The University of Alberta

# Requirements for a courseware preparation system and implementation of an editor/formatter

by

YIM, Hoi Keung

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring, 1986

ISBN 0-315-30236-4

# THE UNIVERSITY OF ALBERTA

## RELEASE FORM

NAME OF AUTHOR: YIM, Hoi Keung

TITLE OF THESIS: Requirements for a courseware preparation system and implementation of an editor/formatter

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1986

(Signed) [illegible]

Permanent Address:
5-13, Tsat Tse Mui Road,
Cheung Fai Kok,
8/F Block "C",
North Point,
Hong Kong.

Dated [illegible]

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Requirements for a courseware preparation system and implementation of an editor/formatter** submitted by **YIM, Hoi Keung** in partial fulfillment of the requirements for the degree of **Master of Science.**

[illegible] ..........................................
Supervisor

[illegible] ..........................................

[illegible] ..........................................

[illegible] ..........................................

To my family

## ABSTRACT

Increasing demands for quality of and accessibility to education has stimulated research in technologically-mediated instruction in the past few decades. This has led to the notion of courseware and hence to the need for courseware preparation systems. This thesis investigates the requirements of a courseware preparation system and presents a preliminary design together with a high-level description of courseware materials. A document editor, which is part of the authoring component, capable of editing/formatting a document with text, figures and equations is designed and implemented. The structures representing the document content are compatible with the corresponding high-level description of courseware materials.

# ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my supervisor, Prof. W. W. Armstronge, for his guidance, assistance, criticisms and encouragement throughout the research and the preparation of this thesis.

I am grateful for the careful reading and useful comments provided by the members of my examination committee, Drs. E. W. Romaniuk, M. W. Green and Mr. W. Adams.

Financial support from NSERC and the Department of Computing Science in the form of teaching and research assistantships are gratefully acknowledged.

Special thanks to all my friends, especially H. C. Chan, P. Lau and S. Pun, for their encouragements during my stay here in Edmonton.

## Table of Contents

# List of Figures

# Chapter 1

# Introduction

High quality, accessible education is today recognized as an essential factor in the success of individuals and in the economic well-being of industrialized nations. The pervasive impact of computing and other technologies means that a higher standard of education and training must be achieved to prepare young and older persons alike for increased demands and opportunities in the workplace.

However, with limited budgets for education, and increasingly costly traditional instructional techniques, it is very difficult, if not impossible, to meet the existing demand for quality and accessibility. This has prompted researchers to explore the possibility of more cost-effective instructional methods which would attain equivalent or better results at reduced cost. During the past two decades, much research has been done in exploring computing technology as a means of facilitating learning in particular areas of education and training such as military training, training of sales personnel, and training of equipment operators and repair persons. This has led to the notion of courseware and hence to the need for courseware preparation systems.

Depending on the context of application, the meaning of the term "courseware" varies. There is not a universally accepted definition. For the purpose of the present thesis, we will employ Bunderson's definition[Bun81] since the scope and meaning implied more appropriately reflect our interest:

> "Courseware is an economically replicable and easily portable package which, when used in combination with a technologically mediated instructional delivery system, is capable of mediating training and performance improvement."

A Courseware preparation system is a collection of software tools that facilitate the creation, organization and modification of courseware material. It is distinguished from delivery system software, which controls interactions between a student and a computer[Bun81].

In a piece of courseware, there are two types of information stored: content and interaction. Content refers to the knowledge or materials to be learned. Interaction information controls the dialogue between student and computer. Once they are stored in digital form, economical replication and centralized updating become possible. It is the capability of replicating interaction economically that revolutionizes the instructional procedure. We can deliver instruction to students instead of sending students to instruction. Moreover, with the interaction information stored, it is possible to individualize instruction. Students can proceed at their own pace and convenience.

It is also important that existing materials can be modified easily to incorporate any change without major reorganization. A digital format of storage means that updating can build upon good materials while not diminishing in any way the fidelity of reproduction. If we store both content and interaction as data in a digital format, it is only necessary to edit the data or change the format should modification be needed. The ease of modification and portability between machines depends on the level at which content and interaction are described.

The work attempted in this thesis concerns the study and analysis of the requirements of a courseware preparation system. A preliminary design is proposed together with a high-level description of courseware and the design and implementation of a document editor which is capable of automatic formatting of a document with some text, figures and equations.

## 1.1. Organization of the thesis

Chapter two presents a review of existing courseware preparation facilities. In chapter three, the requirements of a courseware preparation system are discussed. Chapter four provides a preliminary design of the courseware preparation system together with a discussion of various components associated with it. A high-level description of courseware materials is also presented. In chapter five, the detailed design of a document editor, which is part of the authoring component of the overall design presented in chapter four, will be described. Chapter six reports the implementation of the document editor. Chapter seven provides the conclusions and suggestions for further work.

# Chapter 2

## Review of existing courseware preparation facilities

This chapter briefly reviews existing courseware preparation facilities. The main objective of the review is to identify existing approaches to courseware authoring and the related issues. It also lays the foundation for the analysis and discussion of the requirements for a courseware preparation system presented in the next chapter.

The materials presented here are mainly based on research articles by Greg Kearsley[Kea82], Richard E. Pogue[Pog80], Luis Osin[Osi76], Michael W. Allen[All84], Stanley Voyce[Voy82] and Jack Schwartz[Sch83], books edited by Harold F. O'Neil, JR [Bun81,ONe79], and the dissertation by Romaniuk[Rom70]. The above list is not meant to be exhaustive; other references will be quoted elsewhere as appropriate.

### 2.1. Authoring languages

Just as simulation languages facilitate programming of simulation models, authoring languages (often called author languages) represent a family of special purpose application languages which facilitate the writing of instructional programs[Kea82]. The first identifiable authoring language was TIP (Translator for Interactive Programs) developed for an IBM 650 around 1960[Kea82]. Before then, computer-based instructional materials were written in general purpose programming languages such as FORTRAN. TIP later evolved into the COURSEWRITER language. Since then, many authoring languages have been developed and have emerged into the market. These languages can be classified into system specific and system independent languages[Kea82]. System specific languages were developed for and operated on a particular family of machines. For example, COURSEWRITER was developed for IBM machines[Kea82,Voy82] and TUTOR was developed at the University of Illinois for the PLATO system[Sch83]. System independent languages such as PLANIT (Programmed LANguage for Interactive Teaching)[Fei68] and NATAL

(NATional Author Language)[Wes77] were developed to improve transportability of courseware materials. NATAL was developed in Canada with the major objective of establishing a standard authoring language. Examples of recent attempts to address the transportability issue of courseware written in authoring languages are the multilingual-interpreter system proposed by Stanley Voyce[Voy82] and the definition and implementation of microNATAL[HBF85].

## 2.2. Authoring systems

Authoring systems represent a high level interface intended to minimize or eliminate the need for learning or using a programming language during the courseware authoring process[Kea82,Pog80]. The time and expense required to develop courseware using authoring languages, estimated at 50-200 hours of development per hour of instruction[Kea82,Osi76,Voy82], has stimulated researchers to seek alternatives. Around 1970, the first authoring systems started to appear. VAULT (Versatile AUthoring Language for Teachers) was an authoring system written in PL/1, implemented on the IBM System 360, Model 67[Rom70]. It produced COURSEWRITER code which could then be assembled on the IBM 1500 system. VAULT introduced the important concept of separating course content and instructional logic during the authoring process. It was divided into a logic division and a data division. A different set of verbs (action words) were used in each division. Teachers or subject matter specialists started by selecting a suitable programmed logic (strategy) from a pool of available pre-programmed logics prepared by instructional specialists which formed the basic framework for entering content via the data division. A similar approach was employed by IMPACT in which a generalized Instructional Decision Model, IDM, incorporated most of the logic of a course. The IDM contains rules that decide the text to be presented next based on certain factors related to the performance of the student[Dow74].

Authoring systems can basically be divided into three categories: macro-based, form-driven and prompting[Kea82]. In a macro-based system, authors are provided with a small set of high level commands (macro routines) that are similar in meaning to the intended actions such as IF...THEN, DISPLAY, ERASE, SHOW, etc., in VAULT[Rom70]. Another example of a macro-based system is the BOOK system[OtT81] developed in Japan, which also possesses built-in information retrieval capabilities. In a form-driven authoring system, authors are provided with online or offline forms to fill in the information required to author a course. Examples in this category are the TICCIT (Time-Shared, Interactive, Computer-Controlled Information Television) system[Dow74,MSF80] and MONIFORMs[Kea82] developed for the PLATO system. In a prompting system (sometimes referred to as conversational type system[Dow74]), authors are prompted for the information necessary to create the courseware materials. COURSEMAKER[Dea78] which produces COURSEWRITER III code and an authoring system developed at the Medical College of Georgia[Rog80] are examples in this category. In the system developed at the Medical College of Georgia, an instructional strategy is created by defining a frame structure which specifies the type of content to be entered into that frame. These frame structures are stored in table form so that new instructional strategies can be added by adding entries to the table.

## 2.3. Other approaches

The systems mentioned above represent the frame oriented approach[Kea82,Osi76] towards courseware authoring. This is also the most prevalent approach and represents the majority of courseware produced. For courseware materials produced using this approach, the decision of what to present next involves mainly evaluation of a logical expression, without regards about the contents of the materials to be presented[Osi76]. Another approach is the Intelligent Computer-Assisted Instruction (ICAI) approach in which the system stores structured information about the subject materials in the form of semantic nets[RoP83], production rules[KoP76], procedural knowledge[RoP83]. A review of recent developments using these approaches has been given by Franklin and Ok-choon[RoP83]. SOPHIE (SOPHisticated Instructional Environment)[BBB75] and BIP (Basic Instructional Program)[BBA76] are examples. However, due to the difficulty in building the knowledge bases required and the complexity of the programs involved, it is not as popular as the frame-oriented approach.

The IRO-CAI (Information-Retrieval Oriented CAI)[Osi76] approach suggested by L. Osin is somewhat midway between frame-oriented and ICAI approaches. In an IRO-CAI system SMITH (Self-directed Mixed Initiative Tutorial Helper)[Osi76], developed by Osin, special purpose algorithms were used to explore minimal content information provided by the course author for controlling the presentation of materials to students.

## 2.4. Summary

In this chapter, we have briefly reviewed the development of courseware preparation facilities in the past few decades. Several main factors have influenced the development of such facilities: transportability/sharability of the courseware materials, ease of use, time required to produce the materials and the quality of the end product. While the main objective of an authoring system is to ease the authoring process and hence minimize the time required to produce instructional materials, many authors have complained about the limitations imposed. The main reason for the limitations is that features and functionalities are often compromised for ease of use. An authoring system eases the life of the authors by providing defaults[Kea82]. However, defaults can never exhaust the demands for precision which arise under various circumstances. Thus, what we really need is a system that provides defaults (macros, forms or prompts) for common features and activities while allowing the leeway for precise specifications of detailed individual needs.

# Chapter 3

# Requirements for a courseware development system

After reviewing the development of courseware preparation facilities, the next step is to study and analyze the requirements for a courseware preparation system. As mentioned in chapter one, a courseware preparation system is a collection of software tools that interact with authors to produce, organize and modify courseware materials. In formulating the requirements of such a system, we will consider the following aspects: the user-interface, the courseware produced and the design and implementation of the system.

## 3.1. User interface

Computer time and storage have traditionally been considered as scarce resources. Thus, storage and time efficiencies were often the dominant factors during software-development process. Nowadays, with the continual drop in hardware cost and the advent of personal micro-computer technology, we can afford to emphasize user productivity and satisfaction with little additional cost[FoD83].

In order to study the desirable features of the user interface component of an authoring system, we need to identify its potential users. The potential users of an authoring system are teachers, faculty members or experts in specific subject areas. They are familiar with terms and concepts related to their own fields of study, but might not and should not be required to know those related to computers and computer science. Moreover, maintaining competency in their own areas of study and fulfilling various professional responsibilities already consume most of their time and energy, and there is little left for exploring and mastering an authoring tool. Therefore, the user interface should be designed in such a way that is easy to learn and simple to operate. The user-computer dialogue should be natural to the users. This might be accomplished by mapping the interaction with the computer to their usual working environment, as is done via the desktop concept of the Macintosh microcomputer.

However, power and flexibility should not be sacrificed for the sake of ease of use. The users should be able to express whatever is essential to the system during the authoring process. Whenever possible, available choices and options should be displayed so that users can always react correctly and confidently. At the same time this reduces the burden of memorizing system commands.

In order to avoid panic and frustration, the system should also provide appropriate feedback to the author informing him/her about the current status of the courseware or the authoring process, and to allow undoing mistakes. Moreover, for actions which can cause irreversible catastrophic effects such as deleting some materials, the system should ask for confirmation before performing them. However, mechanisms should also exist to allow suppression of some confirmation requirements for expert users who would find it annoying to have to carry out these steps every time. In general, the system should become less verbose and more streamlined as the user learns how to operate it.

As mentioned before, courseware authors seldom have the time and inclination to learn programming and programming languages. Furthermore, their training and duties are concentrated mostly on aspects that are related to educational subject matter. Therefore, they should not be required to attend to computer related details. Thus the need for programming should be minimized during the authoring process. Instead, a more human-oriented interface should exist to assist authors in controlling access to a wide variety of resources(capabilities). This human-oriented interface should actually be employed in both courseware development and in programming: it is the result of the activity of detailed specification. Without a heavy burden of formal syntax, and with heavy reliance on meaningful description in English(or another natural language), the description of course content and interaction should be created by the author in a systematic manner.

### 3.2. Courseware

Courseware is the ultimate product of the authoring process. It controls the instructional process for students. Thus it directly affects the quality and effectiveness of the instruction that is received by students. Therefore, a study of its desirable characteristics is both desirable and necessary.

#### 3.2.1. Transportability

Courseware portability(from one machine or technology to another) is essential for its sharability. In addition to increasing the cost/effectiveness of producing computer-based curriculum, sharing of courseware allows good materials to be distributed to students across institutions. It is therefore essential that courseware materials be stored and packaged in such a way that allows transportability to other systems. Among other things, system specific elements should be minimized in courseware. Otherwise, when it is transported to other systems, modifications to the system-dependent components would be unavoidable. Depending on the degree of system dependency, such modification may involve a substantial amount of work. It is neither desirable for the creator nor for the maintainer of courseware that it involve system dependencies.

#### 3.2.2. Flexibility in organization

Depending on the nature and content, different materials may require different ways of organization. An organizational structure suitable for certain materials might not be appropriate for others. Forcing materials into an inappropriate structure will cause unnecessary overhead and degradation of author productivity and the effectiveness of the resulting materials. Thus the system should not dictate a predefined structure(such as a tree-structure) for organizing courseware materials.

However, it is helpful to have a pool of frequently used structures available as long as their use is not mandatory.

### 3.2.3. Testing and modification

Courseware authoring is a highly creative activity. Creative work is seldom the result of a one-pass process. Instead, it is usually accomplished through iterations and revisions. It is therefore desirable to allow testing, modification and keeping versions of materials at various stages. If the author cannot test his/her materials before finishing a complete product, complete perfection is required at every step during the authoring process in order that what emerges is close to what is expected. However, if the quality and effectiveness is far from expectation, the author might either choose to start the whole process again or to perform major modifications to the existing product. Both choices involve a significant cost penalty because modifying a nontrivial product which leads to some preset objectives requires a significant investment in time and energy.

For non-changeable supports such as laser-disks and compact disks (CD-ROMs), etc., ease of testing and modification to ensure the quality and correctness of the materials before resorting to these supports are even more important. Once materials are stored in these media, they are essentially unalterable except by using differential files or the equivalent in a magnetic medium. Putting materials on laser-disks, for instance, involves first recording them onto video tapes. These tapes are then sent for production of a master disk from which disks are replicated. Modification would mean retaping some of the materials and production of a new master disk.

### 3.2.4. Incremental development

The time required to create instructional materials can be expected to be reduced substantially if existing materials can be used in constructing new courses, and if a course can be built in an incremental manner such that it can be expanded by subsequent addition and integration of new materials. The cost-effectiveness of producing courseware materials will likely increase as a result of higher utilization of materials. As materials accumulate, this may imply the need for sophisticated data management facilities and the enforcement of tight security and integrity policies. However, the effort is totally justifiable because the expected improvement in efficiency and productivity would outweigh the initial investment in establishing the required facilities and the cost for subsequent maintenance. This is especially true as the volume of existing materials grows.

### 3.2.5. Extraction of materials

By extraction of materials, we mean retrieval of selected portions of course content. The facility provided by the system for such activities directly affects the efficiency and the time required to adapt the courseware for special purposes such as preparation of a set of slides representing selected subsets of materials. Courseware materials should thus be organized in such a way that a prescribed subset of materials could be easily and precisely specified and retrieved. The specification scheme should utilize properties, such as keywords, etc., related to the desired materials. There should be some support to make sure that the sequence selected is in a logical order and without gaps, which can be done by using stated prerequisites among the parts of a course.

### 3.2.6. A wide variety of student support

In addition to guided learning, a wide variety of supports should be available to students. These might include on-line consultation and printing of documents representing selected materials. This would allow students to take the initiative to learn and to be able to locate the desired references and documentation when needed.

Suitable aids should also exist to help students in gaining an overall view of the organization of courseware materials, just like the table of contents of a book, and in guiding them to the relevant materials as the index and section headings of a book do.

### 3.2.7. Individualized instruction

Different persons tend to have different styles and paces of learning. Learning at one's preferred pace and style increases his/her expected performance in achieving certain instructional objectives. Therefore, courseware materials should support individualization of instruction such that a student can proceed at his/her own pace and convenience, and the materials can be tailored to his/her preferred style of learning.

## 3.3. Hardware and software requirements

In this section, we shall discuss the desirable characteristics of the hardware and software of a courseware preparation system together with their significance and implications.

### 3.3.1. Expandability

The system should be designed and implemented in such a way that allows continual expansion. New capabilities or modules should easily be added without making fundamental changes. This is important because the introduction of new concepts and theories, and the continual drop in the price of hardware devices, such as audio-visual equipment, is approaching the reach of educational institutions, would permit future incorporation of new features to an existing system. Modularity, together with adequate documentation of the system are of fundamental importance. If a system is clearly divided into modules whose functions and operations are well documented in a clear and organized fashion, it is possible to incorporate new modules without painful surgery to the existing implementation.

### 3.3.2. Portability of the system

While portability of software systems in general is important, portability of a courseware preparation system is even more essential. First, the construction of a nontrivial system involves substantial investment. If it is only usable on a particular machine or operating system, cost-effectiveness is drastically compromised. Moreover, with today's rate of change of hardware technology, the chance that a particular machine becomes obsolete cannot be neglected. Therefore, system specific components in a courseware preparation system should be minimized and separated from those system independent modules. Secondly, if the system can easily be transported from one machine to another, it can be installed in more institutions that have access to computer facilities. This would encourage sharing of materials and ideas, which is both beneficial to research in instructional techniques and in technologically-mediated instruction.

Although complete portability is almost impossible to achieve in practice, it is a goal that is worthwhile striving for. Techniques such as emulation and metalanguages exist to help in alleviating software portability problems[Ian82], however, system implementors should not count solely on them. They should try their best to reduce and isolate system/device dependent code.

## 3.4. Summary

In this chapter, a discussion of the desirable features of a courseware preparation system has been presented. It will serve as a guideline for the design and implementation of such a system. Based on these requirements, a preliminary design of a courseware preparation system is proposed together with a high-level description of courseware materials in the next chapter.

# Chapter 4

# Preliminary design of the courseware preparation system

In this chapter, a preliminary design of the courseware preparation system will be presented. It starts with an overview which provides a global view of the whole system. Subsequent sections describe and discuss each component and the associated issues in greater detail.

## 4.1. An overview of the system

The following diagram provides an overview of the courseware preparation system in terms of various functional components and the interactions between them.

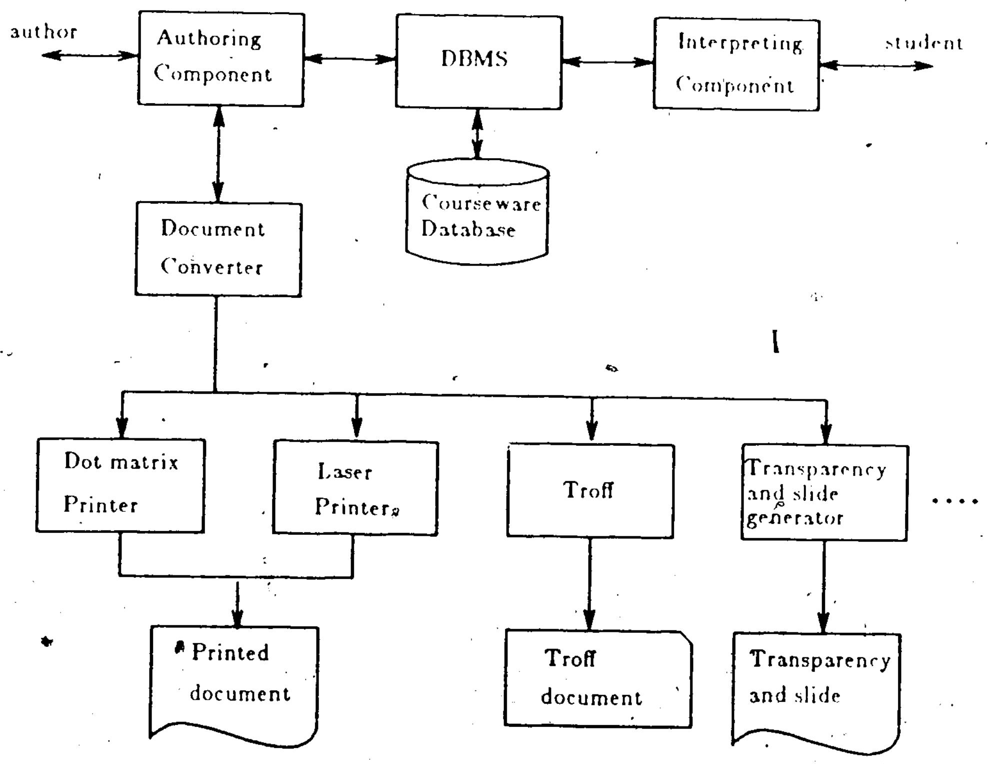


Figure 4.1 Functional components of the Courseware Preparation System

As shown in the above diagram, the system consists of four main components:

1) Authoring component,

2) Courseware database,

3) Interpreting component, and

4) Document production facilities.

The author interacts with the authoring component which is capable of generating a document (one form of delivery to students) and updating the courseware database. The interpreting component interprets (searches) the knowledge and course information stored in the courseware database in response to students' input and generates the appropriate output in various forms such as sound, graphic images or some text.

The performance of the interpreting component in conveying instructions to students depends, to a large extent, on how well the knowledge and course information are documented in the courseware database. It does not create information, it only interprets and derives information from the materials stored. A good interpreting (searching) algorithm derives the maximum information out of the stored data.

The output from the authoring component, whether destined for the database or some hardcopy support, is in the form of a high-level representation. It stores information necessary for its display instead of device specific details. Thus it is possible to automatically translate the document into input for specific printing devices or document formatting tools.

In the sections that follow, a more detailed discussion of each component will be presented. Detailed design and implementation of selected parts will be given in subsequent chapters.

## 4.2. Courseware database

The courseware DBMS is the central component of the courseware development system and use of a DBMS constitutes a key design decision. The decision is mainly based on the following considerations.

1) Courseware portability,

2) Ease of testing and modification,

3) Expandability of the system.

If courseware can be described and expressed in a machine-understandable format, it is possible to store and package courseware materials as data instead of machine executable code. This implies that a courseware package can be transported to any machine that has the appropriate interpreting software. In this way, the courseware portability issue is transformed to that of the interpreting software. However, this is transparent to course authors. The design, implementation and maintenance of interpreting software are the responsibility of computer programmers and analysts.

Since courseware materials are directly interpreted by the interpreting software, there is no need for compiling or translating the materials to machine executable codes. This may account for savings in time for testing and modification during the courseware development process.

The data driven technique employed in the design also facilitates expansion of the system. The data format of the courseware materials can be modified to compensate for any inadequacy. New capabilities can be added to the system by incorporating tools into the system that manipulate the courseware database. The interpreting component can be modified to reflect changes in the data format. However, the modification should not render old material obsolete. Any changes should be upward compatible.

### 4.2.1. A high-level description of courseware

The feasibility of the data-driven approach relies on whether courseware can be described and expressed in a format that can be interpreted in a computationally economical manner. In this section, a high-level description of courseware is attempted.

#### 4.2.1.1. Purpose and scope

Documentation is vital for communication and co-operation. Before anything can be understood, maintained or shared, it must be documented clearly in a language which, for this purpose, is both sufficient and efficient. Similarly, a high-level description of courseware is essential for its understandability, maintainability and portability.

In addition to providing a common conceptual view of courseware materials between system implementors and course authors, high-level documentation also serves as the basis for **study** and **evaluation** of the adequacy and consistency of the underlying course model. Moreover, it is considerably less expensive to iterate over versions of a design using a high-level document than when it is implemented in hardware and software. Thus, a high-level description of courseware serves the two-fold purpose of self-documentation and as the basis for future research and evaluation.

It is also important that the scope of the effort to describe courseware at a high level should include all essential factors for the presentation of the course. They include information for course management, definition of presentation units (lessons) and creation of course content. If any essential component is embedded in, say assembly language for the 6502 microprocessor or a particular dialect of BASIC, portability and ease of modification are disastrously compromised. However, the description should be elastic enough to incorporate pre-written instructional materials in various forms such as a simulation module. This suggests need for a high-level programming language (also required elsewhere) but this is beyond our scope. In particular, the user interface and interaction must be taken into account. Tools exist for this purpose[Gre85] although it is beyond the scope of this thesis to develop this aspect of courseware description.

A conceptual schema is developed and presented in the following sections. It assumes the presentation of a course as a sequential process of presenting materials and conditional branching on a student's response[GWR81]. It does not mean to be a universal schema, however, it serves as a prototype for further evaluation and enhancement.

### 4.2.1.2. The description

Figure 4.2 shows the logical organization of courseware materials. A **course** is a collection of **content elements** grouped into a set of **presentation units.** Each presentation unit corresponds to material which can be presented in a given sequence to satisfy a particular objective, for example, to teach a method of solving second-order differential equations. Associated with each presentation unit is a set of prerequisite units which indicates material required of the learner before presentation of the unit. To provide the flexibility of referencing material as a group, a presentation unit is allowed to refer to other presentation units as component parts, as well as to refer directly to content elements. Thus an entire course may be described by one presentation unit consisting of some content elements and a collection of presentation units, each of which corresponds to a topic in the course.

The **content element**, which in another context might be referred to as a frame, page or screen, is the basic unit of presentation. It is also the basic element of course content, hence the name content element. Depending on the nature of material to be presented, a content element may correspond to a block of information in the form of some text, figures or equations, etc.. Alternatively it may correspond to a decision structure for branching in an interactive situation. A content element might also be augmented with sounds for illustration and explanation. Work underway at the Media Laboratory of MIT[BaG82, Neg81] and elsewhere has revealed the importance of simultaneous perception through multiple sensory channels (sight, sound, speech, etc.) for the effectiveness of learning.

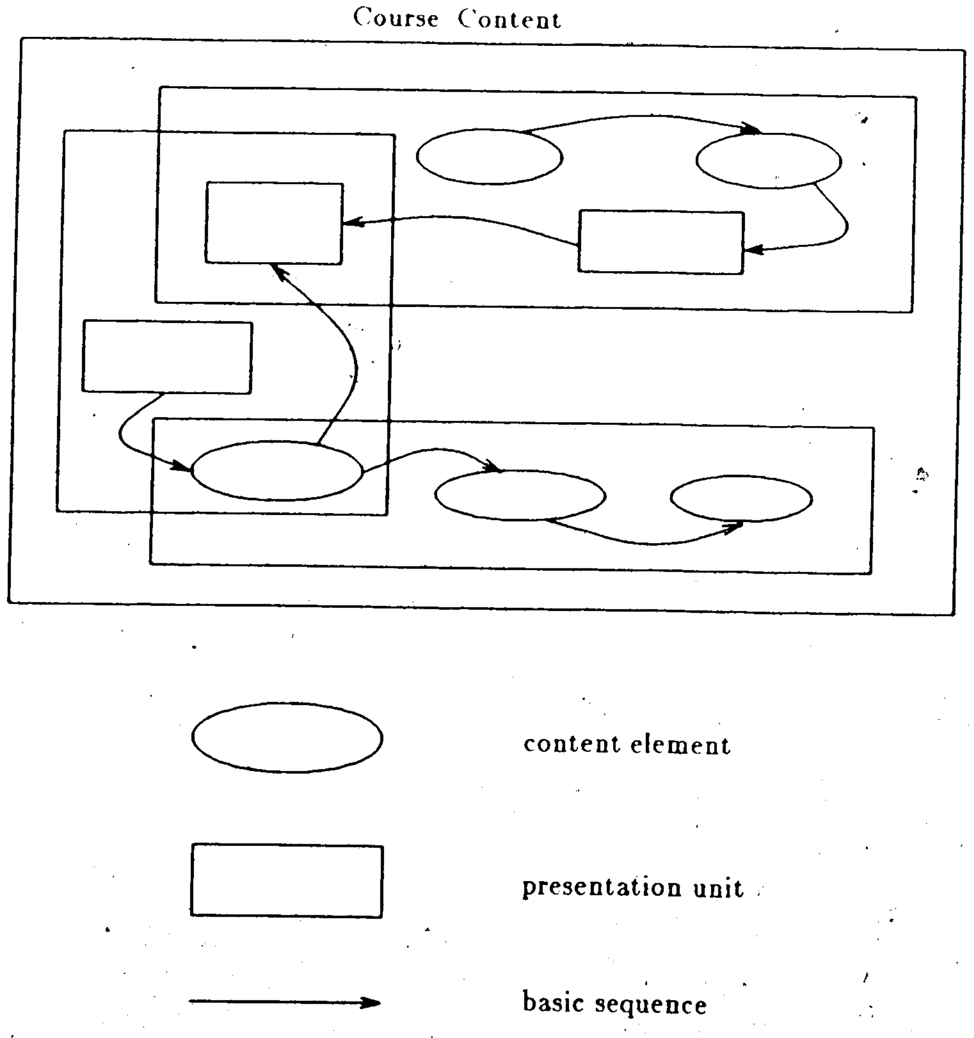


Figure 4.2 Logical organization of courseware materials

Content elements are shared among presentation units for economy of storage so that "one fact is stored in one place". This is similar to a problem encountered in formatted databases, and which is partly solved by normalization theory[Kro83], but is much more difficult here because of the broader range of types of content elements.

Presentation units are allowed to overlap. This is because we want to allow an educator to pick out subsets of the total materials to prepare subcourses for particular needs. Ideally, the system would then use its knowledge of prerequisites to aid the educator to make sure there are no gaps in the new presentation unit created.

As shown in the above description, the courseware database is a collection of objects of arbitrary complexity. The fact that an object may refer to a set or sequence of objects of arbitrary complexity implies heterogeneity of the data. To describe data of such nature precisely and accurately requires an appropriate data description language.

The Semilattice Data Model, proposed by W.W.Armstrong[Arm84], offers a set of data description primitives capable of expressing heterogeneous relationships effectively. Therefore, we shall use it to describe the course content.

The high-level description of a course would be stored as data in a (non-normalized) relational database and would be accessible via a browsing facility of the courseware editor (not yet implemented). It could also be interpreted or compiled into a more efficient form for courseware delivery. The high-level description would also be called upon in forming presentation units involving only a part of the total course.

### 4.2.1.3. The conceptual schema

In this section, a conceptual schema of the courseware database will be presented as a set of tuple-schemes. A tuple scheme is a descriptor for a composite value whose parts are named. For example a presentation unit has a part called unit_header. A value is assigned to this name "unit_header" and to all other names (objective ...), called attributes, in forming a composite value called a tuple. We have tried to give just enough detail to characterize semantically what each presentation unit must have, without limiting generality of the concept. This applies to other tuple-schemes too, however we do sometimes introduce extra attributes just to illustrate possible variability. Some tuple schemes are built-in, others must be defined by course authors for specific needs. Tuples are similar to frames in the sense of artificial intelligence. M. Green has implemented a database system based on the frame concept[Gre82]. The use of tuple schemes differs from the use of frames in that tuple schemes never allow new "slots" or attributes to be added. This permits a unique precise meaning to be attached to the tuple scheme. A tuple scheme is written as

tuple_scheme_name(attr1:domain1, attr2:domain2, . . . . ),

where tuple_scheme_name is the name of the tuple scheme with attributes attr1, attr2, ..., of types domain1, domain2, ..., respectively. The union type is written as

(tag1:domain1 | tag2:domain2 | . . . . . ),

meaning one of the types occurs, depending on the tag. A set of objects of type t is written as {t}, and a sequence of objects of type t is written as <t>.

#### 4.2.1.3.1. Tuple-schemes for logical entities

This section defines tuple-schemes for logical entities. They correspond to objects that are important for the presentation, organization and management aspects of courseware materials.

**presentation_unit**(unit_header:header, objective:<string>, pre_requisites:{header}, body:<(PU:presentation_unit|CE:content_element)>)

**meaning:**

**"Each presentation_unit tuple represents a presentation unit, the *unit_header*[1] is a unique system-wide identification of the presentation units which contains some descriptive information as well. The *body* represents the sequence of materials to be presented. It is a sequence of presentation units or content elements. The *objective* attribute is a sequence of strings**

[1] The attributes of tuples presented in this chapter appear in italics.

describing the objective of the unit. The *pre-requisites* attribute refers to a set of headers of presentation units that are pre-requisites to the present unit. The first element of the *body* might be an interaction element (described below) which displays a menu of available sub-units for selection".

**header**(name:string, title:<string>, date:string, version:Real, key_word_list:{string})

constraint:

"*name* and *version* attributes together form a unique system identification of an object described by a header tuple.".

meaning:

"A header tuple describes an object identified by the *name*, characterized by the *key_word_list* and *title*, with the *date* of its creation and a unique *version* number. The *title* is usually a condensed summary of the object described by the header and hence useful keywords may be extracted from it. The *key_word_list* contains keywords supplied by users, and those extracted from the content of the object and the *title*. ".

**content_element**(element_header:header, element:(DE:document_element|
IE:interaction_element))

meaning:

"A content_element tuple represents a content element described and identified by the *element_header* attribute as the *unit_header* attribute describes and identifies a presentation unit. The *element* attribute may correspond to a document element or an interaction element as described below.".

**document_element**(content:<(TE:text|FIG:figure|EQN:equation)>)

meaning:

"A document_element tuple describes a document element whose *content* is a sequence of text lines, figures and equations. The types (text line, figure, equation) of the *content* may be expanded to incorporate other materials such as chemical formulae.".

Note:

Use of a one-attribute tuple scheme allows for extension.

**interaction_element**(type:string, information:document_element,

interaction:{(response:string, feedback:document_element,

unit_needing_review:{header})})

meaning:

"An interaction_element tuple contains information for a particular instance of interaction. The *information* attribute consists of the information to be conveyed to student at the beginning of the interaction. The *type* attribute describes the type of interaction technique employed. It is defined by authors using tools of the authoring component and is interpreted by the interpreting component which also contains a catalogue of standard interaction techniques and device interfaces. The *interaction* attribute represents a set of information entities necessary for the interaction. The *response* attribute represents an expected response from students and the *feedback* attribute is the feedback message from the system corresponding to that response. In the case that an interaction element represents a question, the *unit_needing_review* attribute contains a set of headers of the presentation units which may need to be reviewed by students if the *response* indicates a wrong answer. Depending on the *type* of interaction technique, the interaction based on the *interaction* attribute may take various forms such as a list or a menu of response to be selected by students.".

Note:We have modeled the interaction as a process of response and feedback.".

**student**(name:string, id:string, faculty:string, major:string, year:string)

constraint:

"The attribute *id* uniquely identifies a student tuple.".

meaning:

"Each student tuple represents a student of the name, *name*, identification number, *id*, belonging to the given *faculty* and majoring in the field specified by *major* in year, *year*.".

**presentation_history**(student_id:string, unit:header,

history:<presentation_segment>)

meaning:

"A presentation_history tuple represents history of presentation of a presentation unit, *unit*, to the student of id, *student_id*. The history is represented as a sequence of presentation segments (described below) within the unit.".

**presentation_segment**(start:(object:header, time:string),

end:(object:header, time:string))

meaning:

"A presentation_segment tuple records the *start* and *end* time and object (presentation unit or content element) in a session of a presentation unit.".

#### 4.2.1.3.2. Tuple schemes for graphical entities

This section defines tuple schemes which correspond to graphical entities in the course content. They describe graphical constructs that can be produced and represented by the current design and implementation of the document editor reported in subsequent chapters. The graphics structures defined are not meant to be built-in, they constitute one package among several possibilities. The fields of various data structures in the implementation reported in chapter six might not be exactly the same as attributes of the corresponding tuple schemes presented here. However, both forms of representation can easily be converted to the other (except for header attribute in a figure or an equation tuple).

**point**(x:Real, y:Real)

meaning:

"A point tuple represents a point which is expressed as an ordered-pair of horizontal and vertical co-ordinates, *x* and *y*, respectively.".

**text_line**(the_line:string, line_info:<segment_header>)

meaning:

"A text_line tuple represents a line of text, *the_line*, described by a sequence of segment headers, *line_info*.".

**segment_header**(font_number:integer, font_size:integer, width:Real, number_of_character:integer)

meaning:

"A segment header describes the *font, font_size, width* and number of characters, *number_of_character*, in a text segment.".

**figure**(figure_header:header, height:Real, width:Real, objects:{object}) meaning:

"A figure tuple represents a figure described by the *figure_header* attribute of height, *height*, and width, *width*. The *objects* attribute refers to the set of geometrical or document objects in the figure.".

object(type:string, control_points:<point>, DocObj:(TE:text_line|EQN:equation))

meaning:

"An object tuple represents a basic geometrical object that can be described by a sequence of control points, *control_points*, or it may corresponds to a line of text or an equation (refers to by the *DocObj* attribute) at a location pointed to by the only control point in the set of control points (a singleton), *control_points*.".

equation(equation_header:header, sequence_of_boxes:box_sequence)

meaning:

"Each equation tuple represents an equation which is headed and described by the *equation_header* with the equation expression represented as a sequence of boxes described by the *sequence_of_boxes* attribute.".

Note:

Throughout the definition of an equation, no assumption has been made about the device. The actual display of an equation will derive the necessary information from the structure.

**box_sequence(box_header:box, remaining_boxes:**

**<(symbols_box|special_symbol_box)>)**

constraint:

"width of *box_header* = sum of widths of *remaining_boxes*, upper and lower height of *box_header* = maximum upper and lower height of *remaining_boxes* respectively.".

meaning:

"A box_sequence tuple represents a sequence of boxes with the *box_header* storing global information about the *remaining_boxes*.".

**box**(width:integer, local_width:integer, upper_height:integer, lower_height:integer, font_number:integer, font_size:integer)

meaning:

"A box tuple represents a data entity containing information about the font and the dimensions of a box. The *font_number* attribute describes the font and the *font_size* attribute describes the font size. The *upper_height* and *lower_height* attributes represent the heights above and below the base reference of the box. The *width* attribute represents the width of the box, taking into account the expressions associated with it, while the *local_width* attribute stores the width of the box without regard to other expressions or boxes.".

**symbols_box**(the_box:box, symbols:string, superscript:box_sequence, subscript:box_sequence, from:box_sequence, to:box_sequence)

meaning:

"A symbols_box tuple represents a box containing a string of symbols of font number and size described by the attribute, *the_box*. The *superscript, subscript, from* and *to* attributes represents the corresponding expressions of the box respectively.".

**special_symbol_box**(the_box:box, the_sym:integer, superscript:box_sequence, subscript:box_sequence, from:box_sequence, to:box_sequence)

meaning:

"A special_symbol_box tuple represents a box containing a symbol that differs in the default font size from ordinary symbols. The symbol and the variation in size depend on the interpretation of the attribute *the_sym*.".

The above description illustrates the possibility of describing courseware materials systematically by a suitable data description language such as that offered by the Semilattice Data Model. By providing a set of system-defined tuple schemes, novice users could create courseware easily following the guidance of the system. With the capability of user defined tuple schemes, experienced users can define tuple schemes to represent structures of any complexity needed.

## 4.3. Authoring component

This component is responsible for guiding the author through the authoring process. It includes tools that are relevant for the creation and modification of course materials and allows the formation of a document from selective portions of the courseware materials. These tools can basically be divided into three categories: editors, user-interface facilities and database management facilities.

The editors interact with the authors for entering and modifying course content, course organization and management information. Among them is a document editor that allows editing of course content and producing a document in the form of a high-level representation. Detailed design and implementation of this editor will be reported in the chapter that follows.

The user-interface management facilities allow definition and automatic generation of user-interfaces. The University of Alberta User Interface Management System (UIMS)[Gre85] and similar systems developed elsewhere demonstrate the feasibility of defining and representing user-interfaces at a high-level.

The database management facilities are responsible for updating the courseware database and enforcing various security, consistency and integrity constraints. A Semi-lattice DBMS is currently being developed. With its completion, tools can be built to assist authors in defining and modifying the tuple schemes through the DBMS facilities. With all these tools, a prototype system ready for experimentation will then be available.

## 4.4. Interpretation component

This component interacts with students based on knowledge and information derived from the courseware database. Among other things, it includes an interpreting (searching) algorithm that interprets the course materials in determining the actions to be performed and materials to be presented. Given a fixed amount of information available in the database, the effectiveness of the instruction delivered to students depends on the intelligence of this algorithm. In addition, it should also include facilities for retrieving and browsing through materials and for helping students in locating the necessary references.

## 4.5. Document production facilities

The document converter, together with various document processing facilities such as dot matrix printer, laser printer, troff, transparency and slide generator, etc., constitute a collection of document production facilities that allows a document to be produced in different forms. The document converter interprets the high-level representation of a document from the authoring component and converts it into input for specific printing devices or document formatting tools.

# Chapter 5

# Detailed design of the document editor

In chapter four, we have presented a preliminary design of the courseware preparation system in which the authoring component is one of the major components. In this chapter, detailed design of the document editor, which is part of the authoring component, will be described. Discussion will concentrate on the representation of document content and the associated editing operations.

## 5.1. Basic objectives

The document editor presented here allows **interactive** specification and manipulation of a document consisting of some text, figures and equations. "What you see is what you get" has been the guiding principle throughout the design process. Users are given immediate feedback about the operations just performed. Moreover, the resulting document should be compatible with the high-level description provided in chapter four. The other objectives are:

1) to serve a wide range of users.

   The editor should be usable by users ranging from the original author of the document to his/her typist or secretary with equal ease. One of the ways to achieve this objective is to require the users only to have a two-dimensional interpretation of the document.

2) to represent a document in such a way that allows operations to be performed on selected portions of a document.

   This would increase the efficiency of editing operations performed on the document.

3) to minimize the time spent in preparing a document.

4) to support editing capabilities on objects of different natures, such as text and

figures, etc., in one editor.

This saves the users time and trouble in switching to other editors to edit document content that cannot be manipulated by the current editor. Past experience by some course authors[Moh85] has shown the restrictions of limiting the types of objects that can be manipulated in one editor.

5) to minimize the need of memorizing editing commands.

This is important for user satisfaction and productivity because of the time and energy saved in memorizing and referencing volumes of manuals.

6) to enhance the transportability of the resulting document to other systems.

7) to allow easy expansion of the present system by adding modules to it without major modification.

This is significant because text, figures, and equations are by no mean an exhaustive set of possible objects in a document.

## 5.2. Document and document editor

The **document editor** is a software tool which aids the author in creating and manipulating a document **easily** and **interactively.** It also performs automatic formatting on the document. A document is a sequence of **document objects.** Document objects are represented in such a way that each of them is a separately selectable and identifiable entity, hence a group or subsequence of document objects can be formed. Each subsequence of objects contains **one** or **more** objects; thus a unified set of editing operations can be defined on a subsequence of object(s). Figure 5.1 shows the logical view of a document.

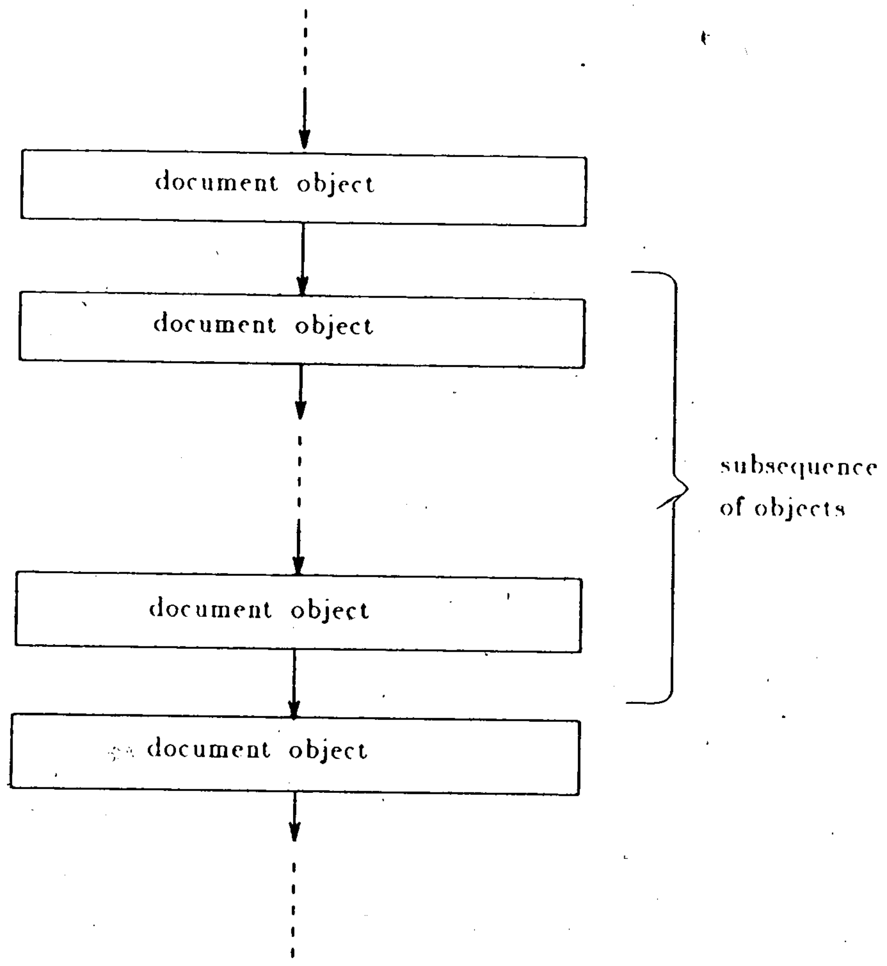


Figure 5.1 Logical view of a document

Editing operations provided by the document editor can basically be divided into two categories: individual and group. Editing operations on an individual object depend on the type of the object. For example, the operation in response to a particular event on a line of text may be very different from that of a figure. Therefore, we shall discuss editing operations on an individual object along with the description of its representation. On the other hand, editing operations on a group (subsequence) of objects are independent of the type of individual objects present in the group. They correspond to operations to be performed on the currently selected group of document objects. Basic editing operations in this category are:

1) Delete - delete the currently selected group of document objects,

2) Copy - insert the currently selected group of document objects after the object indicated by the current cursor position,

3) Move - move the currently selected group of document objects to a point following the object indicated by the current cursor position.

Since these operations are independent of the type of objects present in the group, they are transparent to changes in the types and representations of the objects.

## 5.3. Document objects

A **document object** is a separately selectable and identifiable entity present in a document. Associated with each document object is an object header containing the following information:

1) Type - represents the type of document object,

2) Height - records the height of the document object,

3) Width - stores the width of the document object,

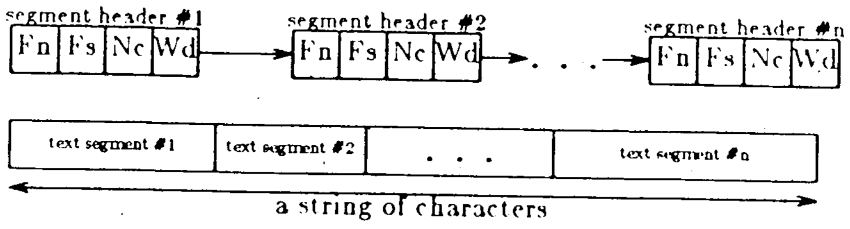
4) Default font - records the default font defined on the document object,

5) Default font size - stores the default font size of the document object,

in addition to object-specific information. In the present design and implementation, there are three types of document objects: text, figures and equations. However, it is easy to introduce new types of document objects into the system by specifying the corresponding object headers. The structures needed to represent the objects and the corresponding manipulation routines can be added easily without major modifications to the existing package.

### 5.3.1. Text: representation and operations

A line of text is composed of a sequence text segments. A text segment is a subsequence of characters in a text line which is of a different font and/or font size as compared to the characters to its immediate left and right. Figure 5.2 shows the representation of a line of text.



Wd: Width of the text segment
Nc: Number of characters within the text segment
Fs: Font size of the text segment
Fn: Font number of the text segment

Figure 5.2 Representation of a line of text

Each segment header contains the following information about the corresponding text segment.

1) Font : Fn;

2) Font size : Fs;

3) Number of characters : Nc;

4) Width of the segment : Wd.

An alternate scheme is to represent a text line directly as a sequence of text segments. However, since we cannot predict the number of characters present in each segment, we have to reserve enough space for a whole line of text for each of them. This will cause serious storage overhead if the number of text segments per line of text is large.

Editing operations on a line of text are: delete, insert and move a character or subsequence of characters. A subsequence of characters may cross text segment boundaries. Each operation may cause an update of the height and width of the corresponding text line.

### 5.3.2. Figures: representation and operations

A figure is a collection of basic geometrical objects and document objects. Each object in a figure is a separate entity described by the following attributes:

1) Type - represents the type of object in a figure,

2) A set of control points - stores the set of control points of a geometrical object or position of a document object in a figure,

3) BoundsRect - refers to the smallest rectangle enclosing the object,

4) DocObject - refers to an object in a figure other than a geometrical object

A set of control points, together with the boundary rectangle (BoundsRect), describes an instance of a geometrical object. The interpretation of the control points depends on the type of the object. They are measured relative to the top-left corner of the figure. An object is created by specifying the control points through interaction with users. Document objects (text lines and equations) in a figure are specified in the same way as document objects in a document, and are given a particular base reference stored as one point in the set of control points.

Within a figure, a subset of objects may be grouped to form a subfigure. Each subfigure consists of one or more object(s). A unified set of editing operations can thus be defined on a subfigure. Basic operations are:

1) Delete - delete the currently selected subfigure,

2) Translate - translate the currently selected subfigure to a new position,

3) Copy - translate without deleting the old subfigure.

Since objects within a figure represent separate entities, and since they are independent of each other, editing operations on any of them will not affect other objects in the figure. However, after each editing operation, the overall dimensions (height and width) of the figure might be changed. Therefore, the dimensions of the figure must be checked and updated after each editing operation or editing session.

### 5.3.3. Equations: representation and operations

Editing effects within a figure are localized, hence we only need to consider operations on a particular subfigure at a time. However, for an equation, the situation is much more complicated. The main reason for the complexity is due to the automatic formatting requirement of an equation. Editing effects on any part of an equation are not localized, they propagate to other parts of the equation and, in the worst case, to the whole equation structure. Consider the following equation,

$$\frac{a}{x^{\frac{b}{c^2}}}$$

If the user deletes the superscript, 2, of c, c will have to move **up** because it is the denominator of the fraction b/c, and this fraction itself have to move **down** because it is the superscript expression of x which itself have to move **up** because it is the denominator of the whole equation.

Observe that the updating operation on a simple equation like this one is non-trivial. Those on more complex equations will be much more complicated. Therefore, the question is, given the requirement of interactive automatic formatting of an equation, what is an appropriate structure that would represent the information necessary for its display and, at the same time, support automatic formatting of an equation in an efficient manner. Moreover, it should conform to the high-level description of an equation presented in chapter four. A possible structure is designed and presented in the following sections.

### 5.3.3.1. Representation of an equation

An **equation** consists of a collection of expressions, occupying disjoint rectangular regions, linked together by various relations such as "subscript of" and "superscript of". An **expression** is a sequence of boxes described by a **header box** which stores global information about the expression such as the type, default font and font size. An equation is represented as an expression of type "Equation".

Associated with each box (except a header box describing an equation expression), there can be four expressions:

1) a subscript expression,

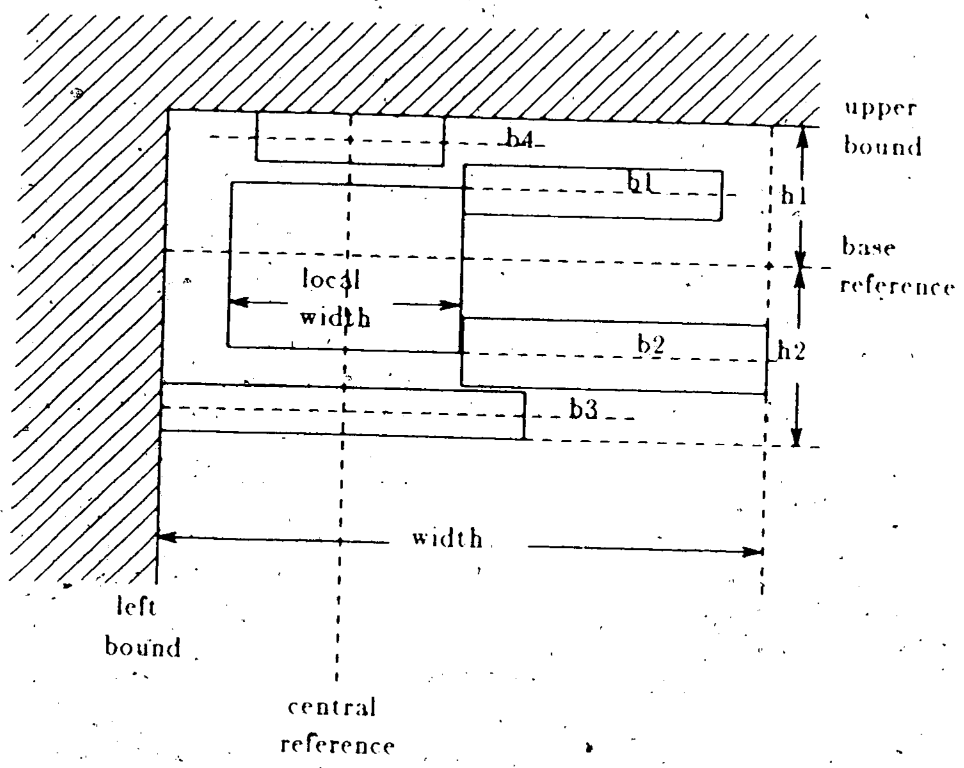
2) a superscript expression,

3) a from expression, and

4) a to expression.

Figure 5.3 shows the general structure of a box and the associated parameters. The height above and below the base reference, and the width parameters represent the corresponding dimensions of the box as they appear to the outside environment, taking into account the expressions associated with it. The local width represents the width of the box excluding the associated expressions.

A box may represent a string of symbols (a sequence of Greek letters or characters of any available font), a mathematical operator such as summation, integral, union, intersection, etc., a fraction, or it may refer to the header box of an expression. In the latter case, the local width of the box is the width of the expression it refers to. The reason for separating ordinary symbols with mathematical operators is that the default size of an operator is greater than that of an ordinary symbol. In the case of a fraction box, the numerator and denominator are represented by the to and from expressions respectively.



h1 = height above base reference line
h2 = height below base reference line
width = width of the whole box
local width = width localized to box
b1 = base reference for the superscript expression
b2 = base reference for the subscript expression
b3 = base reference for the from expression
b4 = base reference for the to expression

Figure 5.3 Logical structure of a box

Notice that in the box structure shown, there is no need to store the actual location of the base reference of any box. A box (except header boxes) always belongs to an expression which is a sequence of boxes. Boxes in the same expression refer to a common base reference which is determined by the relationship of the expression with other expressions in the equation (as shown in figure 5.3).

The fact that no positional information is stored greatly facilitates updating operations on an equation. As mentioned before, updating effects are not localized to a restricted part of an equation, they affect positioning of other parts of the equation. Since no positional information is stored, there is no need of traversing the whole equation structure to update all the positional parameters on every updating operation.

### 5.3.3.2. Updating operations on equations

An updating operation on an equation refers to an operation that would cause a change in the **structure** and **dimensions** of **a box** in an expression. Figure 5.4 shows the logical structure of an expression and the corresponding sequence of boxes.

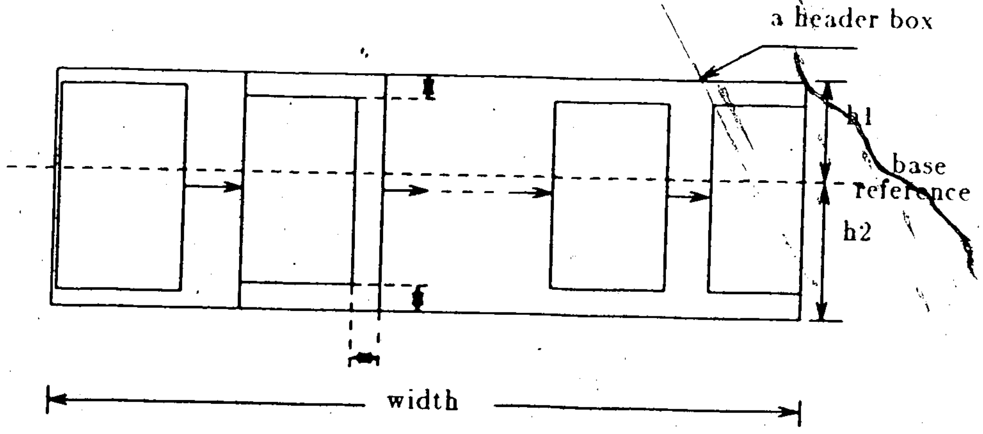


Figure 5.4 Logical structure of an expression

Observe that changes in the dimensions of a box in an expression will **not** affect those of others in the same expression. However, they may affect the overall heights (h1 and h2) and width of the expression. Assume that at any given instance, there is a current box belonging to an expression and operations always refer to this box. For each updating operation, we perform the following steps:

1) if the operation is box insertion, insert the box after the current box. GOTO step 5.

2) if the operation is box deletion, delete the current box. GOTO step 5.

3) update h1 or h2 of the current box;

4) update the width and local width of the current box;

5) check if the overall dimensions of the expression containing the current box are affected;

6) if not, RETURN;

7) update the corresponding parameters of the header box of the expression;

8) check if the expression relates to a box by relations such as "subscript to" or "superscript to";

9) if not, RETURN;

10) check the effect of change in dimensions of the expression on the box it relates to;

11) set the box as the current box;

12) GOTO step (3).

In step (10), checking the effect of a change in the dimensions of an expression on the box related to it involves checking whether the parameters h1, h2 and the overall width of the box are affected or not (please refer to figure 5.3). For a superscript expression, a change in h2 will not affect the height above the base reference, h1, of the box. Similarly, a change in h1 of a subscript expression will not affect the height below the base reference, h2, of the box. However, a change in h1 and/or h2 of a to or from expression affects the height above or below the base reference of the box respectively.

## 5.4. Summary

In this chapter, we have presented the objectives and design of a document editor in terms of the representations and manipulation of document content. We have deliberately avoided specification of how it should be implemented for portability of the design itself. In the next chapter, one of the possible implementations is reported.

# Chapter 6

## Implementation of the document editor

In this chapter, the implementation of the document editor will be described. The main objective of the implementation is to test the validity of the design presented in the previous chapter and to provide a tool as the starting point towards the ultimate system.

### 6.1. Implementation environment

The document editor has been implemented for the Apple Macintosh microcomputer, using SUMacC[2] on VAX[3] 11/780 running UNIX[4] 4.2 BSD. SUMacC is a software package which, together with the cc68 compiler, allows a Macintosh application written in the C programming language, with calls to various Macintosh toolbox managers and graphics routines, to be converted into machine code that can be executed directly in a Macintosh microcomputer.

There are a few reasons for choosing Macintosh as the development medium. First, the desktop concept of Macintosh maps close to the usual working environment of potential users (teachers, students). Users manipulate objects on the screen as they manage files and folders on their desk top. This naturalness of user interaction minimizes the initial hurdle of getting used to the system and the need for future consultation of system manuals. The resulting increase in user satisfaction and productivity will increase the chance of acceptance of tools and materials built for it. Secondly, various toolbox facilities available in ROM, especially the pull-down menu and window management facilities, help to create a multi-window menu-driven system. In cases where pull-down menus are not appropriate, the window management facility and others such as the icon definition facility allow alternatives such as pop-up menus.

[2] SUMacC was developed at Stanford University by Bill Croft.
[3] VAX is a trademark of Digital Equipment Corporation.
[4] UNIX is a trademark of AT & T BELL Laboratories.

Thirdly, the powerful M68000 processor together with a versatile graphics package (Quickdraw) allow more sophisticated applications to be written. Furthermore, the treatment of characters as bit-maps and the availability of various font definition facilities allow easy addition and manipulation of fonts. This greatly increases the variety of materials such as equations, chemical formulae and foreign languages which can be included in courseware materials.

Last but not the least, the price of a Macintosh is within an affordable range of educational institutions. Thus, it is suitable to be used as a creation and delivery medium. Moreover, a lot of effort has been devoted by the research community (especially in North America) to exploring the potential of the Macintosh. Improvement and discovery of new possibilities are expected through sharing of ideas.

## 6.2. Editing environment

The editing environment of the document editor is best illustrated by the screen layout(shown in figure 6.1). The editor is basically a menu-driven system with the following major components:

1) Document window,

2) Menu bar,

3) Figure object menu,

4) Equation symbol table.

Each component represents a part of the user interface responsible for a specific subset of functions.

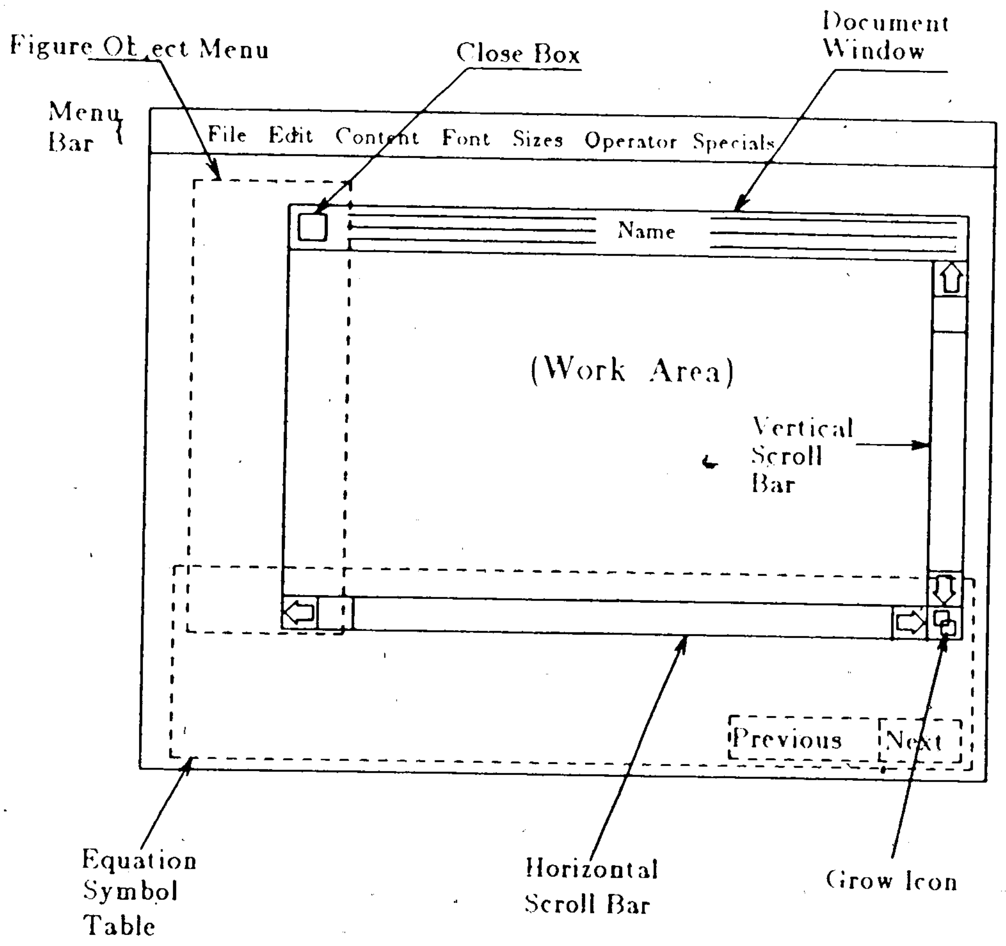


Figure 6.1 Screen layout of the document editor

### 6.2.1. The document window

The document window represents the user's view of his/her document. The Name field displays the name of the document. The vertical scroll bar allows scrolling the document upward and downward while the horizontal scroll bar permits left and right sideways scrolling. The grow icon allows shrinkage and expansion of the document window. Clicking the close box closes the document window. The present implementation handles only handle one document at a time. Therefore, closing the document window also causes exit from the document editor. Future implementations should allow several documents to viewed at one time.

Part of the document is displayed in the work area. The user enters and manipulates the document through various menus, the equation symbol table, the keyboard and the mouse. Notice from the screen layout that tools exist for the manipulation of text, figures and equations. This allows the flexibility of handling a document with varied types of document content in one editor, which may account for much saving in time compared to switching back and forth among several editors each of which handles one kind of document.

### 6.2.2. The menu bar

The menu bar displays the titles of a set of pull-down menus available in the document editor. Other menu types are stationary and pop-up. A menu groups a set of items corresponding to functions indicated by the title. For example, the File menu consists of items for file manipulation functions. The use of pull-down menus has the advantage of saving valuable screen space, however, it also increases the amount of motor activity required to initiate a certain command. Therefore, if the items in a particular menu are comparatively little used, we shall group them into a pull-down menu.

The File and Edit menus contain items for the usual file manipulation and editing commands respectively. The Font menu consists of the names of available fonts provided by the editor and the Sizes menu contains the list of font sizes supported.

The Content menu lists the types of document content supported by the current implementation. It consists of the following items:

1) Text,

2) Figure, and

3) Equation.

It also serves as an indication, both to the editor and user, of the current type of document content (text by default) being manipulated. The item corresponding to the current type is checked. If item Figure is selected, the Figure Object Menu will be displayed and the work area erased. If a figure is currently selected (by double-clicking on it), it will be displayed in the work area. Otherwise, the work area is left blank and is available for the user to enter a new figure. The newly created figure is then inserted after the cursor position just before switching to Figure mode.

When Equation is selected, the Equation Symbol Table is displayed. Together with the Operators and Specials menus and the keyboard, the user is able to enter an equation. The Operators menu consists of items corresponding to commands that would initiate the definition of a new type of expression such as fraction, subscript, superscript, square root, from and to. It can easily be expanded to incorporate new expression types such as a pile of expressions, etc.. The Specials menu contains the names of symbols whose default sizes are greater than that of an ordinary symbol, and which usually appear in an equation. In the present implementation, the default sizes of these symbols are 1.5 times that of ordinary symbols. They include integral, summation, product, union and intersection. Again, it can be expanded by adding items to the menu.

### 6.2.3. The Figure Object Menu

The Figure Object Menu contains a catalogue of objects that may appear in a figure. To enter a geometrical object into the figure, the user selects the appropriate item from the menu and specifies the control points that define the object through interaction with the editor. Other objects such as text lines and equations are entered in the same manner as if they were document objects in a document with a given base reference.

The Figure Object Menu appears when Figure is selected from the Content menu and disappears when other items (Text and Equation) are selected. The Equation Symbol Table will also appear if the user wants to enter an equation into the figure by selecting the appropriate item from the Figure Object Menu, and disappears when other items are selected. This menu can be expanded by adding items into it and the corresponding handling routines. If the items added correspond to objects other than text, equation and geometrical objects that can be characterized by a set of control points, the appropriate data structures would have to be added.

### 6.2.4. The Equation Symbol Table

The Equation Symbol Table consists of a table of symbols (mostly Greek letters and other mathematical symbols) that are not available on the keyboard. Except for the appearance, they possess the same properties, such as default size, as keyboard symbols. The users enter a symbol by clicking on the appropriate table entry as if they were typing a character on the keyboard. The Previous and Next buttons near the bottom-right corner of the table facilitate expansion of the table. A doubly-linked list of symbol tables can be constructed when more symbols are required. This table disappears when the users are not currently manipulating an equation.

In the editing environment discussed above, we have introduced three modes of operation: text, figure and equation. According to the standard user-interface guidelines for a Macintosh application, modes are to be avoided whenever possible. However, a modeless editor requires that different types of document objects be treated as homogeneous entities, which is not acceptable since different document objects have different editing/formatting requirements. Therefore, we have introduced three modes of operation, each of which corresponds to a particular type of document object. In doing so, we have tried to provide adequate indications of the current mode of operation to users (for example, display of figure object menu in figure mode, equation symbol table in equation mode and checking the appropriate item in the Content menu).

## 6.3. Internal representation of a document

A document is described by a document record and the document content is represented by a doubly-linked list of document objects (shown in figure 6.2)

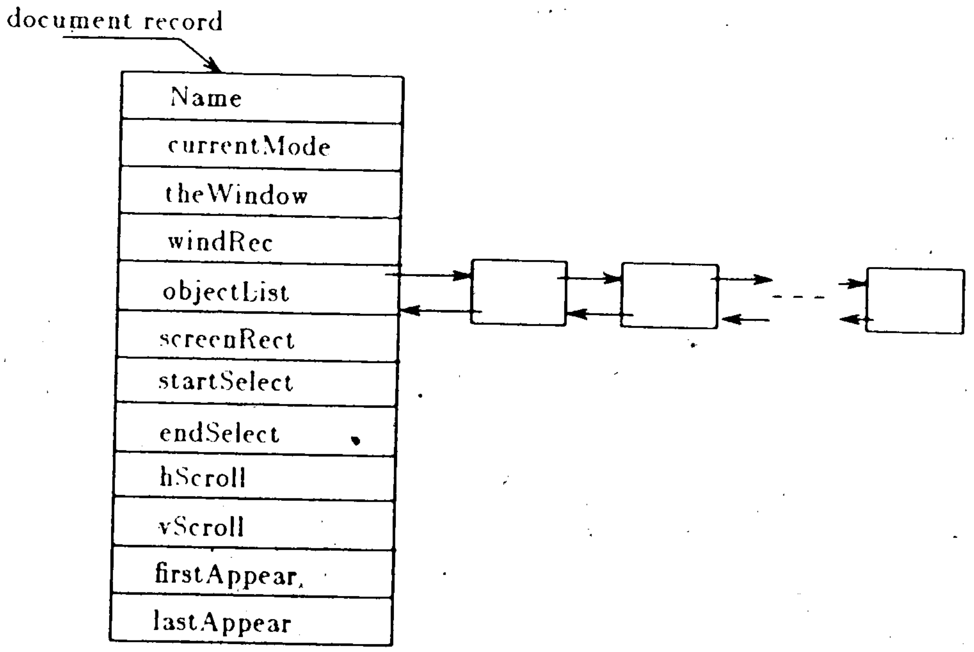


Figure 6.2 Data structure representing a document

The document record contains descriptive information about the current status and characteristics of the document . The *Name*[5] is a string of characters storing the name of the document. The *theWindow* and *windRec* fields are pointers to the graf-port of and a record containing descriptive attributes about the document window respectively. The *hScroll* and *vScroll* are control handles to the horizontal and vertical scroll bars associated with the document window. The *screenRect* field is the rectangle bounded by the document window on the Macintosh screen. The *objectList*

[5] The names of fields of data structures described in this chapter will appear in italics.

field points to the list of document objects. The *startSelect* and *endSelect* fields point to the beginning and end of the current selection range of document objects and the first and last objects appearing on the document window are indicated by the *firstAppear* and *lastAppear* fields. The *currentMode* field shows the current mode of operation (text, figure or equation).

The main reason for using a doubly-linked list, instead of a singly-linked list, is to represent the document structure in a way that facilitates vertical scrolling of the document. A document can thus be scrolled upward and downward with equal ease computationally. By storing all the information about a document in a document record, the document editor can easily be expanded to handle more than one document at a time. Each document would then occupy a separate window and would be described by a different document record. An obvious advantage of a multiple-document editor is the possibility of transferring contents among documents.

## 6.4. Internal representation of a figure

The data structure representing a figure is shown in figure 6.3. A figure record contains descriptive attributes about a figure. A singly-linked list of objects is used to represent component objects in a figure.

The *type* field indicates the type (figure) of an instance of the object. The *objList* field points to the list of objects in the figure and the currently active object is pointed at by *currentObj*. In the current implementation, grouping of objects into subfigures is not supported. Thus the editor can only handle one level of object at a time. The *height* and *width* fields store the height and width of the figure respectively. The *next* field points to the next document object and the *previous* field points to the previous document object.

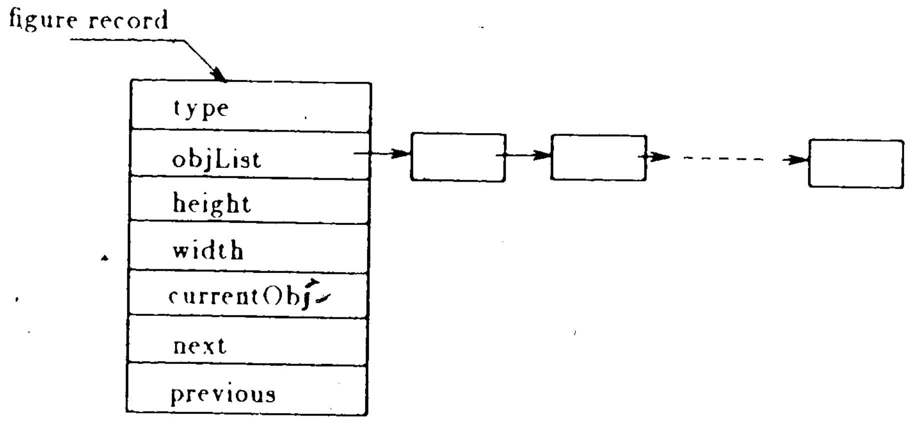


Figure 6.3 Data structure representing a figure

An object in a figure is described by a figure object record (shown in figure 6.4). The *type* field tells the type of the object bounded by the *boundsRect*. It might be a basic geometrical object describable by a list of control points and the associated the *boundsRect,* or it might be a document object (a line of text or an equation). In the latter case, the *controlPoints* field points to one point containing the horizontal displacement and the base reference of the document object pointed at by the *otherObj* field. The *next* field points to the next object in the figure.

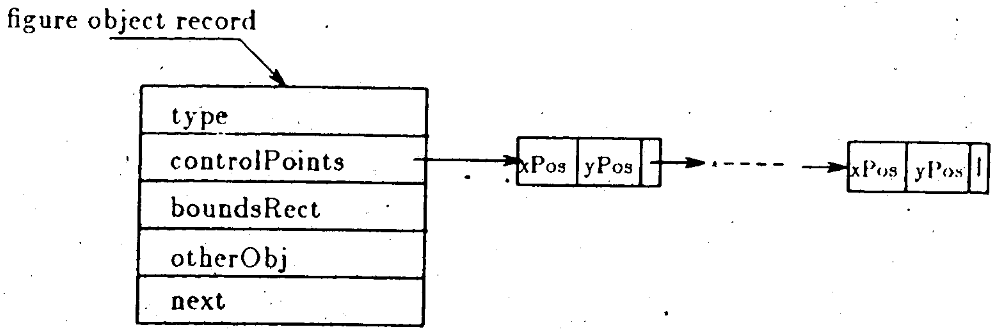


Figure 6.4 Data structure representing an object in a figure

## 6.5. Internal representation of a line of text

The data structure representing a line of text is shown in figure 6.5. A line of text is described by a text line record.

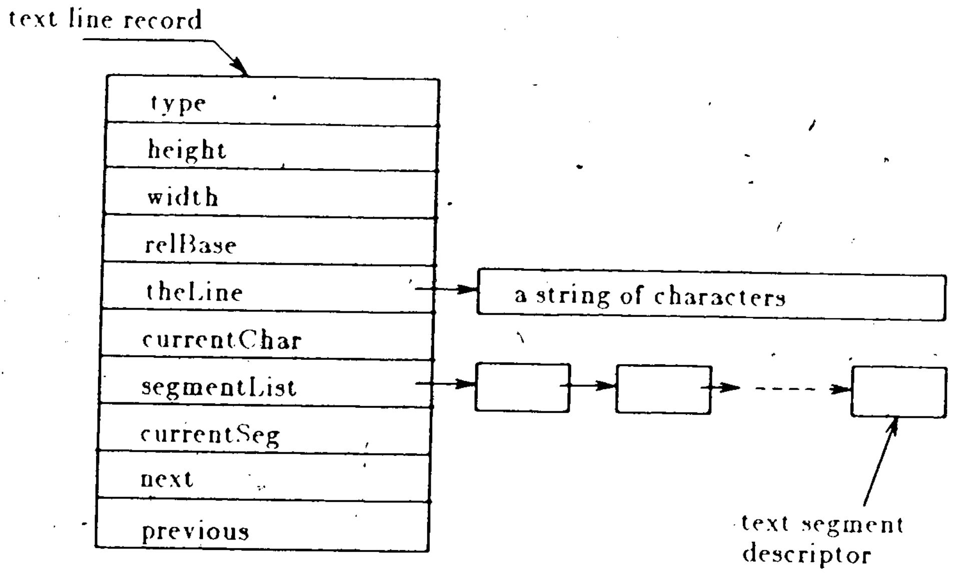


Figure 6.5 Data structure representing a line of text

The type, height, width, next and previous fields of a text line record describe a text line just as those of a figure record describe a figure. The *relBase* stores the common base reference of the text segments relative to the bottom of the previous document object. The string of characters in a text line is stored in an array pointed at by the *theLine* field and is described by a list of text segment descriptors (shown in figure 6.6). The *currentChar* field stores the index to the current character in a text segment described by the text segment descriptor pointed at by *currentSeg*. In the current implementation, we only support single character selection and operations. However, there is no conceptual difficulty in extending to multiple character selections and operations.

| |
|---|
| font |
| fontSize |
| length |
| numChar |
| next |

Figure 6.6 Record structure representing a text segment descriptor

## 6.6. Internal representation of an equation

An equation is represented as a sequence of boxes(shown in figure 6.7) headed by a box of type Equation.

| | | |
|---|---|---|
| type | info | to |
| h1 | font | |
| h2 | fontSize | superscript |
| width | localWidth | subscript |
| previous | next | from |

Figure 6.7 Record structure representing a box

The sum of h1 and h2 fields of a box of type Equation tells the height of the equation, and the width field represents its width. As usual, the next and previous fields point to the next and previous document objects respectively. The provision of the font and fontSize fields in a box is to facilitate local variation of font and font sizes, though it is not yet supported by the current implementation.

In updating an equation, we traverse the equation structure recursively in searching for the box being updated, which thus establishes a path towards that box. After performing the operation, we determine the effects caused by the operation on other parts of the equation using the algorithm described in chapter five. In the present implementation, we support arbitrary positioning of the cursor in an equation and performing single-symbol and box insertion and deletion. As mentioned in the previous chapter, expressions (hence boxes) occupy disjoint rectangular regions on the screen. A group of expressions and/or boxes can thus be formed by specifying the region bounded by the selected group. Thus editing operations can be extended to multiple levels of expressions and boxes.

# Chapter 7

# Conclusions and further work

## 7.1. Conclusions

This thesis has investigated the requirements for a courseware preparation system and various aspects related to its design and implementation. We have started by reviewing existing courseware preparation facilities which helped to identify the difficulties involved in courseware preparation, and the existing approaches to tackling these problems. The advantages and limitations associated with each of these approaches provide the guidelines for formulating the requirements. We have considered the requirements for the user interface, for the courseware and for the independence from hardware and software choices.

Based on these requirements, a preliminary design of a courseware preparation system has been proposed in which the courseware database is the central component. The author enters and modifies materials in the courseware database, produces a document that represents selected portions of the total materials via the authoring component. The interpreting component derives the necessary information from the courseware database for presentation of materials to students. Thus the feasibility of the whole design depends on whether courseware can be expressed as a database schema. This implies the need of describing courseware at a high-level by a suitable data description language. A conceptual schema, expressed in the data description facilities provided by the Semilattice Data Model, has been developed and presented.

Two basic constructs, content element and presentation unit, have been introduced for representing course content and grouping courseware materials into modular units for presentation. Information necessary for the presentation, organization and management aspects of courseware materials are described by attributes associated with the corresponding tuples. Though the description may only support a limited subset of all possible instructional techniques, and may not allow detailed diagnostics based on a student's response, it does satisfy the purposes of providing a common conceptual view of courseware materials and serving as a basis for evaluation and enhancement, as stated at the beginning of the description.

— A document editor capable of editing/formatting a document with some text, figures and equations is designed and implemented. It is part of the authoring component of the overall design and, in addition to producing a document, is capable of representing document content in such a way that can be converted to tuples as described in the conceptual schema. Hence an author may prepare his/her documents at any place where an Apple Macintosh computer is accessible. The resulting documents could then be integrated with existing materials in the courseware database. The formatting of document contents (text lines, figures and equations) is performed automatically. The author is just required to specify the type and characteristics (font, font size, etc.) of the document content to be handled.

## 7.2. Further work

As the work reported in this thesis represents a first step towards the actual implementation of a courseware preparation system, much is left to be done. In this section, some suggestions for possible directions of further work will be provided.

Since courseware database is the central component of the system, refinement of the conceptual schema (high-level description) is of highest priority over others. More research is needed to explore other factors essential for the presentation of courseware materials. Once a finalized version of the schema is ready, other components of the system could then be designed and specified in detail.

In the current design and implementation of the document editor, emphasis has been placed on the representation of various types (text, figure and equation) of document content. There is not much support for formatting and structuring an entire document. Further work is suggested to incorporate more document formatting and structuring facilities into the system, and to extend the variety of types of document content in addition to text, figure and equation.

## References

[All84] Michael W. Allen. A new approach to authoring CBE courseware, Control Data Corporation, 1984.

[Arm84] W. W. Armstrong, A Semilattice Database System , University of Alberta, 1984.

[BaG82] David S. Backer and Steve Gano, Dynamically Alterable Videodisc Displays, *Graphics Interface*, 1982.

[BBA76] Avron Barr, Marian Beard and Richard C. Atkinson, The computer as a tutorial laboratory: the Stanford BIP project, *Int. J. Man-Machine Studies Vol. 8*, (1976), pp.567-596, Institute for Mathematical Studies in the Social Science; Stanford University.

[BBB75] John Seely Brown, Richard R. Burton and Alan G Bell, SOPHIE: A Step Towards Creating a Reactive Learning Environment, *Int. J. Man-Machine Studies Vol. 7*, (1975), , Computer Science Division, Bolt Beranek and Newman Inc..

[Bun81] C. Victor Bunderson, Courseware, in *Computer-Based Instruction A State-of-the-Art-Assessment*, Harold F. Jr. ONeil (ed.), Acedemic Press, 1981, pp. 91-125.

[Dea78] P. M. Dean, Computer-assisted instruction authoring systems, *Educational Technology Vol. 18 Number 4*, (April 1978), pp.20-23.

[Dow74] M. W. Dowsey, Easy Author-entry System: a Review and a Prototype, *Int. J Man-Machine Studies, Vol. 6 Number 4*, (1974), pp.401-419.

[Fei68] Samuel L. Feingold, PLANIT - A language for CAI, *Datamation 14*, (September 1968), pp.41-47.

[FoD83] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison Wesley, Reading, MA, March 1983.

[GWR81] Robert M. Gagne, Walter Wager and Alicia Rojas, Planning and Authoring Computer-Assisted Instruction Lessons, *Educational Technology*, September 1981, pp.17-21.

[Gre82] Mark Green, FDB: A Frame Based Database System, University of Alberta; User Manual, 1982.

[Gre85] Mark Green, The University of Alberta User Interface Management System, *Proc. Siggraph '85 San Francisco*, 1985, pp. 205-213.

[HBF85] A. M. Hlady, J. W. Brahan, B. Farley, W. H. Henneker, R. A. Orchard and C. S. Phan, *NRC/DEE Bulletin Vol. 2 Number 5*, (May 1985), pp.55-63.

[Ian82] Sommerville, Ian, in *Software Engineering*, Addison-Wesley Publishers Limited, 1982.

[Kea82] Greg Kearsley, Authoring Systems in Computer Based Education, *Communication of the ACM Vol. 25 Number 7*, (July 1982), pp.429-437.

[KoP76] Elliot B. Koffman and James M. Perry, A model for generative CAI and concept selection, *Int. J. Man-Machine Studies Vol. 8*, (1976), pp. 397-410.

[Kro83] David M. Kroenke, ***Database Processing: Fundamentals, Design, Implementation***, Science Research Associates Inc., 1983.

[MSF80] M. D. Merrill, E. Schneider and K. Fletcher, TICCIT, *Educational Technology Publications*, 1980.

[Moh85] Thomas G. Moher, Videocassette Course Development Using Microcomputer Graphics, *IEEE CG&A*, June 1985, pp. 34-40.

[Neg81] Nicholas Negroponte, Media Room, *Proceedings of the SID Vol. 22 Number 2,* (1981), , Massachusetts Institute of Technology.

[ONe79] Harold F. Jr. ONeil, *Cognitive and Affective Learning Strategies* , Acedemic Press, 1979.

[Osi76] Luis Osin, SMITH: How to produce CAI courses without programming, *Int. J. Man-Machine Studies Vol. 8,* (1976), pp.207-241.

[OtT81] Setsuko Otsuki and Akira Takeuchi, A Unified C.A.I. System For Authoring, Learning and Managing Aids, *Computers in Education: Proceedings of the Third WCCE.* , 1981, pp.249-256.

[Pog80] Richard E. Pogue, The Authoring System: Interface between Author and Computer, *Research and Development in Education Vol. 14 Number 1,* (January 1980), pp.57-68.

[RoP83] Franklin C. Roberts and Ok-choon Park, Intelligent Computer-Assisted Instruction: An Explanation and Overview, *Educational Technology,* December 1983, pp.7-11.

[Rom70] Eugene William Romaniuk, *A Versatile Authoring Language for Teachers,* Ph.D. Thesis, University of Alberta, Spring 1970.

[Sch83] Jack Schwartz, Languages and Systems for Computer-aided Instruction, *Machine-Mediated Learning Vol. 1 Number 1,* (1983), pp.5-39, Courant Institute of Mathematical Science, New York University.

[Voy82] Stanley Voyce, A Functional Analysis of Courseware Authoring Languages, *AEDS Journal,* 1982.

[Wes77] M. Westrom, Summary and current status of NATAL-74, *AEDS Journal Vol. 10 Number 4,* (1977), pp.83-89.