS) [LTMW98], for example, includes support for 2 instructional processes; (a) scaffolding and (b) coaching; and 4 learning processes; (a) planning and resourcefulness, (b) knowledge representation, (c) communication and collaboration and (d) reflection. In PBLSS, the students together with the teacher decide on goals, break complex tasks down into achievable objectives, develop plans for these objectives by allocating time periods of work, and anticipate and plan for the resources that must be available for an objective to be reached. They can also organize their thoughts into an analogue of a journal article that includes sections of abstract, goals, specific objectives, project team, responsibilities and an applications/extensions section in which they may draw conclusions from their work and make suggestions for further inquiry. All of the activities listed above are done collaboratively and supported by the specialized tools of PBLSS. The evaluation of the system produced some disappointments. Although most of the students (70%) felt the tool was of average-use to very-useful and said that they (76.7%) would like to use it again in future projects, basically they did not seem utilize it correctly. Most viewed the tool as a representational vehicle for their projects' work, but they did not take it as a vehicle for furthering that work or as a tool to support them in the process of doing that work. We discuss the reasons for such attitudes in the following section. The difficulty appears to originate from the students' beliefs about traditional education.

For communicating and debating asynchronously over the WWW, usually a threaded discussion paradigm is used. However, some researchers argue that in the light of constructivism the design of such tools themselves is an important factor in determining the effectiveness of collaboration [SDD99] [KS96]. For example, the same design is seen as appropriate for small group problem solving and general newsgroup like discussion. Most designs make no distinction between the use of the tools in business mode or in an educational context. Klemm and Snell clearly state their ideas: "We think that educators who use computers in education are missing an important opportunity by their slavish acceptance of the threaded discussion paradigm" [KS96]. In their work, they developed a hypertext based asynchronous conferencing environment, FORUM, where messages are not bound to the traditional rigid hierarchy, rather they can be linked via various relationships. It also allows

teachers to create logic structures that mediate students' effort to produce the required deliverables. Although there is not any empirical data, their work is important as being a different approach for threaded discussions.

Yet another approach has been developed at Indiana University [SDD99]. This tool (ACT) makes use of textual labels and colors attached to the posts to indicate how it is contributing to the subject matter. All threaded message headers are displayed, that means the top level message is not treated as a folder to be opened. Basically, the tool supports two educational processes; exploration and analysis. For exploratory activities, ACT provides linear conversation spaces, similar to traditional discussion boards, with the exception of summary labels. Messages are not categorized in any way but rather just sorted by date. Exploration space is meant for free-wheeling and divergent thinking, but an important activity in such environments is pausing and summarizing the discussion, helping to focus attention on emerging key issues. To support analysis space, ACT requires each message to be colored and labeled. The color coded labels serve two educational purposes. First, they require users to pause and think about the nature of their post and how it will add to the ongoing analysis. Second, they allow users to quickly get an overview of the general state of a discussion. The choice of the labels before starting the discussion is one of the most important tasks. The teacher can do this or it can be another subject of pre-(explorative) discussion. ACT has been evaluated in two courses, one undergraduate and one graduate. The instructors indicated that they were pleased with the level and quality of participation, and also reported that ACT successfully helped them meet their instructional goals, and that students displayed better critical thinking skills than in past semesters.

2.4.2 Constructivism and Current Situation

Given these success stories can we conclude that we are ready to shift to constructivism? The answer is clearly "no". The implementations listed above, give some positive indications, but even experienced researchers can not confidently state that the impacts are great, or even highly promising. In fact, it

would be true to say that most of the implementations were not fully constructivist.

The reasons why constructivism has not been implemented fully originate from the reality of the traditional objectivist educational system. Learners, especially older students in high schools and colleges have learned to "play the game" with years of practice. The rule of the game is simple: "A students primary job in school is to find out what his/her teacher thinks is valuable and to achieve to that standard"([LTMW98] p85). Most of them are aware of this need and are successful (success being defined by grades.) Why should they wish to learn in unfamiliar ways in which they may fail? This fear, called "grade-phobia" [CJLM98], is one of the major motivations in current education. For instance, the developers of PBLSS report that many students were not driven to use the tool enough for it to become a major part of their project work, since PBLSS was not adequately aligned with assessment (the representations made in PBLSS were not graded) [LTMW98].

Students seem also to have problems in collaborative activities. Within the collaborative learning model's almost heavenly cooperation, there is no room for competition, selfishness, or envy. Unlimited helpfulness, however, may easily interfere with individual objectives and ambitions [Wes99]. In the NICE Project, for example, evaluators observed that despite the warnings of the teacher students regarded friends in the same group as their competitors. The evaluators argue that competition contributed to the excitement of the children in the group, but kept them off-task and distracted them for nearly the entirety of the experience [RJM+99].

Another false behavior may be taking over individual tasks, while not taking into account of the educational needs of the person involved. One should assist but not take over [Wes99]. Assuming that no such problems exist, still there is a trouble. It is difficult in the classroom, and impossible outside of the classroom, to monitor and mentor collaborative discussion and critical thinking when it occurs in small group meetings. Without this ability only the final product can be reviewed [SDD99].

Industrial Age	**Knowledge Age**
Teacher-as-Director	Teacher-as-Facilitator, Guide, Consultant
Teacher-as-Knowledge source	Teacher-as-Co-Learner
Curriculum-directed Learning	Student-directed Learning
Time-slotted, Rigidly Scheduled Learning	Open, Flexible, On-demand Learning
Primarily Fact-based	Primarily Project-&Problem-based
Theoretical, Abstract Principles&Surveys	Real-world, Concrete Actions&Reflections
Drill&Practice	Inquiry&Design
Rules&Procedures	Discovery&Invention
Competitive	Collaborative
Classroom-focused	Community-focused
Prescribed Results	Open-ended Results
Comfort to Norm	Creative Diversity
Computers-as-Subject of Study	Computers-as-Tool for all Learning
Static Media Presentations	Dynamic Multimedia Interactions
Classroom-bounded Communication	WorldWide unbounded Communication
Test-assessed by Norms	Performance-assessed by Experts, Mentors, Peers, and Self

Table 2.1: Industrial Age vs. Knowledge Age learning practice

Not all learners are likely to be comfortable with the increased level of ambiguity in problems. They tend to say "Why don't you just tell me?" when faced with situations that require critical thinking [CJLM98]. We should not expect learners to adopt a new style of learning all at once. Table 2.1, originally created by Trilling and Hood [TH99], lists Industrial Age (Objectivist) vs Knowledge Age (Constructivist) learning practices. One of the reasons why adoption is not straight forward can be seen from the table: "Many of the behaviors beneficial for Industrial Age learning become their near opposites in the Knowledge Age" [TH99].

Learners are not the only group that are unprepared for constructivism. For the teachers who themselves experienced the transmission model of learning and were trained in the colleges of education "to teach in the objectivist way", it does not seem easy to switch to a significantly different model of teaching. Without support it is almost impossible for an overworked teacher, unfamiliar with the constructivist paradigm, to create constructivist learning experiences or to help prepare learners to be open to and successful in these experience. They are expected to coach not only in the content area, but also on the new and probably unfamiliar learning techniques themselves [CJLM98]. They must adjust curriculum so that constructivist activities, which may take longer, can be balanced with "covering the material" required at certain grade levels. They should also develop authentic assessment techniques that vary according to the context and learning activities. This task list may grow, but these are enough to conclude that basic changes in teacher education are necessary [CJLM98] [TH99] [Gra96].

Finally, there is a third group who have doubts and fears about constructivism: educational institutes and parents. Just as teachers were educated in an objectivist tradition, so were parents, administrators, and evaluators. They have many ways to express their disapproval that can undermine and sabotage the success of constructivist learning environments. For example, some of the alternative assessment procedures are likely to be difficult for school boards and universities to accept. In fact, it is interesting that many colleges and educators now accept constructivist learning theory but they do not teach constructivistically [CJLM98].

As if opposition from these three groups were not enough, modern culture particularly in the workplace does not promote constructivism. We conclude this section with the following observations from Greening: "Mindless tasks (in the sense they do not encourage or rely upon reflection or evaluation) are commonplace in the real world. Goals are externally determined, tasks are owned by others, rules are adhered to as a means of achieving objectives; such inflexible environments may form the basis of training in which the demands of constructivist learning are at odds"([Gre98] p32).

2.4.3 Looking Ahead

In general, most aspects of the system are afraid and suspicious of change [CJLM98]. Partial adoptions do not seem to be highly successful often resulting in simple variation of existing objectivist methods. But is it fair to expect radical implementation of constructivism in today's conditions? Perhaps constructivism, in its ideal mode, will never be achieved. Here, we should also consider that compared to the pace of change in competitive business environments, education might be the last place for speedy action. Trilling and Hood discuss three main approaches; top-down, bottom-up and systemic-mixed mode to shift to new teaching and learning models [TH99]. Among them the systemic-mixed mode is more difficult to apply but promises more success. In fact, it can be seen as a combination of the other two within a systemic reform strategy. The top-down approach focuses mainly on developing standards, frameworks, and mandated structural changes by national or local authorities. In contrast, change in the bottom-up approach is forced by creative teachers or even students or by whole school experiments. And, in the systemic mode there is a top-down initiated leadership and support for the development and coordination of bottom-up initiatives. As a result, in the developed countries, first researchers then educational institutions and governments have recognized that some changes, simple or radical, are needed for the new Knowledge Age. And, constructivism seems the desired target of this change. We will see what will happen.

Chapter 3

Educational Software and ReMemex

In the research arena, perhaps the most important task that needs detailed work is the definition of the subject matter, that is the aim of the research. The case was the same with our work. We have made considerable effort to devise a solid topic to work on. Besides the background about the educational science presented in Chapter 2, we also have investigated the educational software side, that is the research on educational software, the most used software applications in teaching and learning, and the latest trends in this field. This chapter presents these investigations and some discussions that have formed the grounding of the main ideas of this thesis work.

3.1 Educational Software

The history of educational software can be divided into three sections in terms of their main characteristics: the single user drill and practice systems, the intelligent tutoring systems (ITSs), and finally large, open networked, multiuser integrated teaching and learning systems. In the early days of computer based technology, from the educational point of view the most interesting applications to mention were some educational programming environments for

children [RJ96] [RKSK98]. Among them, Logo, Dynabook, and Smalltalk are well-known examples. The aim was to provide students a simple programming language that enables them to construct software and new tools to express ideas. In the late 1970s Dynabook, for example, was designed specifically to match the learners' (children's) needs: portability, high-resolution graphic displays, user-friendly interfaces, and low-cost, including support for drawing, word-processing, a child-centered programming language. The Dynabook led to the Xerox Alto and Star systems, and later to the Macintosh, and modern graphical user interfaces. Similarly, Smalltalk, the first extensible object-oriented programming language, has been adopted as a professional programming environment for industrial use. Logo, on the other hand, has not changed its line, and still it is being used in some educational settings, such as the Lego-Mindstorms given in section 2.4.1.

Interestingly, the early environments seem to be designed with constructivist principles in mind. The common belief was that children should control computers, not be controlled by them. People have theorized that learning to program is an activity that develops higher-order thinking skills [RJ96]. However, later with the advent of computer technology, child-centered programming environments lost their importance against ITSs, and then against highly interactive multimedia applications. Today, non-technical students routinely compose ideas on their desktop computers, but few use a programming language.

3.1.1 Intelligent Tutoring Systems

During 1980s, the second line of educational software applications, the ITSs, appeared as a result of the developments in artificial intelligence(AI) research. The traditional ITS approach uses AI techniques to formulate a model of the student's knowledge, and a model of expert knowledge, and then intervenes with tutorial advice, when differences become evident [RKSK98]. Constructivists claim that ITS do not have any place in constructivist learning frameworks. Jonassen and Reeves [RJ96], for example, state that even the advocates of the ITS field began to acknowledge the lack of the impact they have had

on main stream education and training. They mention two main reasons for this failure: students are treated as perceivers and recipients of educational communications, and technical difficulties inherent in building student models and facilitating human-like communications. Similarly, Winn and Snyder state that the student model should use the cognitive approach to learner analysis, that is computational descriptions of students' mental model, not their levels of performance prior to instruction [WS96]. As learner-centered education is one of the main concepts in constructivist views, the negative ideas about the ITS approach should taken as a natural reflex against an environment where learners, as passive receivers, are totally directed by computers and where there is no or very restrictive collaboration and where high level thinking skills are not promoted. The concept, learner control, is used to refer to the degree of control of learning activities given to the teacher, the computer or the students themselves. Constructivist authors argue that this term is rather related to objectivist views. In this sense, the ITS approach, as an environment that take learner control in hand, is definitely "instructivist" [RJ96], and can not be a building block of CLEs: "However, in teacher control, its primarily a control of the content and the basic learner task. In computer-controlled literature, the control is far more pervasive in that the computer takes over even the minute decision making." ([DC96], p186).

On the other hand, the ITS field is evolving and trying to incorporate emerging ideas and technology [SP96]. For example, already web-based ITS environments [Hwa98] and integrated online teaching and learning tools that use some degree of AI [Ozh97] have appeared. Some specific research questions compiled by Shute and Potzka include: "(a) how can computers better understand natural language? (b) what kind of inference mechanisms can optimally model students' knowledge status? (c) how can computers be programmed to understand semi-logical reasoning (including intuitions, pet theories, prior experiences)?" ([SP96], p594).

The third milestone in the history of educational software was the utilization of computer networks in teaching and learning. The real explosion appeared when Web technology has gained the necessary facilities/characteristics, such as easy access, low-cost, dynamic content, some multimedia capabilities and

interactivity. The educational role of the Web can be considered from two main viewpoints, which are "Web as a network" and "Web as an information source". The former is related to the delivery of instructional material over the network, and the Web offers great advantages for this process: universal access, rich content presented via platform independent user interface, different modes of communications, easy to reach and use with well-integrated modes of navigation and cross-referencing. Meanwhile, the Web is growing exponentially; millions of servers and billions of documents are available forming a huge information repository, where searching, accessing, and retrieving information is fast and not constrained by space and time. Today, integrated Web based teaching and learning tools, or Web-based instructional tools (WBI tools) have a dominant impact in the field of the educational software research and development. Therefore, during the educational background work, we also tried to explore the WBI tools, latest trends, and research areas about them. The results are given briefly in the next section.

3.1.2 Web-based Instructional Tools

A classical integrated WBI tool provides necessary client and server software to setup and manage online education and training centers, to create and deliver course material and finally to register and take online courses. The first examples of such tools are developed in universities during early and mid 90s. WebCT, CyberProf, Serf, and Virtual-U are some of the tools that were originally developed at the universities, and then became commercial products as a result of the explosive demand for such tools after mid 90s. Today, very often new products are coming into the market and established ones are regularly being updated. Generally, the feature descriptions of WBI tools fall into four categories: authoring tools, collaboration, student tools, course management and administration.

Authoring Features

Putting a course online is more than simply converting existing course materials to Web pages. Every WBI package offers some help in planning and design of course material. For instance, templates are one the common options. While some instructors might find a template too restrictive, templates do serve the purpose of streamlining a good chunk of the process of designing a Web course, and provide a common look and feel to courses. Most of them are simply starting points, and can be modified and augmented relatively easily. Some of the products such as WebCT automatically creates glossary, table of contents, index and search engine for the course. More advanced ones provide graphical outlining tools, which enable to create, visualize, and modify "learning paths" easily. Defining different learning paths for different learners is one the latest research fields [Hwa98] [Ozh97], which aims to present custom tailored courses according to predefined characteristics of individuals or according to the online test scores.

The ideal case is first deciding on unique, smallest learning goals and providing a mechanism that can pack all the necessary materials of a learning goal into learning modules, or into learning objects. Then, in a hierarchical way combination of learning modules compose a topic, which is another learning module, and topics compose a lesson, and finally lessons compose a course. TopClass, for example, uses this approach to export a course or any element of a course to a plug-and-play file to use it somewhere else. However, to exchange courseware between different systems standards are needed.

Collaboration Features

Basically, Web offers two kinds of communication, synchronous and asynchronous, in other words, real-time and delayed communication. E-mail, mailing lists, threaded discussions, bulletin boards are examples of delayed communication, while chat rooms, whiteboards, audio-video conferencing, application sharing, and collaborative surfing provide real-time communication and collaboration. Usually, each course has its own communication channels. Today,

asynchronous tools are preferred because of the usability problems encountered with synchronous ones. Collaboration options also include file sharing between participants. It is useful to include a private storage space, where students can organize any files they intend to share, plus a separate file space for public access. Lastly, most of the WBI packages enable instructors to form and reform collaborative groups as the course unfolds. Group discussion areas provide private space for the group to organize and track collaborations, and group Web pages provide a place for group members to display the unfolding results of their work together.

Student Tools

Although the features given in this section are not supported by every WBI product, some of the them are really useful and should be considered essential for such a system. For example, LearningSpace enables students to annotate the resources they created collaboratively. The student also has control over who has access to his notes, either keeping them private or sharing them with the team, the class or the instructor. Some of the tools keep the history of the student's path through the courseware and next time the student can continue where he left off in his work. Systems, such as WebCT, provide a bookmarking system for this purpose. As mentioned within authoring features, learner and/or teacher generated glossary, course index, and a search engine for the entire course Web site are other basic elements that can be provided for use. Also, most packages support inserting review material and self-scoring exercises that help students deepen their learning. Finally, students can create or import their homepages to get organized and share information with other students.

Course Management and Administration Features

The primary features in this category include student management options such as student course registration, attendance, and participation tracking, a grade book, assessment tools, a place for students to post their assignments for instructor's feedback. Some packages provide analysis and report generation

tools that present the student tracking process to teachers from different point of views. Students can also follow their performance and compare with class summary data. Tools that support advanced student monitoring enable instructors to see when and for how long a student viewed a certain page or area and even from which IP address the student accessed the information. In most WBI systems course management options also include easy ways to replicate, backup, and archive courses. Usually, all other administrative work is assumed to be managed by the administrators, not by the instructors. However, systems that support administration over the Web enable instructors to use some of the administrative facilities such as creating a course provided that they are given appropriate privileges.

3.1.3 Web-based teaching and learning and the future

All authorities agree that the demand for Web-based education will continue to increase. Everhart for example clearly states this idea in an interview:"In the relatively near future, there will be a greater mix of virtual and residential learning. ... Considerable evidence indicates that the growing number of adult learners will double the number of learning hours needed by the total student population. Delivering instruction in a traditional sense to that entire population would require doubling the number of campuses, classrooms and professors. That is impossible; no economic plan could fund it. ... I do not foresee universities doubling campuses and professors. Instruction has to be leveraged, and virtual delivery is a great way to achieve this leverage" [Mor99a]. This increasing need and evolving hardware will lead the development of more robust, large-scale integrated WBI management systems with more opportunities such as intelligent virtual reality environments, video conferencing, virtual teachers, etc. "Virtual classrooms will change higher education in ways that we can not even imagine now" says J. Boettcher in another interview about the future of education [Mor99b]. But she does have a vision: "In 20 years, we will have seen multiple generations of software applications and agents come and go. Computer and software will be so integrated into educational resources that they will be invisible to the user of those resources. Many of us will have personalized software agents (PSAs) or maybe robots (PSRs) that will provide us

with daily digest of information according to our customizable, programmable profiles. ... They will update these assistants verbally or "feed" these assistants with data about their study areas and responsibilities; the more advanced models will support students in their problem-solving and thinking" [Mor99b].

Currently there is a big effort to turn another vision into reality: interoperable online teaching and learning systems. Once standards are established we may at least expect the reuse of educational resources at micro and macro levels. Moreover, there are other possibilities. Consider an online student, who takes courses from different education institutes, which are approved by the national or international online education committee as being an official online education provider that applies both interoperability and quality standards. Before enrolling to a course, the student searches for a suitable one, using online course search tools that enable users to select or enter course properties they are looking for. Since each course provides its description as meta-information in a standard manner, the student can select the exact course of appropriate content and length within the search results. Directly by clicking the apply button, he is prompted with an online application form where he enters his name and password, which are used to access his educational profile that is kept by the national (or international) Online Education Center. The system checks his transcript and if prerequisites match, he is registered. Upon completing the course his profile is updated automatically by the system. Of course, there are many leak points in this scenario, such as authentication of the users and other security problems. But, it is a good example that make people think of the opportunities that can come to life with the help of the standards. Today, there is a shared expectation that initiatives will seek out ways to collaborate with each other to form these standards [Ric98].

Already, there are some specifications such as the meta-data specification for learning material description from the instructional management systems (IMS) project by EDUCAUSE, and European ARIADNE project. Also, the Aviation Industry CBT Commitee (AICC), an open forum of training professionals that develops guidelines for inter-operable learning technology, has established the most comprehensive Computer Managed Instruction (CMI) specification. Some other initiatives working on learning technology standards

include the Advanced Distributed Learning (ADL) initiative, the Computer Education Management Association (CEdMA), the World Wide Web Consortium (W3C), and the Meta-data for Multimedia Information (MMI) Workshop. Many of the industry leaders and universities support the standardization efforts. For example, currently the IMS project has more than 200 developer members and 35 investment members from academic, coorporate, non-profit and government organizations, including IBM, Sun, Microsoft, ORACLE, Apple, and US Department of Defense.

Although frequently misunderstood, a specification is not a standard. In this sense, the IEEE Learning Technology Standards Committee (IEEE LTSC) is qualitatively different from other initiatives, because it is an accredited standards body. Consortia such as IMS, and AICC increasingly acknowledge the IEEE LTSC as the single forum for turning specifications into standards. Already, they have submitted their specifications to IEEE LTSC for standardization. As a result, it seems that the first versions of standards about learning technologies are on the way, but the real utilization of them requires some more time.

3.1.4 The World Wide Web and Constructivism

Constructivist authors agree that the Web is a potentially powerful tool for education, when utilized correctly. Reeves claims that Web may prove to be a more powerful vehicle for constructivist pedagogy than it is for direct instruction [Ree98]. He views the Web itself as an cognitive tool for investigating and representing knowledge. Since hypertext links work by association rather than indexing, this nature of the WWW can be used for the meaningful linking of data, which leads to the notion of a semantic knowledge space that will mirror learners' own developing cognitive structures. Duchastel thinks that the richness and relative chaos of the Web in terms of information and learning resources fits well with the knowledge production process of students [Duc97]. Also, all constructivists value the opportunities Web offers for collaborative work. Finally, Reeves points out that sharing knowledge and creation with

anyone anywhere in the world may be highly motivating because of the powerful "sense of audience" students feel.

The positive approach to the Web is not the same with the WBI tools we discussed before. From a pedagogical point of view, there is nothing new in the WBI tool approach. Similar to the ITS and old closed drill and practice systems, a classical WBI tool is mainly based on a transmission mode of teaching [Squ99]. This means that the WBI tool approach alone can not use the full potential of the WWW [Ree98]. Duchastel uses the phrase "just a nice way of publishing teaching notes" to refer to the inefficient utilization of the Web. Obviously people like Duchastel argue that new models addressing Web-based learning are needed; and already some suggestions have been published [Duc97] [Ree98].

As a result of our background research on Web-based instruction and the software used in that area, we decided not to go in that direction for several reasons. First of all there is already an enormous effort (research and development) both in the academic and in the industrial community. Secondly, we wanted our work to be based on a solid, strong educational theory, which was undoubtly constructivism. Constructivism is usually associated with cognitive tools rather than ITS or WBI tools. Therefore, we pursued the background work with the exploration of cognitive tools.

3.1.5 Cognitive Tools

In the broadest sense, Jonassen and Reeves define cognitive tools as "technologies, tangible or intangible, that enhance the cognitive powers of human beings during thinking, problem-solving, and learning" ([RJ96], p 693). "Something as complex as a mathematical formula or as simple as a grocery list can be regarded as a cognitive tool in the sense that each allows humans to "off-load" memorization or other mental tasks on an external resource" [RAL98]. Computers as cognitive tools, for example, should be used for storing, retrieving, filtering, and sorting information rather than conceptualizing, applying, analyzing, synthesizing, and evaluating information, which are things that humans

do much better. This role of computers can be described as intellectual partnerships in the form of distributed cognitive processing (between the learners and the computers). Also referred to as "cognitive technologies", "technologies of the mind", "mind tools", or "content free courseware", the term "cognitive tools" is widely used today. According to the list given in the cognitive tools framework of Jonassen and Reeves, one of the most cited reference in this field, the examples of cognitive tools in computer technology include ([RJ96], p 694) databases, spreadsheets, semantic networks (concept mapping tools), expert systems, multimedia/hypermedia construction software, computer-based conferencing, collaborative knowledge construction environments, computer programming languages, and microworlds.

We can say that the cognitive tools approach originated from the shift to thinking and learning processes versus outcomes, that is, the shift to active versus passive learners, or the shift to engaging versus teaching students, or the shift to meaningful versus easy learning or the shift to constructing versus communicating knowledge, or generally the shift to constructivist versus objectivist views. Cognitive tools are learner controlled not teacher controlled or technology driven: nor are they "fingertip tools" that learners use naturally, effortlessly, and effectively [JM94]. As Hannafin et al. state: "Ill-structured domains, of which there are many, require tools that support varied perspectives, encourage the students to analyze from a variety of points of view, and avoid artificial simplification of complex concepts for the sake of expediency. Easier is not necessarily better." ([HHH+96], p 395). Also, Jonassen and Reeves argue that cognitive tools should be rather unintelligent tools, relying on the learner to provide intelligence [RJ96].

The increasing research interest spawned many applications of the cognitive tools approach guided by constructivist principles, for example the Belvedere Project, Programmable Bricks and the PBLSS Project, which were mentioned in section 2.4.1. The RELATE (Realizing Environmental Literacy Through Advanced Technology and Experimentation) is the largest and the most serious project we encountered in the literature [RAL98]. The project has been developed at the University of Georgia, for the Environmental Literacy study of undergraduate students, and as of March 1998, more than 600 students have

participated. Periodically students are taken to the State Botanical Gardens of Georgia to collect data, and then they return to computer labs to complete data analysis and prepare their reports using cognitive tools such as a GIS data analysis software, and a research center simulation. The results show that students consistently perform better on final examination questions related to field research labs than on the questions drawn from any other parts of the course such as lectures and assigned readings. Judging by all these research findings and latest trends, we can conclude that the utilization of cognitive tools may increase gradually and that many different cognitive tools will appear.

3.1.6 Knowledge Construction and Representation Tools

During our survey about cognitive tools, we are particularly attracted by the knowledge construction and representation tools, and we have narrowed our research into that direction. The main reason of this interest was that we had constructed some preliminary work and devised some ideas about a software tool relevant to this field. Later we discuss and propose the tool, but before here we introduce knowledge construction and representation tools to lead that discussion.

As being a general term "knowledge construction and representation" may encapsulate a large meaning. Here we use it to refer to tools that enable us to construct and to represent a knowledge domain by visualizing the concepts and entities the domain is composed of, and the relationships between them. This process is also called "knowledge mapping" or "information mapping". Today, the most widely adopted knowledge mapping tool in educational contexts is the concept mapping technique, which was developed by Joseph D. Novak at Cornell University, in 1970s based on the Ausubel's cognitive learning theory.

As shown in figure 3.1 concept mapping is semi-formal diagraming technique, or a graphical representation, where nodes correspond to concepts and links represent the relationships between concepts. Both the nodes and links can be labeled. The links can be one-way, two-way, or non-directional. As Gaines and Shaw [BRG95] state, the term concept map is used to encompass a wide range of diagrammatic knowledge representations. For example, in many

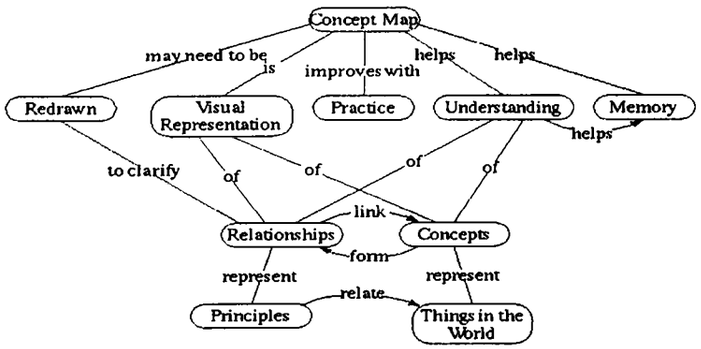

Figure 3.1: A concept map of concept map

of his publications, Jonassen uses the terms concept map and semantic network interchangeably, since there is no difference between them from the educational point of view [Jon98] [RJ96]. We can say that semantic nets are formal concept maps used to represent knowledge in the Artificial Intelligence (AI) field.

There are two key points that make concept maps powerful: visualization and the node-link mechanism. Is a picture worth a thousand words? Perhaps it is, perhaps not; but how could Einstein explain (and construct) his theories using words only: "The words or language, as they are spoken or written, do not seem to play any role in my mechanism of thought" Albert Einstein (cited in [McL97]). In short, pictorial and visual forms of representation can offer advantages over linguistic thinking [Plo97] [McL97] such as:

- ability to show spatial inter-relationships,
- proportional relationships within an object,
- facilitation of perceptual inference (e.g. relative size of objects),
- quickly and easy recognition, and
- development of holistic understanding that words alone cannot convey.

The choice of artifacts and their roles in a visualization context is the main decision before attempting to create a visual representation. In concept maps, nodes and links, as artifacts, enable the organization of concepts and their relationships in a way apparently similar to the human memory [RJ96]. Semantic

networking and schema theories underly that the ideas in human memory, known as schemas, are arranged (semantically) in networks (schemata) of interrelated concepts, which are associative rather than hierarchical. Because of this close relationship between concept maps and human cognitive structures, concept maps have been widely adopted by educational community. This is also the reason why they are called "cognitive maps". By making explicit what is normally implicit, concept maps allow people to present what they know and to see what they do not know. For this reason, concept mapping is also used as an assessment tool or to detect or illustrate the misconceptions learners may have as explanations of content matter.

3.1.7 Constructivism and Concept Maps

Within the constructivist approach to learning, concept mapping is seen as a cognitive tool that simplifies the knowledge construction process by engaging learners "in an analysis of their own knowledge structures, which helps them integrate new knowledge into existing knowledge structures"([RJ96], p707). Marta and Jonassen call this process "internal negotiation": "Just as meaning is derived and socially negotiated between individuals from a constructivist perspective, so individuals do the same with themselves"([JM94], p 5).

3.1.8 Applications of Concept Maps

Besides education, concept mapping has been used to support the interviewing process in knowledge acquisition from experts, to analyze organizational decision making, and to represent the process of conceptual change in scientific revolutions [BRG95]. Today, in the market there are different types of software packages that present concept mapping characteristics. The application areas they provide include:

- Brainstorming
- Hypertext (Web) design

- Communication of ideas
- Learning
- Assessment
- Computer aided thinking and decision making
- Presentation planning
- Information navigation

Some major packages currently available in the market are:

- Decision Explorer
- Inspiration
- Idons-for-thinking
- Axon Idea Processor

In addition to this list, some tools originating from academic work such as Belvedere mentioned in section 2.4.1 and KMap mentioned in section 3.1.10 are professional looking tools that have gained a substantial number of users. To what extend are these tools successful? The empirical results we encountered were oriented rather about learning. Reader and Hammond [RH94] found that providing students who are browsing through a hypertext environment with a concept mapping tool were more effective in terms of scores on a context-oriented post-test that those students who used a computerized note taking tool. Similarly, Jonassen and Wang (cited in [CJLM98]) experienced a significantly better performance in students who created semantic networks presenting what they had learned from hypertext compared to the students who created an unstructured set of nodes for the same purpose.

In addition to graphical outlining, organizing and concept mapping, the tools listed above have some other features desirable for such an environment. For instance, Decision Explorer, Idons-for-thinking and Belvedere support collaborative construction of maps in real-time. Inspiration has two additions,

namely professional and k-12; and provides predefined map templates and a text-based outline in addition to diagram view. Decision Explorer also offers more than one views for the same map. Yet another option it includes is analyzing the maps such as finding concepts that have great influence (central concept) or showing groups of closely related concepts (clustering). Axon Idea Processor has a flexible visualization system, and is capable of producing basic 3D visualization. Using Idons-for-thinking, one can also create mind maps, which are invented by Tony Buzan, a leading authority in the field of human brain and minds. A mind map consists of a central word or concept; around the central word generally 5 to 10 main ideas that relate to that word are drawn. Then each of those child words are taken and again 5 to 10 main ideas that relate to each of those words drawn. The resulting structure is a multi-branched tree where main concepts are close to center and supportive ones are close to leafs. The difference between concept maps and mind maps is that a mind map has only one main concept, while a concept map have several. This hierarchical structure prevents the mind maps to model the human mind properly. Today mind mapping programs such as Visimap and Mind Manager are widely used all over the world. Mind Manager is especially common because of its powerful features such as enabling users to link local files and Web documents to the nodes; and to export maps as Rich Text Format (RTF) and HTML documents.

QuestMap is a unique tool in that it is specifically designed for discussions only. The artifacts found in a discussion environments such as questions, ideas, pros, cons, and decisions are predefined as nodes. Starting with a main concept, a classical discussion is structured hierarchically around it by adding nodes of the types above and links between them.

3.1.9 The Way towards ReMemex

After investigating the concept mapping tools, we concentrated on our idea for a "research aid tool" which has similarities with these tools in terms of visualization. The result is ReMemex; an information mapping tool based on cognitive mapping, information retrieval, information filtering, and knowledge construction. Before going into the details of ReMemex, here we glue the last

piece to the picture of the "ReMemex Story": the need for an information organization and filtering environment.

Consider how the words of H.G. Wells in "The Brain: Organization of the Modern World", 1940 are impressive and valid in 1999:

"An immerse and ever-increasing wealth of knowledge is scattered about the world today...but it is dispersed and unorganized. We need a sort of *mentalclearinghouse*: a depot where knowledge and ideas are received, sorted, summarized, digested, clarified and compared" (cited in [Koy97]).

In 1945, Vannever Bush wrote his famous article titled "As we may think" [Bus45] where he describes an imaginary device, memex, using the following phrases.

"... It (the human mind) operates by association. ...Consider a future device (memex) for individual use, which is sort of a mechanized private file and library. ...Most of the memex contents are purchased on microfilm ready for insertion. Books of all sorts, pictures, current periodicals and newspapers are thus obtained and dropped into place. ...As he has several projection positions, he can leave one item in position while he calls up another. He can add marginal notes and comments... It affords an immediate step, however, to associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select immediately and automatically another. This is the essential feature of the memex. The process of tying two items together is the important thing. ...Moreover,when numerous items have been thus joined together to form a trail... It is exactly as though the physical items had been gathered together to form a new book. It is more than this, for any item can be joined into numerous trails"

Today Bush's vision is assumed to be the first idea of the WWW. Also, we could claim that Well's vision has came into reality with the utilization of database management systems, information retrieval systems and many other sophisticated software applications available. However, today in the Knowledge Age, we still need Well's mental clearing house and Bush's memex. Information is everywhere, even without any attempts people are pushed with information via TV and radio broadcasts, newspapers, and the WWW. Accessing, storing,

indexing, and filtering information is easy and cheap. The problem is the ever-increasing amounts of unorganized pieces of data that we have on our cluttered desktop at home or in the office. Paper documents and other types of media (videos, audio tapes etc.) also tend to pile up on desks and in file cabinets.

We believe that Bush's vision was the personal information landscapes that are organized according to mental models in human brain which the WWW cannot provide. ReMemex is an attempt to clear the desktops and enable organization of information with guidance of the user, once that information has arrived from a variety of sources to the person's work environment.

3.1.10 Overview of ReMemex and Related Applications

Here we give an overview of the main features ReMemex should have. Formal requirements are given in Section 4.1. ReMemex is basically a cognitive map, where nodes usually represent arbitrary data and links construct the relationships between the data. In this sense, a node can represent an image, a Web address, an article, part of an article, a video or audio segment, a journal in the library etc. Consider someone who makes research on a particular topic using ReMemex. While collecting information and data, he also inserts them into his research map, and specifies the relationships with the other objects that are already in the map. During this process, he gains substantial insights about the subject matter and sometimes restructures some parts of the map. By clicking any node, he can see the content of it, that is the data associated with it, in the associated external application. He also adds notes as new nodes or attach annotations to existing nodes.

After some effort, as the map is getting larger, he recognizes that some nodes are complementary. He uses the powerful grouping mechanism of ReMemex and groups complementary data into a subtopic, similar to the clustering mechanism found in other packages. However, ReMemex offers two different modes, open and closed, for a group-node which enable users to focus on information at any level. Here it should be noticed that group-nodes themselves can also

be grouped. For a higher level view, he switches group-nodes to close mode where they are represented as a single node and all the nodes they contain are hidden. In this mode, the links to these hidden nodes point to the group-node. For the details, he opens the group-node and hidden nodes and their links are restored.

After completing the map with sufficient information he decides to write a document about the subject matter. For this purpose he requests ReMemex to export the map into an RTF document and uses it as the outline of his report. All of these processes are done seamlessly by using pulldown menus, action buttons and mechanisms such as drag and drop, and cut-copy-paste. We believe that ReMemex can be applied to many areas including education and research. As we discussed before, concept maps are powerful cognitive tools. ReMemex adds more power to concept mapping by providing features explained above. However, we think that one more feature is needed: online collaboration. Imagine how the students would be engaged by constructing a ReMemex map collaboratively for their homework.

According to our observations there is not any single tool that incorporates the features of ReMemex. We encountered some similar ideas however. In Knowledge Science Institute at University of Calgari, Gaines and Shaw [BRG95], have developed KMap, a hypermedia concept mapping tool, where nodes can be linked to other concept maps and files either locally or across the Internet. The user can open the actual file anytime via the associated nodes. KMap also supports synchronous collaborative construction of concept maps. So far, it sounds very similar with ReMemex, but there are differences. KMap does not support grouping; all the nodes in KMap are shape based text nodes, that is, any image or other type of icons can not be used to visualize the nodes; KMap does not provide multiple view for a single graph and finally while it incorporates powerful scripting capabilities (through Apple's open scripting architecture), KMap works on Macintosh based platforms only.

A tool called BRAIN, can also be seen a kind of information organizer. It does not apply concept mapping technique, rather nodes are arranged hierarchically. A node may be linked to any file or to any Web document. The user

does not have control over some features. For example, a semi-automated layout prevents users to move a node to a particular place on the map. Also the users can not break the hierarchical structure. Yet another automated feature is that the selected node is made as the central node and only the directly linked nodes are shown around it, preventing users to see the complete structure (map). This is consistent with the aim of the tool, which is organization of data not the visual representation of this organization. Other restricted features are that all nodes are text based (labels only) and that there is not any support for grouping.

Concluding this chapter, we think that ReMemex is a very promising project and can be highly successful when implemented properly. Our aim in this work is to provide the base of such an implementation.

Chapter 4

Design

4.1 Requirements

The requirements of ReMemex are grouped by the main features of the system and presented in that manner. They are likely to be extended and modified after the evaluation of the system.

Importing Data

ReMemex should allow users to import any kind of data in electronic form including local files, Uniform Resource Locators (URLs), personal notes (note cards or text nodes), and other ReMemex maps.

The users should be able to copy any kind of data from other programs into ReMemex. The cut-copy-paste mechanism is thus essential. Also drag and drop (DnD) may be supported for ease of use. Think of a user who has many open documents of all sorts and a Web browser on his desktop. He drags a file from the system file manager and drops it onto ReMemex, and a node representing the file is created. Then, he copies some text from his word-processor, and paste it into ReMemex, and then links the created text node to the file node. After that, he inserts the Web address that the browser points into the ReMemex as an another node via a DnD operation. ReMemex

should support all these operations seamlessly.

Not all information is available as soft copy. Therefore, ReMemex should also provide mechanisms to create nodes that represent "off-line data".

Exporting Data

The user mentioned above should also be able to do the reverse process. ReMemex should allow nodes (the data represented by nodes) to be copied into other programs.

ReMemex should also have exporting mechanisms for the whole map. There should be at least two options: exporting the visual appearance and creating report-like documents. The visual appearance can be exported as an image file in a well-known format, such as gif and jpeg. For creating reports, formats such as html, postscript, portable document format (pdf), and rich text format (rtf) can be used. In this option, ReMemex should also deal with the data that can not be exported directly into a document of one of those formats. For this purpose, instead of the data itself, the label of the node, the location of the data, or the annotations user has made for that node, can be inserted to the document. The insertion sequence of the nodes can be taken from the user or extracted from the structure of the map, or from the selected set of nodes to be exported.

Multiple Views and Filtering

Multiple views and filtering are important in that they enable users to inspect the map from different perspectives. For example, a text based outline or a miniature version of the map could be other alternative views. Examples might be filtering (hiding) a particular type of node or link, the elements added after a particular date, or the elements added by a user.

This requirement is closely related to the flexibility of the resultant programming API. This means that ReMemex should provide a mechanism to programmers to write their own filters and views, and incorporate them into

the program easily.

Collaboration

The collaboration facilities in ReMemex should at least include synchronous editing of the maps and a text based communication during that process. According to the experiences with collaborative whiteboard applications, the best practice in such environments is that only a single user is allowed editing while the others are observing. To acquire the right to edit the map a user must raise his virtual hand. Once the user who owns the right completes his work, the system could apply a turn-based strategy to give it to others, or the coordinator of the group-work could manage this.

ReMemex may also offer some asynchronous communication facilities. The construction of maps may not be completed in a single session. In such cases, especially if the map is large and takes days or weeks to be completed, an asynchronous threaded discussion tool may be very helpful in keeping track of past discussions about the construction of the map. The messages in the discussion tool might be added to the map itself, too. A user should be able to logon to the system at anytime, and make modifications and add messages. If there are other users online at the same time, then the synchronous editing rules explained above should be applied.

Open Object-Oriented API

One of the objectives in the ReMemex project is to create an easily extendible and modifiable framework for iterative development. A carefully designed object-oriented class library works well for this purpose. Especially in case of large software projects. object-oriented approach has been proven to be more successful because of the well-known features it provides, such as ease of coding using good encapsulation, modularity, and abstraction principles, predefined reusable patterns, and rapid prototyping.

In an academic environment it is common practice for large software projects

(eg. the subject of MS theses), to be developed iteratively by graduate students, over a period of several years to finished. For this reason, the need for an extendible API is one of the most important requirements, and in fact, one of the hardest to achieve.

Grouping Nodes

As mentioned in Section 3.1.10 grouping is a unique feature of ReMemex, which enables users to abstract the details when the map gets larger. All types of nodes including the group-nodes themselves should be groupable. Group-nodes should have at least two different views that depict open and closed states.

A group-node itself is a node. This means that all operations applied to an ordinary node should be available for group-nodes, too. For example, the user should be able to link a group-node to another node.

When nodes are grouped, the links that connect the nodes in the group with the ones outside the group should be kept. Upon the close operation on the group-node those links should point to the group-node, and as the group-node is expanded those links should be restored to their original state.

Node and Link Appearance

ReMemex should provide set of shapes and images for node icons, and different connector shapes to represent links. Also, different fill patterns and line styles should be available and node icons should be resizable.

A node icon may have multiple views. The ReMemex API should provide open mechanisms to add custom views to the application without too much effort. For example, a node representing an image file might have two visual modes: "icon mode" and "thumbnail image mode". For links, multiple views do not make much sense.

Data Types, Node Types, and Link Types

ReMemex should allow users to define new data types. A data type definition should contain necessary information that is used to identify, transfer, and open data of that type. This information may include the content description, the file extensions associated with the data type, and the applications that are capable of manipulating (displaying, editing, etc.) data of that type. Here we assume that the content types are specified using MIME notation; and the contents of the files are specified using file extensions.

ReMemex should allow the user to define new node types. A node type definition may include a default icon for the nodes of that type, the associated data type (the content), associated file extensions, and applications. The following example makes the distinction between a data type and a node type clear. Assume that we have defined a data type called "plain text" and specified that the MIME type of the content is text/plain, the file extensions are "txt, text" and we also listed several applications that can process plain text data. Now, consider that we want to add C source files into ReMemex. Although a C file contains plain text data, the system can not recognize the file as a known type. What if we add "C" extension to the associated file extensions list of plain text data type? In that case, the node representing a C file would be treated as an ordinary text file, that is, its icon would not be different from that of ordinary text and, when requested, a text processing application would open it, not a C programming environment. Defining another data type for C files is not the solution, because we already introduce the type of the data (content) of C files to the system: it is plain text. These kind of problems can be handled by a node typing mechanism. For example, in this case we define a new node type to represent C files, and specify that the *associated data type* is plain text, the file extension is "C", and the associated application is a C programming environment. In addition to the user-defined node types, ReMemex should automatically provide a default node type for each data type, eg. a default "plain text node type" for "plain text data type". Each node in the map should have a particular node type, and not all node types have to be associated with a data type. For example, one can create a node type to represent books in the library. This type does not have a data type because there is no data available

as soft copy. In this manner, anything can be represented in the map provided that its node type is created.

ReMemex should allow users to define additional properties for the node types. In the book-type example, the name of the book, the authors, and publishing date might be useful additional properties for the type. When the users create a book node, they should be able to enter these properties of the book easily.

ReMemex should also support link types. Links are not associated with data types, but otherwise should work like node typing.

Styles

Style support is not an essential requirement for ReMemex, but we think it might be extremely useful. A style is a collection of node and link types that are customized for that style. The customization may be applied to the appearance, that is, to the default icon of the types. This means that a book node may look like different in different styles. It is not necessary for a style to incorporate all the available node types in the system. Why are styles extremely useful? Two immediate reasons are customization and categorization of the nodes. Example style names might be *Research Style, Homework Style, David's Desktop Style, Emek's Thesis Style* etc. Styles can also be used to enable ReMemex to mimic various kinds of graphical outlining tools. For example, the QuestMap tool mentioned in Section 3.1.8 can be incorporated into the ReMemex environment by defining the necessary node types such as question, answer, decision, and idea, and then by adding them into a style, called for example *QuestMap Style*. By making the QuestMap style the currently active style, users can add question nodes, idea nodes, and other nodes defined in that style.

4.2 Design Overview

Designing large projects is hard, and achieving reusability and extendibility in these projects is even harder. In a traditional object-oriented software design process, first the pertinent objects are identified, and factored into classes at the right granularity. Then, class interfaces and inheritance hierarchies are defined and key relationships among them are established. All these processes include very critical decisions which play great roles in the later phases of the project. Designers always try to fit solid entities or concepts into classes with minimal dependencies, so that they are reusable and changes made to a class do not propagate other changes throughout the code. However, it is not always that straightforward.

4.2.1 Design Patterns

Probably, the design patterns movement is one of the most important steps forward in object-oriented design. The book called "Design Patterns Elements of Reusable Object-Oriented Software" by Gamma, Helm, Johnson and Vlissides[GHJV94] is known to be the major source about design patterns; a source of what has now become an essential, almost mandatory, vocabulary for object-oriented programmers. Simply, a design pattern is an especially clever and insightful way of solving a particular class of problems. The authors define it as follows: "Design patterns are not about designs such as linked lists and hash tables that can be encoded in classes and reused as is. Nor are they complex, domain-specific designs... (They) are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context."([GHJV94], p3)

Although they are called "design patterns" not all of them are applied or tied to the design phase of the traditional analysis-design-implementation view. For example, the simplest pattern, Singleton, which is a way to provide one and only one instance of an object is usually applied at the implementation phase, while the Model-View-Controller (MVC) pattern described below is rather a meta-level strategy and applied at the high-level design stage. In the

ReMemex project we used various patterns including Singleton, Command, MVC, Abstract Factory, Decorator, and Mediator.

4.2.2 High-level Design

There are several alternative architectural approaches for systems that need communication over the Net. These days multi-tier architectures are preferred over the traditional client-server paradigm. Both of these approaches require a central server application that coordinates the communication. Yet another approach for ReMemex might be the non-central server architecture where each client program is capable of managing server-like operations. This approach is usually used for software, such as multi-player computer games, where there is no need for a permanent common data storage. Whichever approach is used a client program is essential. In case of ReMemex, this client application should also work in off-line mode, that is, in single user mode as a stand-alone application. Therefore, within the scope of this thesis, our focus is the development of the stand-alone client part of the ReMemex system, while keeping in mind that it will be integrated into a multi-user platform. Throughout the rest of the thesis "ReMemex" is used to refer mostly to this single user version.

A first observation of the specifications clearly suggests that the visualization subsystem could be developed as a general library that can be used in the systems that require similar visualization infrastructures. The visualization subsystem, in turn, is very suitable for the MVC pattern and can be divided into other subsystems in that manner. MVC was originally a basic design paradigm in Smalltalk to build user interfaces. The Model is the application object implementing the basic functionality, the View is its screen presentation, and the Controller defines the way user interface reacts to user input [GHJV94]. MVC decouples views and models by establishing a subscribe/notify protocol between them, which enables multiple views to attach to a single model, one of the requirements of the ReMemex's maps and nodes. Whenever the model's state or data changes, it modifies its subscribed views, and the views update themselves to reflect the new state of the model. Controllers, on the other hand, handle external modification to the model-view system and transfer these

modifications to the model.

The MVC paradigm has been extensively used in the computer science world, such as in a heapsort visualization where the model is the algorithm with data, and the two views are a balanced binary tree view and a typical array view of the heap. [NBJP+97] As in this case, simulation of the algorithms and visualization of the data structures fits very well to the MVC pattern. This is another advantage for ReMemex since its backbone is based on a graph structure. Today, one of the widely used practices of the MVC architecture is a slightly modified version of it where the controller is embedded into the View. The windowing component library of Java, known as Swing, is a well known example that is based on the modified MV version [Fow98]. We also applied MV instead of MVC in the design of ReMemex.

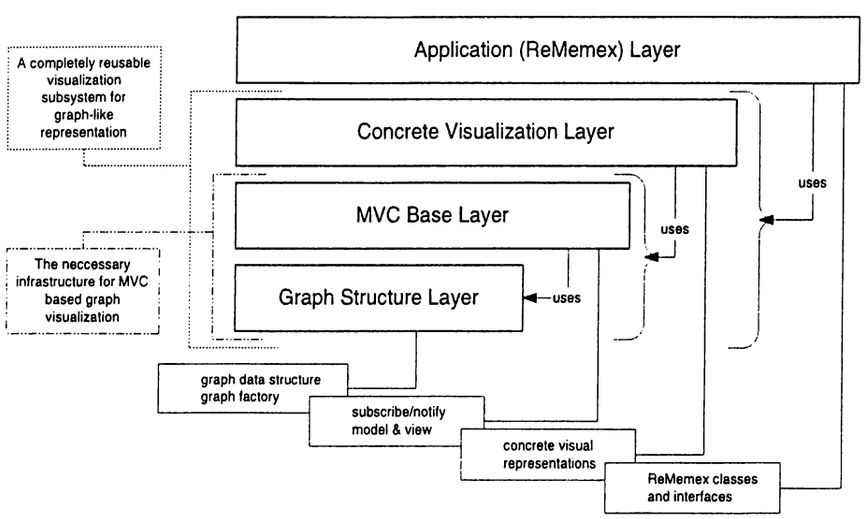

Figure 4.1: The layered architecture of ReMemex

Figure 4.1 depicts the high-level view of the ReMemex's layered architecture. Each layer is built on top of the layers below it and, in an additive manner, they construct a subsystem that serves the upper levels. To achieve loose coupling and reusability, the relations (dependency) between the layers has to be unidirectional, that is, from top to bottom. In other words, no object in a layer is aware of the objects that are in upper layers. The Graph Structure layer and the MVC Base layer, together provide a compact, flexible and extendible base for the MVC visualization of graph based interfaces. The third layer, the Concrete Visualization layer, provides the real visual artifacts displayed on

the screen. It can be thought of as being a concrete implementation of the MVC Base providing a different *look and feel* from those of the other possible implementations. Finally, on the top there is the Application layer as usual, which contains classes and interfaces that are rather ReMemex specific. The important details of each layer are discussed in the detailed design below.

4.2.3 Detailed Design

In this phase, in addition to the design patterns, we have applied two other design strategies as appropriate. They are known as "program to an interface, not an implementation" and "favor object composition over class inheritance". [GHJV94] In fact design patterns themselves also use these mechanisms to form their structure. There are some trade-offs in the use of these mechanisms, however, this topic needs a detailed discussion that is not appropriate to present in this thesis. In short, the main reason for their utilization is the flexibility they provide, and the main negative point in their utilization is the scattered view of the collection of interfaces and classes, which may cause complexity problems in large projects. For this reason, we also tried to establish this balance in the design of ReMemex. In the following sections a Unified Modeling Language (UML) diagram is given for each layer of ReMemex. The diagrams do not (and can not) contain all the classes, interfaces, and relationships, but should give a sufficient overview of the design.

Graph Structure Layer

This is the simplest layer containing classes and interfaces necessary to maintain graphs. This layer does not offer any visualization or view support. The main entities like graph, vertex, and edge are defined as an interface to allow different implementations of them to be embedded into the system. AbstractGraph is a base implementation for the Graph interface, and provides common implementations for some of the operations leaving the rest unimplemented (i.e. to be implemented in concrete subclasses.) For instance, in addition to

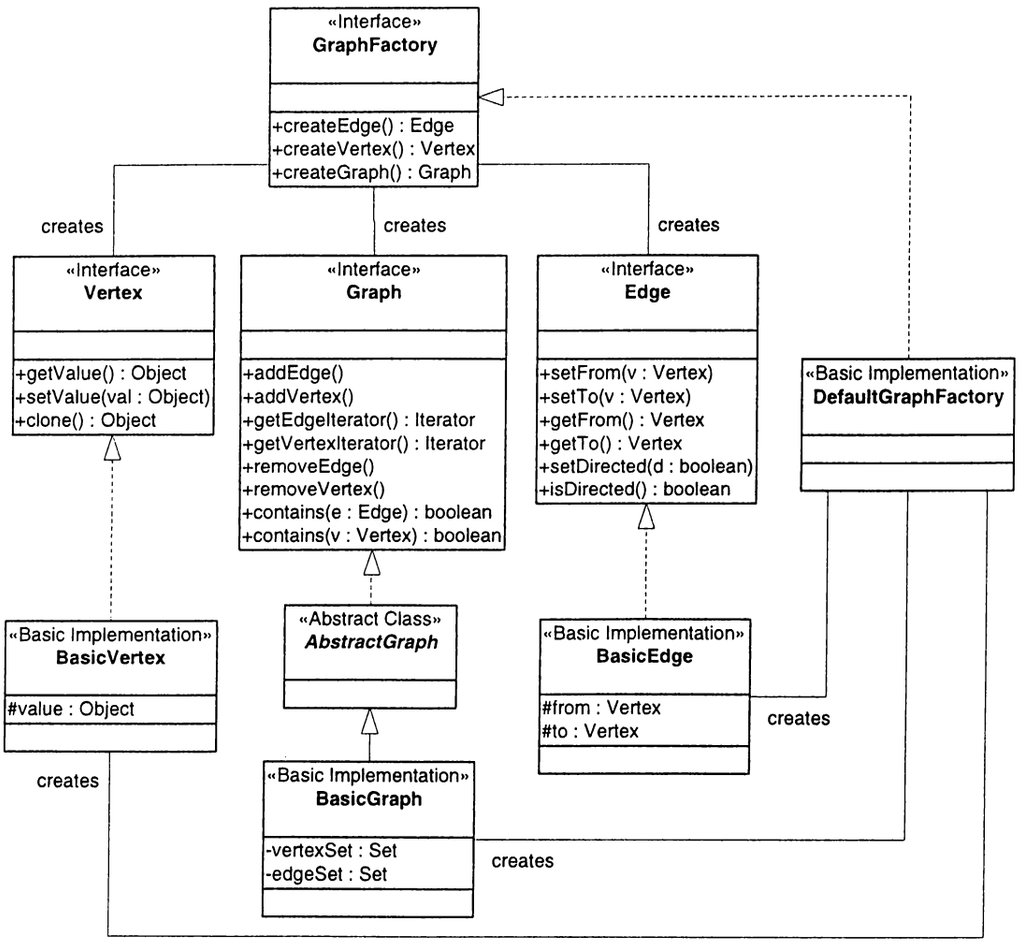

Figure 4.2: Graph Structure Layer

BasicGraph, two different concrete subclasses might be AdjListGraph and AdjMatrixGraph, which are adjacency list and adjacency matrix implementations of the graph data structure. The Basic... classes are the default basic implementations of the interfaces. The GraphFactory interface is an example for the *Abstract Factory* pattern and is used to attach a set of (related) vertex, edge and graph objects to the system. For example, if we want the system to work with AdjListVertex, AdjListEdge, and AdjListGraph classes that we might write for a special application, we would need to add a GraphFactory to the system, say AdjListFactory, which creates instances of these classes.

MVC Base Layer

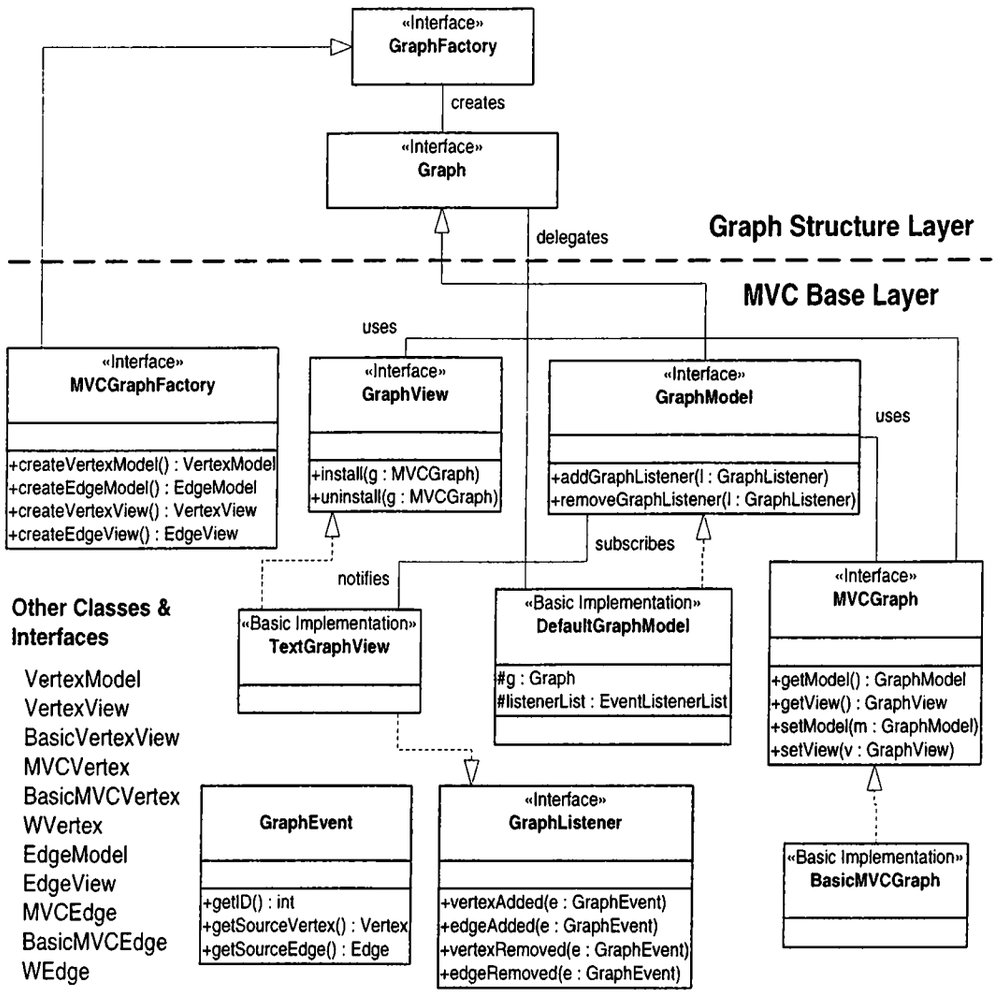

Figure 4.3: MVC Base Layer

Coming to this layer, the roles of the main entities in the Graph Structure Layer are changed. They provide the base for the model related interface and classes; e.g. GraphModel is derived from Graph and DefaultGraphModel delegates the Graph interface calls to a concrete subclass of the Graph interface, an example of the composition strategy. In this layer the classes that implement the MVC... interfaces represent the main entities, e.g. BasicMVCGraph. They keep a model and a view and play a mediator-like role. For example, when the model is changed, it notifies the view so that the view unsubscribes from the old model and subscribes to the new one. One of the important aims of this layer is to provide this subscribe/notify mechanism to enable views to

be updated as the model changes. This mechanism is, in fact, the Observer pattern, where the classes implementing GraphListener are termed as Observer (e.g. TextGraphView) and the classes that implement GraphModel are termed Observable. In this scheme, the GraphEvent encapsulates the modification information, and can be called ObserverState. In figure 4.3 there is not any class or interface corresponding to the Controller concept in the MVC architecture. Controllers are assumed to be either embedded into the View classes or defined in upper layers. Another thing that is expected from the subclasses is implementations of concrete view classes. There is nothing about visual appearances in this layer, rather it just structures or provides the backbone of the MV pattern. Finally, the MVC Base Layer has a factory mechanism which enables production of MVC related objects in addition to the objects based on the Graph Structure Layer.

Concrete Visualization Layer

As its name indicates, the Concrete Visualization Layer is expected to be a MVC-based visualization subsystem for graph-like user interfaces. Different implementations can be ported to this layer. We provided an implementation based on Swing, the widget (component) framework of Java. In designing such a visualization subsystem, the choice of the implementation language can be important in designing the details. The designers can refer to some language specific libraries and class hierarchies in this layer. Although the overall design we have come up for this section is a general solution, there are also some Java specific parts. In Section 5.1 we present the motivation and reasons behind the choice of the Java as the implementation language.

The concrete visualization layer is quite large. For this reason, in figure 4.4 we give only some important classes and relationships without much details. The upper diagram is relatively simple to inspect. The *J notion* in naming classes is taken from Swing where component class names start with the letter *J*. The model and view interfaces from the MVC Base Layer are not inherited further by J..Model and J..View interfaces, rather their abstract, incomplete implementations are provided, which serve as the base for model and view

classes in this layer. Abstract classes are favored against interfaces when the code should not be scattered further. The reduced flexibility due to this choice should not matter as this is an implementation layer that doesn't require as much flexibility as the base classes do.

It is obvious that the GraphItem interface has a dominant place in this layer. This interface is a conceptualization of the idea that any object can be added to the graph provided that it implements the interface. The interface provides the necessary operations to manage an item on the graph. In this manner, one of the requirements that a GraphItem has to provide is a component that is supposed to contain the appearance of the corresponding item. This mechanism has potential benefits; for instance the predefined components can be added to the graph by putting them into a wrapper class that implements the GraphItem interface. Also, this mechanism allows all kind of views to be integrated to the system. Consider two different view classes where one of them uses some shape objects and the other uses images to construct the visual appearance. All the system requires from them is to implement AbstractJVertex and provide a component that contains the appearance. This means that the view classes should keep a component instance and draw the shape or put the image on it. When we try to add vertices that use this kind of views to a graph, the graph's view need not be aware of their internal visual representations; it just requests their components and adds them to its Panel.

In this layer, besides the classes and interfaces related to the main entities (vertex, edge and graph) there are other large subsystems such as the icon-shape subsystem, and the drag-and-drop subsystem; collection of class hierarchies that are essential parts for such a visualization system.

Application Layer

The Application Layer in ReMemex is a quite large package composed of several subsystems. The Edge-Vertex concept becomes Node-Link concept in this layer, and the two classes Node and Link are designed to be the base of possible special nodes and links. The design of these base classes directly reflects the requirements: nodes and links are based on types and styles, and nodes are

capable of keeping a content. A Content object, in turn, keeps information about the data it represents. The type of the data is one of the essential items of information about it, since the visual appearance, some important processes, such as data transfer, and the associated external applications are based on data type. The base DataType class provides mechanisms to store and manipulate information about a data type, and subclasses provide concrete information about the data type they represent. This design allows the users to define their own data types via an appropriate user interface, as well as programmers to write data type classes that perform some special operations or provide additional features such as different views (e.g. TextAreaView provided by PlainText).

The classes AppSettings, ReMemex, and AppWindow are rather meta-level classes. AppSettings is a static class that keeps and manipulates application-wide settings and information such as the currently used map factory to create node and links, or current country and language settings (Locale). The ReMemex class can be thought of as the main class of the application. It creates an AppWindow at the beginning and is capable of creating other AppWindows whenever requested. The AppWindow class conceptualizes an application window with its building blocks. These building blocks include action and toolbar managers, menu managers, undo/redo managers, the collection of documents in the Multiple Document Interface (MDI) environment, etc. Each of these parts are subject to a detailed design within themselves.

Since grouping is one of the important features, we also placed the main classes related to grouping to figure 4.5. Briefly, the GroupedNode is a Node that utilizes view (GroupedNodeView) and model (GroupedNodeModel) objects that manage the processes and visual representations required by the grouping mechanism.

Obviously, it would take too much space to explain the design of all the subsystems, the other approaches we applied to arbitrary parts of the design and the (thankfully rare) redesign efforts we had to make. Not all the details are necessary. At this point, we trust that the readers' view of the ReMemex's internal structure is sufficient and the next chapter presents some implementation issues to make this view more clear and concrete.

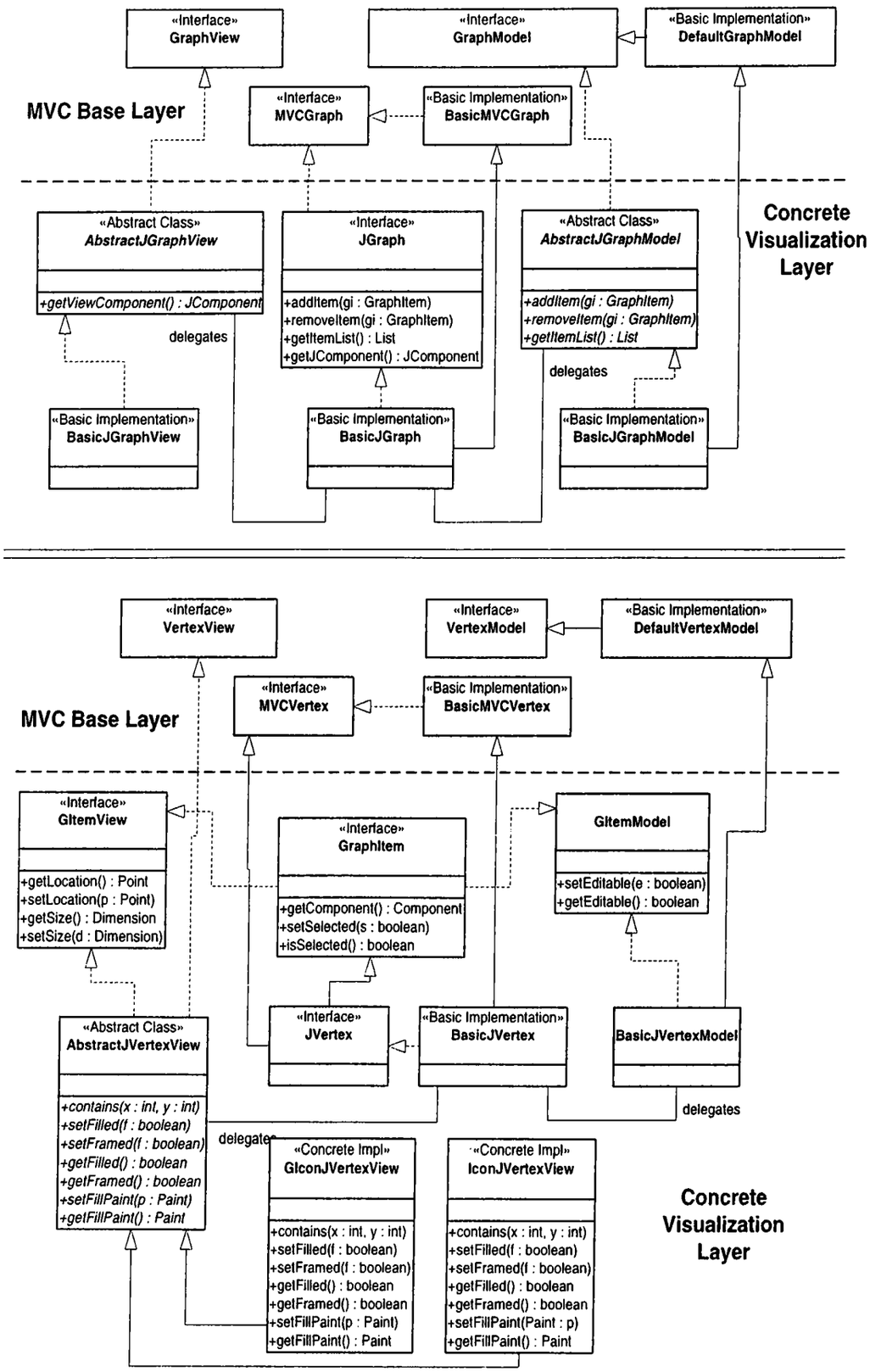

Figure 4.4: The Concrete Visualization Layer

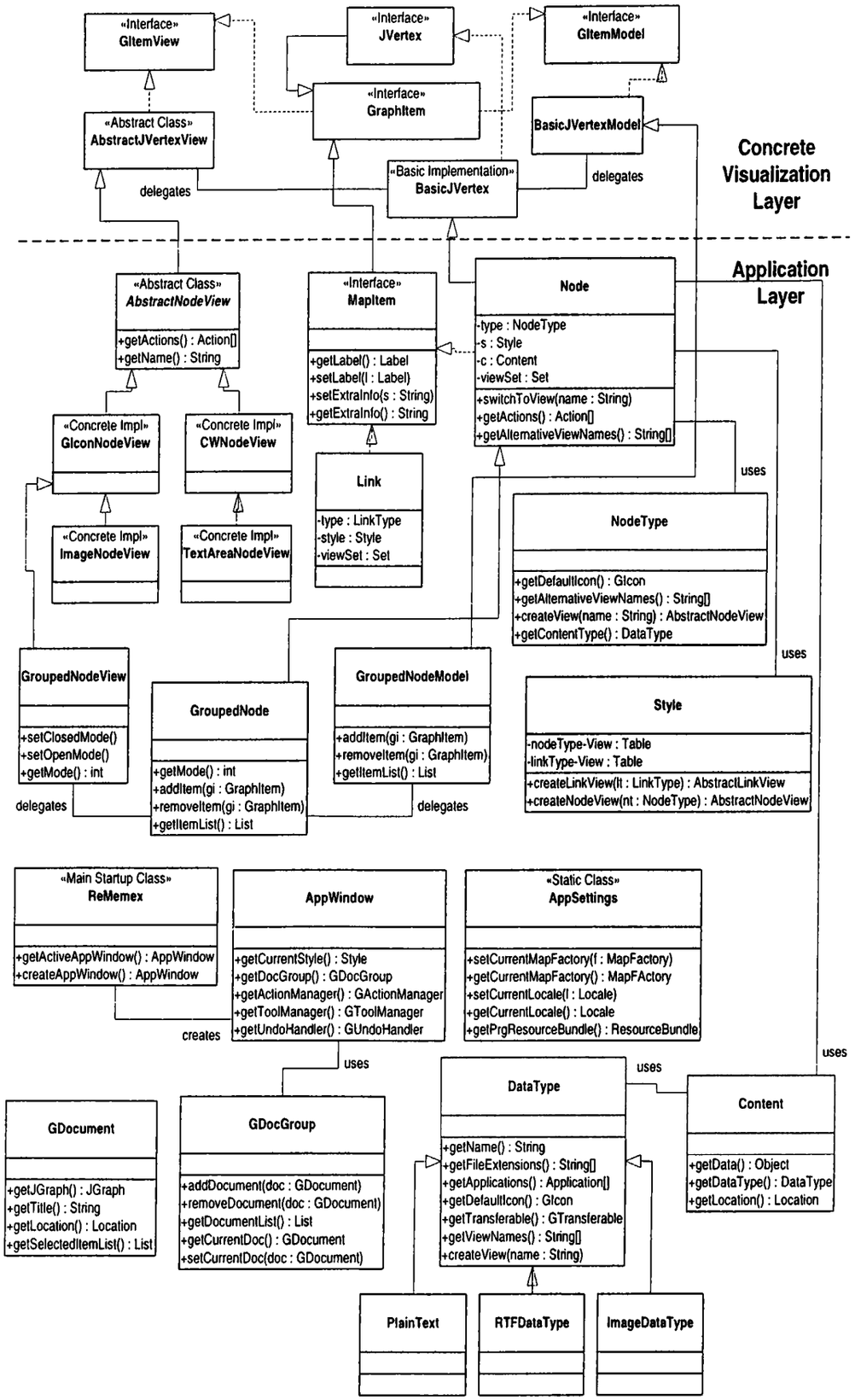

Figure 4.5: Application Layer

Chapter 5

Implementation

Implementation was the major part of this thesis and completed in a seven months long period; from March 99 to November 99. The complete API consists of fifteen packages, more than two hundred classes, and about thirty five thousand lines of Java code. The implementation does not incorporate all the requirements given in Section 4.1, rather it establishes an extensible base for further development.

5.1 Why Java?

The well-known advantages of Java are clear: Platform independent, easy to program and understand, clear syntax without complex mechanisms such as pointers, truly object-oriented, good documentation of APIs, support for networking, etc. These features are important, however, more importantly some of the packages provided in the standard API of the language are desirable for the implementation of ReMemex. They are listed below.

- The Java 2D API provides the basis for creating and managing arbitrary shapes, for defining fill and stoke patterns, for controlling the quality and speed of rendering, etc.

- The Data Transfer API forms a basis for transferring data of any type via a clipboard mechanism within the same Java Virtual Machine (JVM), between two different JVMs and between a JVM and any native application.
- The Drag and Drop API utilizes the Data Transfer API and provides mechanisms to transfer data by dragging and dropping. As in the case of the Data Transfer API the DnD API can also work for *intra-JVM*, *inter-JVM*, and *JVM-native application* combinations.
- The Swing API is basically a collection of extremely flexible Component APIs, including also a powerful text package with RTF and HTML support, and a basis for undo/redo mechanism.
- The RMI and CORBA APIs establish an Object Request Broker architecture that provides high-level support for the development of distributed applications.

Except the RMI and CORBA APIs, all the others given above are highly utilized in the implementation. However, we encountered several problems originating from the bugs in these packages. Especially, the Data Transfer and DnD APIs contain several major bugs, which prevent them to act in their full potential. We, also, have found a bug in the deserialization process of the *java.awt.datatransfer.DataFlavor* class and Sun MicroSystems accepted it as a new bug. Some bugs have work-arounds while some do not. For example, the major bug in the Data Transfer API does not have a work-around and unfortunately, currently it is impossible to transfer data other than plain text, RTF, or file list. The latest Java Platform that Sun has announced is JDK 1.3 (Kestrel), and they plan to ship it in the first quarter of 2000. The beta version of Kestrel is available and still it contains that bug in the DataTransfer API. It seems that it will be there in the final version, too.

There are some other disadvantages. In order to gain insights of large APIs, one has to spent considerable time. Yet another fact developers do not like is that applications written in Java are slow and require large amounts of memory to work properly. Since ReMemex is also a quite large system, we have paid attention to the performance issues all the time during development. Throughout

the code, we have utilized some optimization strategies and coding conventions. For example, two of the strategies are *lazy instantiation* [BW99], and *object caching*, which are related to the object creation process. The lazy instantiation technique defers the creation of objects until they are actually required. This process reduces the memory usage. Object creation and destruction in Java is extremely expensive [SNR+]. Object caching technique reduces the number of instantiations by keeping unused objects, and initializing and reusing them later when an object of that type is required.

The coding patterns we applied are simple. For example, instead of the common "for loop":

```
for (int i= 0; i < size; i++)
```

the following notation is used whenever appropriate

```
for (int i= size; --i >= 0; )
```

In Java, the *size* variable in the first loop above is accessed from the memory in each iteration. The aim in the second usage is to get rid of this. It is reported that the second loop is 25% faster in average [SNR+].

Despite all of these extra efforts, the choice of Java could helped us a lot, if there were not the bugs. One clear lesson learned from this experience is that before choosing Java, one should consult to the bug database of it in order to be sure that there is not any bug preventing the application to be implemented fully.

5.2 Implementation Flavors

The main features of ReMemex implemented so far are listed below.

- A visualization subsystem based on Swing and Java 2D
- DnD support for transferring data
- Cut-copy-paste support for transferring data

- Grouping support with full features
- Multiple view support for nodes
- Multiple view support for maps; the miniature view as an example
- Custom action support for links and nodes
- Configurable action bar and tool bar support
- Multi-lingual support

Figure 5.1 is a screen dump of the resultant program with a sample map about object-oriented design patterns. In the map, there are html nodes (e.g. *Patterns Home Page, Amazon, Thinking in Java, etc.*), image nodes (e.g. *GOF Patterns Book, Objectifier2, etc.*), rtf nodes (e.g. *Objectifier1*), plain text nodes (e.g. *From the preface*), shape based nodes (e.g. *Singleton, Facade, Observer*), and group-nodes (e.g. *Structural Patterns*). This is the most detailed version of this map. All the group-nodes are in open mode. In figure 5.2, some of the grouped nodes are collapsed (closed) and a more abstract view is provided. It should be noticed that the links to the nodes within the closed group-nodes show the group-node in this mode. Also, to illustrate some capabilities of the program Figure 5.2 includes other changes. The view attention node (*ATTENTION.txt*) is switched to "icon view". Similarly, the view of some image nodes are changed (e.g. *Objectifier2 from icon-view to thumbnail-view, or Relationship Patterns from thumbnail-view to icon-view*).

The resulting class hierarchy of ReMemex is quite large. Therefore, here, we give some special implementations only. The explanations contain sufficient details to assist to any possible further development of the project.

5.2.1 Actions and Commands

The action-command mechanism is important in that it offers seamless ways of extending the application and increases the maintainability of the project. All action and command objects are placed into the *rememex.command* package. In ReMemex, all the operations that modify the state of the map are

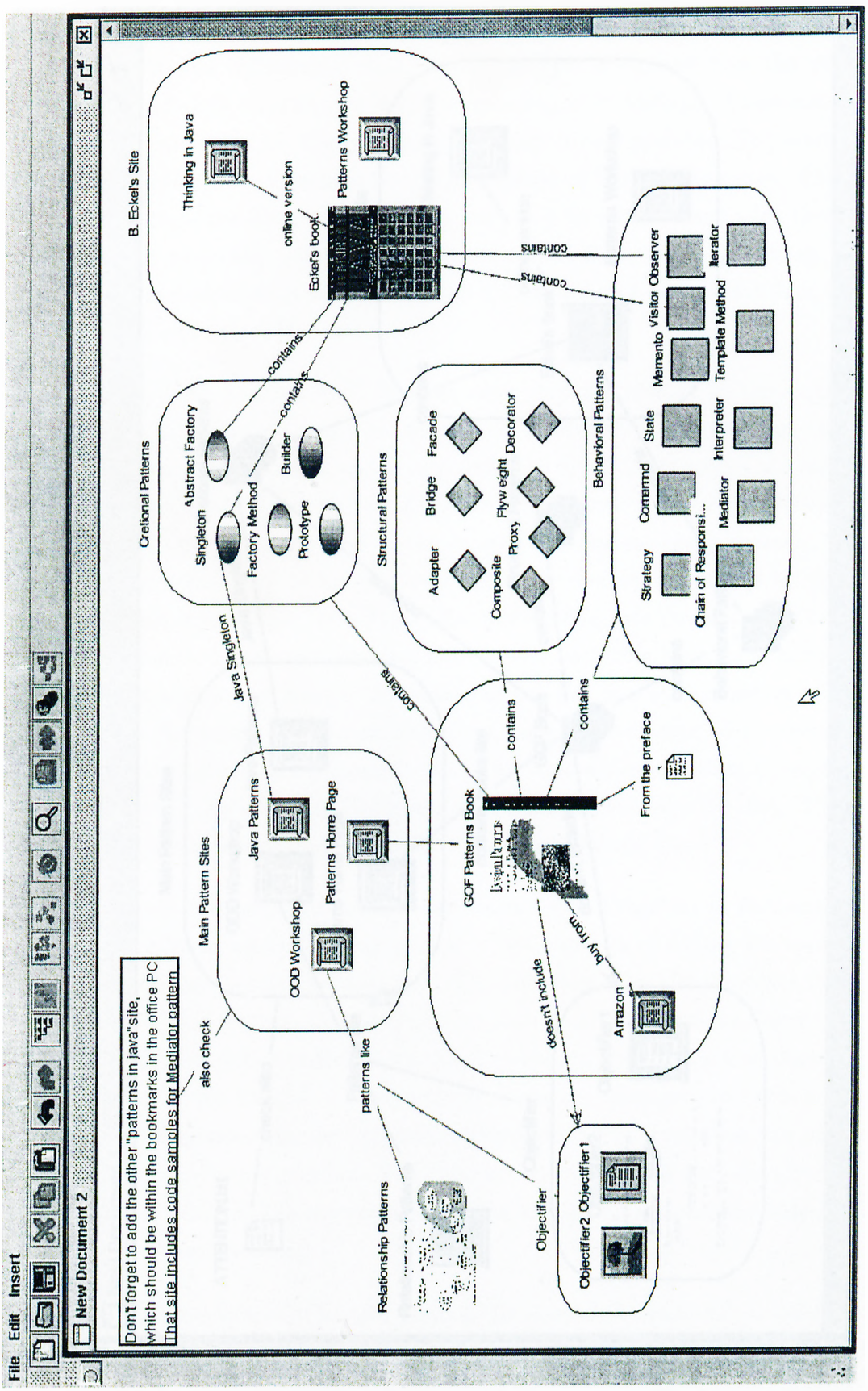

Figure 5.1: A screen dump of ReMemex: Sample map about Design Patterns

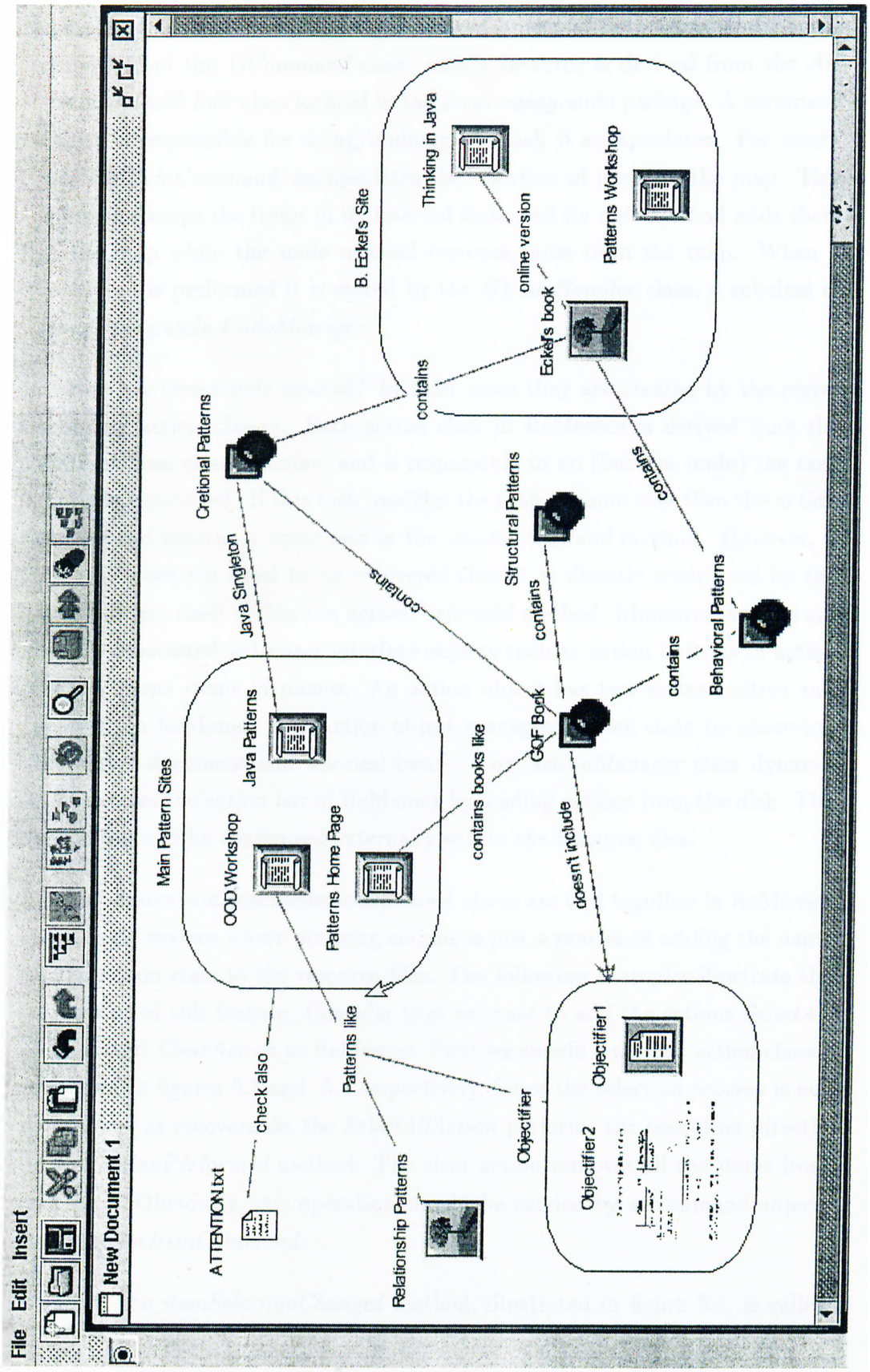

Figure 5.2: The modified version of the sample map

completed by the corresponding command objects. Each command class is derived from the *GCommand* class, which, in turn, is derived from the *AbstractUndoableEdit* class located in the *javax.swing.undo* package. A command object is responsible for doing/undoing the task it encapsulates. For example, *AddItemCommand*, encapsulates the insertion of items to the map. This command keeps the items in its internal state and its *redo* method adds them to the map while the *undo* method removes them from the map. When a command is performed it is stored by the *GUndoHandler* class, a subclass of *javax.swing.undo.UndoManager*.

How are commands created? In most cases they are created by the corresponding action classes. Each action class in ReMemex is derived from the abstract base class *GAction*, and is responsible to do (but not undo) the task they are associated. If this task modifies the map in some way then the action creates and returns a command in the *createCommand* method. However, if the task does not need to be recovered then it is directly completed by the action object itself within the *actionPerformed* method. Moreover, actions are usually associated with user interface objects such as action buttons in action bars or menu items in menus. An action object has two states: active and passive. In ReMemex each action object manages its own state by observing the active document and selected items. The *ActionManager* class dynamically creates the action bar of ReMemex by loading actions from the disk. The action bar can be configured externally within the resource files.

The classes and mechanisms explained above are tied together in ReMemex resulting a system where plugging actions is just a matter of adding the name of the action class to the resource files. The following examples illustrate the utilization of this feature. Consider that we want to add the actions *SelectAllAction* and *ClearAction* to ReMemex. First we should write the action classes as shown in figures 5.3 and 5.4 respectively. Since the selection process is not considered as recoverable, the *SelectAllAction* performs the operation directly in the *actionPerformed* method. The clear action removes all the items from the map. Obviously, this operation should be carried by a command object, say *RemoveItemCommand*.

Here, the *itemSelectionChanged* method, illustrated in figure 5.5, is called

```
package rat.command;
import java.awt.event.ActionEvent;
import java.util.*;
import javax.swing.ImageIcon;
import graph.swingbased.*;

pub lic class SelectAllAction extends GAction {
    pub lic SelectAllAction() {
        super("Select All", new ImageIcon("images/selectall.gif"),
             "Select all items");
    }
    protected void itemSelectionChanged() {
    }
    public void actionPerformed(ActionEvent e) {
        JGraph jg= getDocument().getJGraph();
        GraphItem[] items= jg.getItems();
        for (int i=items.length; --i>=0; )
             items[i].setSelected(true);
        jg.draw();
    }
}
```

Figure 5.3: Sample action: Select All

```
package rat.command;
import java.awt.event.ActionEvent;
import javax.swing.ImageIcon;
import graph.swingbased.GraphItem;

public class ClearAction extends GAction {
   public ClearAction() {
      super("Clear", new ImageIcon("images/clear.gif"),
           "Clear the map");
   }
   protected void itemSelectionChanged() {
   }
   protected GCommand createCommand(ActionEvent e) {
      GraphItem[] items= getDocument().getJGraph().getItems();
      return (items.length > 0) ?  new RemoveItemCommand(items) :  null;
   }
}
```

Figure 5.4: Sample action: Clear All

```
protected void itemSelectionChanged() {
  setEnabled( getDocument().getSelectedItemList().size() > 0);
}
```

Figure 5.5: Method itemSelectionChanged

```
package rat.command;
import javax.swing.undo.*;
import graph.swingbased.*;

publicclass RemoveItemCommand extends GCommand {
      private GraphItem[] items;
      publicRemoveItemCommand(GraphItem[] items) {
            this.items= items;
      }
      public void undo() throws CannotUndoException {
            for (int i=items.length; --i>=0; )
                doc.getJGraph().addItem(items[i]);
      }
      public void redo() throws CannotRedoException {
            for (int i=items.length; --i>=0; )
                doc.getJGraph().removeItem(items[i]);
      }
      public String getUndoPresentationName() {
            return "Undo remove item";
      }
      public String getRedoPresentationName() {
            return "Redo remove item";
      }
      public String getPresentationName() {
            return "Remove item";
      }
}
```

Figure 5.6: RemoveItemCommand class

when an item is selected or unselected, and used to enable or disable the action according to the selected set of items. By default, in the parent class, *GAction*, this method is defined to reflect the most common case, that is, the action is disabled if no items are selected, it is enabled otherwise:

SelectAllAction and *ClearAction* override this behavior to make themselves available even if no item is selected. After writing the action classes, we should write the *RemoveItemCommand* class as shown in figure 5.6 if there is not one already.

Now that, we can add the actions to the action bar and/or to the menus by

inserting the action class names to the appropriate places in the resource files.

actionBar= NewAction OpenAction SaveAction - **SelectAllAction ClearAction** - CutAction CopyAction PasteAction - UndoAction RedoAction

The benefits of action-command mechanism are evident. Yet another potential help of this development practise might appear in the development stage of the collaboration subsystem. Since all the operations are initiated and performed by the actions and commands, to inform the other distributed parties about an operation the serialized light-weight versions of the commands or actions can be sent to them. The light-weight command might not include the actual data the operation is applied to, rather a common notation can be employed to represent the data. Upon arrival of the light-weight command, the corresponding local data and command could be determined and the command might instantiated and performed.

Also the tool-toolbar implementation has been done in a similar way to action-action bar implementation. Tool related classes and interfaces are located in the *rememex.tool* package. The main difference between them is that tools use an attach/detach mechanism to register/unregister to the document to observe, for example, the mouse events.

5.2.2 Icons and Shapes

Here, we explain the *lib.shape* package, which provides the necessary support for creating shapes, connectors, and icons. The *GIcon* interface is the central component of the package. A GIcon is resizable, serializable, and cloneable. The rendering options of a GIcon such as its fill pattern or the quality of rendering can be set explicitly. Finally, a GIcon notifies interested objects when its state is changed, (e.g. when its line stroking pattern is changed). It is used in most of the classes related to icon based visualization many of the other parts of the program, and therefore some predefined icon classes that wrap a visual artifact into a GIcon are supplied for use: *GImageIcon, GConnectorIcon, and GShapeIcon*. Also, an abstract base implementation *AbstractGIcon* is provided to enable quick construction of new icon classes. Forming composite icons

on-the-fly is also supported. *CompoundGIcon* serves for this task.

The *GShape and Connector* interfaces define the behaviors expected from a shape and a connector, in turn. The abstract base implementations *GPolyShape and LineConnector* provide mechanisms to create polygonal shapes such as *GTriangle* (25 lines of code), and line based connectors such as *StraightLineConnector* (85 lines of code), without too much effort. This package is built on top of the Java 2D API as a stand-alone, general-use API. It can be used in any context requiring such facilities.

5.2.3 Data Types

DataType related class and interfaces are placed in the *rememex.datatype* package. A base class, *DataType*, keeps and manipulates most of the required properties for a data type. Subclasses provide additional views, if they support, and a *GTransferable* object, which is capable of transferring data of that data type. Currently, the predefined data type classes in the system are *PlainText, RTFDataType, HTMLDataType, and ImageDataType.* Each of these classes has a corresponding *GTransferable* object in the *rememex.datatransfer* package. Additional custom data types can be added into the system by providing a type and a transferable (i.e. a data encoder/decoder) for the type, plus alternative views, if any. However, it should be noticed that, transferring data of any custom data type won't be possible until the bug in the Java Data Transfer API mentioned above get fixed.

Chapter 6

Conclusion

This thesis presents the framework and the base implementation of a software tool, called ReMemex, which provides an information mapping environment where concepts/objects and the inter-relationships between them are visualized by using "node-link" approach. The idea of ReMemex is based on several theories such as constructivism, semantic networking, and the schemata theory of mind. ReMemex also addresses some facts such as the demand for tools to clear the clattered desktops, or the meaningful knowledge construction skills that Knowledge Age requires from knowledge workers.

ReMemex provides the seamless mechanisms, DnD and cut-copy-paste, to insert information from a variety of sources to the workspace as nodes. By specifying how the nodes inter-relate (i.e. by creating meaningful links), users are engaged in a process of understanding the domain while organizing their information landscapes.

ReMemex offers powerful grouping options, which enable users to view the domain at any granularity. Group-nodes in ReMemex can be in one of the two modes: open and closed. The open mode corresponds to clustering where as the closed modes corresponds to abstraction.

As a result of the MVC based design, the maps and the nodes in ReMemex are capable of providing multiple views, a desirable feature in terms of analyzing information landscapes from different perspectives. The miniature view of

the maps and the thumbnail view of the nodes of image type are examples demonstrating this capability.

All stages of design and implementation of ReMemex has been carried with extensibility, flexibility and reusability concerns in mind. In this sense the resulting API can be regarded as being successful. There are many ready-to-extend parts in ReMemex. For example, one can write an alternative view for a particular node or data type and plug it into the system by adding just a single line to the source code of the class representing that node or data type. In the case of plugging additional action classes, editing or even recompilation of existing classes is not needed.

Not all of the features of ReMemex have been implemented. The major parts that remain unimplemented are the collaboration and exporting-reporting subsystems. There are several reasons for this. First of all, we had concentrated on providing a solid, detailed base for the system rather than rapid, abstract prototyping of major parts. Secondly, the learning curve of some of the APIs was long. Finally, some of the bugs within the language took considerable time to overcome. Due to these time consuming situations, we could not manage the complete implementation within the time frame of this thesis.

Therefore, to obvious future works are completing the unimplemented parts of ReMemex, and evaluating it. Some missing parts are rather easy and direct to complete, such as user interfaces for editing styles, node types and link types, additional data type classes, alternative view classes, and perhaps the reporting and filtering features. The evaluation of ReMemex might be done in two different settings: As a tool for general use and as a tool for educational use.

Bibliography

[BRG95] M. L. G. Shaw B. R. Gaines. Concept maps as hypermedia components. *International Journal of Human-Computer Studies*, 1995.

[Bus45] Vannevar Bush. As we may think. *Atlantic Monthly*, July 1945. http://www.ps.uni-sb.de/ duchier/pub/vbush/vbush-all.shtml.

[BW99] P. Bishop and N. Warren. Lazy instantiation. *JavaWorld*, February 1999. http://www.javaworld.com/javaworld/javatips/jw-javatip67.html.

[CJLM98] A. A. Carr, D. H. Jonassen, M. E. Litzinger, and R. M. Marra. Good ideas to foment educational revolution: The role of systemic change in advancing situated learning, constrcutivism, and feminist pedagogy. *Educational Technology*, Jan-Feb 1998.

[DC96] T. M. Duffy and D. J. Cunningham. *Handbook of Research for Educational Communications and Technology*, chapter Constructivism: Impliations for the Design and Delivery of Instruction. Macmillan, 1996. D.H. Jonassen (Ed.).

[DSL96] C. Dede, M. C. Salzman, and R. B. Loftin. Sciencespace: Virtual realities for learning complex and abstract scientific concepts. In *Virtual Reality Annual International Symposium*, pages 246–252, Los Alamitos, 1996. IEEE Computer Society.

[Duc97] P. Duchastel. A web-based model for university instruction. *Journal of Educational Technology Systems*, 25(3):221–228, 1997.

[Fow98] A. Fowler. A swing architecture overview. Technical report, Sun MicroSystems, 1998.

http://java.sun.com/products/jfc/tsc/archive/what_is_arch/swing-arch/swing-arch.html.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.

[Gra96] R. S. Grabinger. *Handbook of Research for Educational Communications and Technology*, chapter Rich Environments for Active Learning. Macmillan, 1996. D.H. Jonassen (Ed.).

[Gre98] T. Greening. Building the constructivist toolbox: An exploration of cognitive technologies. *Educational Technology*, March-April 1998.

[HHH+96] M. J. Hannafin, K. M. Hannafin, S. R. Hooper, L. P. Rieber, and A. S. Kini. *Handbook of Research for Educational Communications and Technology*, chapter Research on and Research with Emerging Tehnologies. Macmillan, 1996. D.H. Jonassen (Ed.).

[Hwa98] G. Hwang. A tutoring strategy supporting system for distance learning on computer networks. *IEEE Transactions on Education*, 41(4), November 1998.

[JM94] D. H. Jonassen and R. M. Marra. Concept mapping and other formalisms as mindtools for representing knowledge. *Alt-J: Association for Learning Technology Journal*, 1994.

[Jon98] D.H. Jonassen. *Instructional-design Theories and Models*, chapter Designing Constructivist Learning Environments. Lawrence Erlbaum Associates, 1st edition, 1998. C. M. Reigeluth (Ed.).

[Koy97] A. K. Koy. Computer aided thinking. In *Proceedings of the 7th International Conference on Thinking*, Singapore, June 1997.

[KS96] W. R. Klemm and J. R. Snell. Enriching computer-mediated group learning by coupling constructivism with collaborative learning. *Journal of Instructional Science and Technology*, March 1996.

[LAB+] P.B. Lawhead, E. Alpert, C.G. Bland, L. Carswell, D. Cizmar, J. DeWitt, M. Dumitru, E.R. Fahreus, and K. Scott. The web and distance learning: what is appropriate and what is not. In *Report of the ITiCSE'97 Working Group on the Web and Distance Learning.*

[LTMW98] J. Laffey, T. Tupper, D. Musser, and J. Wedman. A computer-mediated support system for project-based learning. *Edutational Technology Research and Development*, 46(1):73–86, 1998.

[MCKE98] T. A. Mikropoulos, A. Chalkidis, A. Katsikis, and A. Emvalotis. Students' attitude toward educational virtual environments. *Education and Information Technologies*, 3(2):137–148, June 1998.

[McL97] Cathrine McLoughlin. Visual thinking and telepedagogy. ASCILITE'97 Conference, December 1997.

[McM] T. F. McManus. Delivering instruction on the world wide web. University of Texas at Austin.

[Mor99a] J. L. Morrison. Higher education in 2010: An interview with rodney l. everhart. The Technology Source, VISION, November/December 1999. http://horizon.unc.edu/TS/vision.

[Mor99b] J. L. Morrison. Reevaluating the basics: An interview with judith v. boettcher about the future of education. The Technology Source, VISION, September/October 1999. http://horizon.unc.edu/TS/vision/1999-09.asp.

[NBJP+97] T. Naps, J. Bergin, R. Jimenez-Peris, M. F. Mcnally, M. Patino-Martinez, V. K. Proulx, and J. Tarhio. Using the www as the delivery mechanism for interactive, visualization-based instructional modules. Technical report, ITICSE Working Group on Visualization, 1997.

[Ozh97] Ozan Ozhan. Wiz: Dynamic web-based course presentation system. Master's thesis, Bilkent University, Department of Computer Engineering and Information Science, 1997.

[Plo97] Eric Plotnick. Concept mapping: A graphical system for understanding the relationship between concepts. Technical report, ERIC Digest, June 1997.

[RAL98] T. C. Reeves, J. M. Affolter, and C. Lo. The implementation and effects of cognitive tools in environmental literacy cources. Presented at the Annual Meeting of the American Research Association, April 1998. Project RELATE.

[Ree98] T. C. Reeves. A model of the effective dimensions of interactive learning on the worl wide web. Technical report, Instructional Technology, The University of Georgia, 1998.

[Res98] M. Resnick. Technologies for lifelong kindergarten. *Educational Technology Research and Development*, 46(4), 1998.

[RH94] W. Reader and N. Hammond. Computer-based tools to support learning from hypertext: Concept mapping tools and beyond. *Computer Educational Journal*, 22(1/2):99–106, 1994.

[Ric98] T. Richards. The emergence of open standards for learning technology. Technical report, Macromedia Interactive Learning Division, September 1998.

[RJ96] T. C. Reeves and D. H. Jonassen. *Handbook of Research for Educational Communications and Technology*, chapter Learning with Technology: Using Computers as Cognitive Tools. Macmillan, 1996. D.H. Jonassen (Ed.).

[RJM+99] M. Roussos, A. Johnson, T. Moher, C. Vasilakis, and C. Barnes. Learning and building together in an immersive virtual world. *PRESENCE*, 8(3):247–263, June 1999. MIT Press.

[RKSK98] J. Roschelle, J. Kaput, W. Stroup, and T. M. Kahn. Scaleable integration of educational software: Exploring the promise of component architectures. *Journal of Interactive Media in Education*, 1998.

[Sch98] R. C. Schank. Horses for courses. *Communications of the ACM*, 41(7):23–25, July 1998.

[SDD99] S. J. Sloffer, B. Dueber, and T. M. Duffy. Using asynchronous conferencing to promote critical thinking: Two implementations in higher education. In *Proceedings of the 32nd Hawaii Int. Conf. on System Sciences*, 1999.

[SNR+] S. K. Singhal, B. Q. Nguyen, R. Redpath, M. Fraenkel, and J. Nguyen. Building high-performance applications and servers in java: An experiential study. Technical report, IBM T.J. Watson Research Center and IBM Software Solutions.

[SP96] V. J. Shute and J. Psotka. *Handbook of Research for Educational Communications and Technology*, chapter Intelligent Tutoring Systems: Past, Present, and Future. Macmillan, 1996. D.H. Jonassen (Ed.).

[Squ99] D. Squires. Educational software for constructivist learning environments: Subversive use and volatile design. *Educational Technology*, May-June 1999.

[STW97] D. D. Suthers, E. E. Toth, and A. Weiner. An integrated approach to implementing collaborative inquiry in the classroom. In *Computer Supported Collaborative Learning*, University of Toronto, Dec 10-14 1997.

[TH99] B. Trilling and P. Hood. Learning, technology, and education reform in the knowledge age or "we're wired, webbed, and windowed, now what?". *Educational Technology*, May-June 1999.

[Wes99] W. Westera. Paradoxes in open, networked learning environments: Toward a paradigm shift. *Educational Technology*, Jan-Feb 1999.

[WS96] W. Winn and D. Snyder. *Handbook of Research for Educational Communications and Technology*, chapter Cognitive Perspectives in Psychology. Macmillan, 1996. D.H. Jonassen (Ed.).

[WT] D. R. Ward and E. L. Tiessen. Adding educational value to the web: Active learning with alivepages. *Educational Technology*, 37(5):22–31.