

# **1. Abstract**

Following the WWW's (World Wide Web's) rapid growth and technical development, more and more different resources in different languages can be easily accessed. In this thesis, we present a solution to difficulties which most foreign language students encounter in reading a new language using those abundant materials.

In this solution, we make a "Study Guide Authoring System" which can automatically convert the foreign language document into a hypertext structure that promotes study by language students. Each hypertext node analyzes a different part of the foreign language document. Some nodes contained embedded comprehension questions which prompt the reader to review the relevant section of text if the questions are answered incorrectly. We also provide an on-line dictionary which can let students look up an unknown word.

Because the output documents of the Study Guide Authoring System are hypertext pages, they also can be posted on the Internet. Because of this, the Study Guide Authoring System can offer the opportunity for distance learning. A student can work independently at the computer, receive instructions visually, and respond by clicking on the image or text with a mouse. The pages are colorful and intriguing, with formats and animated graphics that help interest and motivate students and make practice enjoyable.

The Guide Reading System can be used on the Macintosh, Windows platform and UNIX environments. The input document can be plain ASCII text or HTML. The output generated will be HTML text with hyperlinks. Users can access this system via the internet from web clients like Netscape and Internet Explorer.

## **Chapter 1 Introduction**

The Internet is the catch-all word used to describe the massive world-wide network of computers. The word "internet" literally means "network of networks". In itself, the Internet is comprised of thousands of smaller regional networks scattered throughout the globe. From 1993, on any given day it connected more than 15 million users in over 50 countries. Right now, this number is much larger. People who have surfed the internet must be impressed by its convenience and variety of available information resources. Since the Internet is a world-wide network, peoples in many countries post information in their own language, such as Chinese, French, German, and so on.

We are interested in how to enhance students' comprehension in reading a new language using that variety of available resources on the Internet. The Annotated Foreign Language Document Repository is a system that addresses this idea, whereas Study Guide Authoring System is a system for teachers to create materials for Annotated Foreign Language Document Repository.

### **1.1 The Initial Idea**

In the summer of 1995, Dr. James Davis (Foreign Language Dept, University of Arkansas, Fayetteville) attended a workshop, sponsored by the French Embassy, on French language resources on the Internet. While all of the attendees were impressed by the amount and variety of available resources, many were concerned about enhancing students' comprehension of these documents.

As a Basic French Instruction professor, Dr. Davis discussed this with Dr. Berleant (University of Arkansas, Fayetteville) when he was back. They wanted to work together on

the development of computer software to help students comprehend written French texts, accessed on the World Wide Web.

The proposed software was to respond to two instructional problems: 1) the curricular dissonance between beginning foreign language instruction, which emphasizes oral skills, and advanced courses, which focus on written language; and, 2) the importance of facilitating students' independent access to foreign language resources on the "information superhighway" of the Internet (Davis, 1996).

## **1.2 The objective of this thesis**

The purpose of this thesis is to develop the Study Guide Authoring System which is designed to reinforce foreign language reading skills and is also designed according to the most recent theories of language reading skill acquisition. The primary goal of Study Guide Authoring System is to provide instructors with a means to use computer technology to engage students in the process of learning; that is, to teach students how to use information from the WWW to enhance their reading skills in a second language.

### **1.2.1 The Primary Goal**

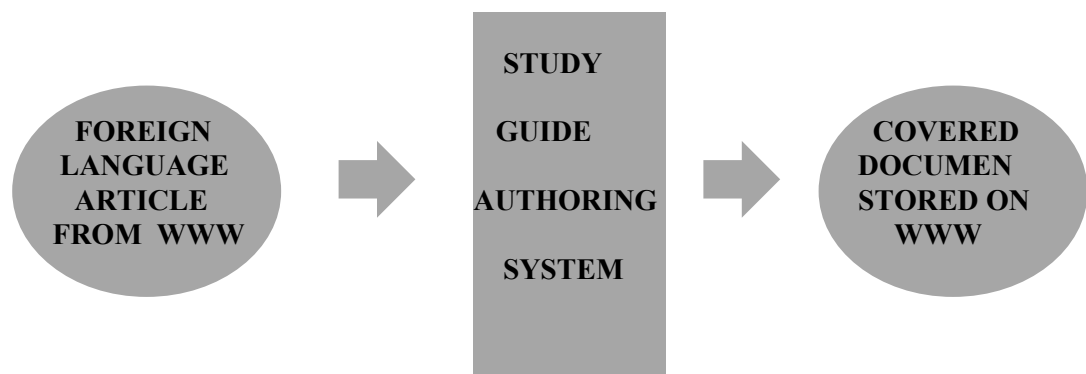
The project is consistent with the newly-published national Standards for Foreign Language Learning, particularly with Standard 3.1 ("Students reinforce and further their knowledge of other disciplines through the foreign language."). "The project should provide Foreign Languages in the Elementary Schools (FLES) instructors with a useful model for combining meaningful content and foreign language instruction; it can demonstrate at middle and high school levels how social studies can be taught through foreign languages in an already overcrowded curriculum; it can also suggest ways to include a social sciences based

Foreign Languages Across the Curriculum (FLAC) more cost-efficiently at post-secondary levels “ (Green, 1997).

Successful development of the proposed educational software must begin with consideration of the student user: What will the student be able to do (or want to do) after using the software? The educational software we developed will stress engaging the user’s attention by stimulating an initial interest, will maintain that attention by providing a challenging activity over which the user has some control, and finally, will provide a rewarding resolution that stimulates continued interest. This continued interest will prompt the user to explore more challenging levels provided by the program.

The software is to contain three challenge levels of increasing complexity through which users can navigate. Each challenge level will aim toward increasingly advanced proficiency or ability levels. These levels will not target specific school grades, but the content of each level will translate into degrees of suitability for different grade ranges.

### 1.2.2 The Process Pipeline



**Figure 1.1 Guide Authoring System Process Pipeline**

First, we can get a foreign language document from the WWW or somewhere else, and then we can use the Guide Authoring System to process this particular document. We can

process several hypertext documents this way. In those documents, comprehension tasks, such as brief writing compositions and questions concerning the documents will be provided. We also provided an on-line dictionary to let students look up the meanings of unknown words.

The hypertext document output by Study Guide Authoring System must be available at its own web site and accessible to any user of the Web.

### **1.3 The Preview Of The Thesis**

This thesis describes my work on Study Guide Authoring System and consists of seven chapters. In Chapter 2, general information about software engineering will be discussed. Chapter 3 to Chapter 7 present the phases of the software engineering life cycle, requirements phase, specification phase, design phase, implementation and integration phase, and maintenance phase.

## **Chapter 2. Software Engineering Review**

Software has become the key element in the evolution of computer-based systems and products. Over the past four decades, software has evolved from a specialized problem-

solving and information analysis tool to an industry in itself. But early “programming” culture and history have created a set of problems that persist today. Software has become a limiting factor in the evolution of computer-based systems.

## **2.1 What is software engineering**

“Software engineering is a discipline that integrates methods, tools, and procedures for the development of computer software.” (ROGER, 1992) A number of different paradigms for software engineering have been proposed, each exhibiting strengths and weaknesses, but all having a series of generic phases in common, such as requirements, specification, planning, design, implementation, integration, maintenance, and retirement. Some of these phases are called by other names. For example, the requirements and specifications phases are sometimes called systems analysis (SCHACH, 1996). Every phase pays special attention to a certain part of software process.

Generally, in the preceding list of phases there is no separate testing phase. Testing is not a separate phase, but rather an activity that takes place all the way through software production (SCHACH, 1996). Almost all of the phases need to be tested. Especially, testing is needed before the product is handed over to the client. Although sometimes testing predominates, there should never be times when no testing is being performed. If testing is treated as a separate phase, then there is a very real danger that testing will not be carried out constantly through every phase of the product development and maintenance process.

We must notice that every method has its own strengths and weaknesses. In “No Silver Bullet” [1995], Brooks specifies “accident” and “essence” as two difficulties in the software process. Essence is the collection of difficulties inherent in the nature of software development and accidents are merely incidental. Brooks also presented four aspects of essence, namely, complexity, conformity, changeability, and invisibility. With the help of

software engineering concepts accidents and essence can be minimized but there is no silver bullet to abolish them.

However, the software engineering process cannot be defined in a fixed style like a food recipe, because of the wide scope of software engineering; rather, the software engineering process establishes a baseline for the interaction between the customer and the developer.

## **2.2 Software Engineering Process in the Study Guide Authoring System**

The Study Guide Authoring System was built step by step according to the software engineering life cycle. The development process for the Study Guide Authoring System was divided into five distinct phases: requirements, specifications, design, implementation and integration, and maintenance.

### **2.2.1 The Life Cycle model**

From requirements to maintenance, the series of steps through which the product progresses is called the life-cycle model. The life-cycle history of each product is different. In the Study Guide Authoring System, we used the rapid prototyping model to construct the whole product.

A rapid prototype is a working model that is functionally equivalent to a subset of the product (SCHACH, 1996). Like all approaches to software development, the rapid prototype begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. A rapid prototype then is quickly built. The rapid prototype focuses on a representation of those aspects of the software that will be visible to the user. The rapid prototype is evaluated by the customer/user and is used to refine requirements

for the software to be developed. Once the client is satisfied that the rapid prototype indeed does most of what is required, the developers can draw up the specification document with some assurance that the product meets the client's real needs. After that, the following phases will occur. The key advantage of rapid prototype is to define the rules of the game at the beginning (Roger, 1992), that is, the customer and developer must both agree that the prototype is built to serve as a mechanism for defining requirements.

The sequence of events for the prototyping paradigm is illustrated in Figure 2.1.

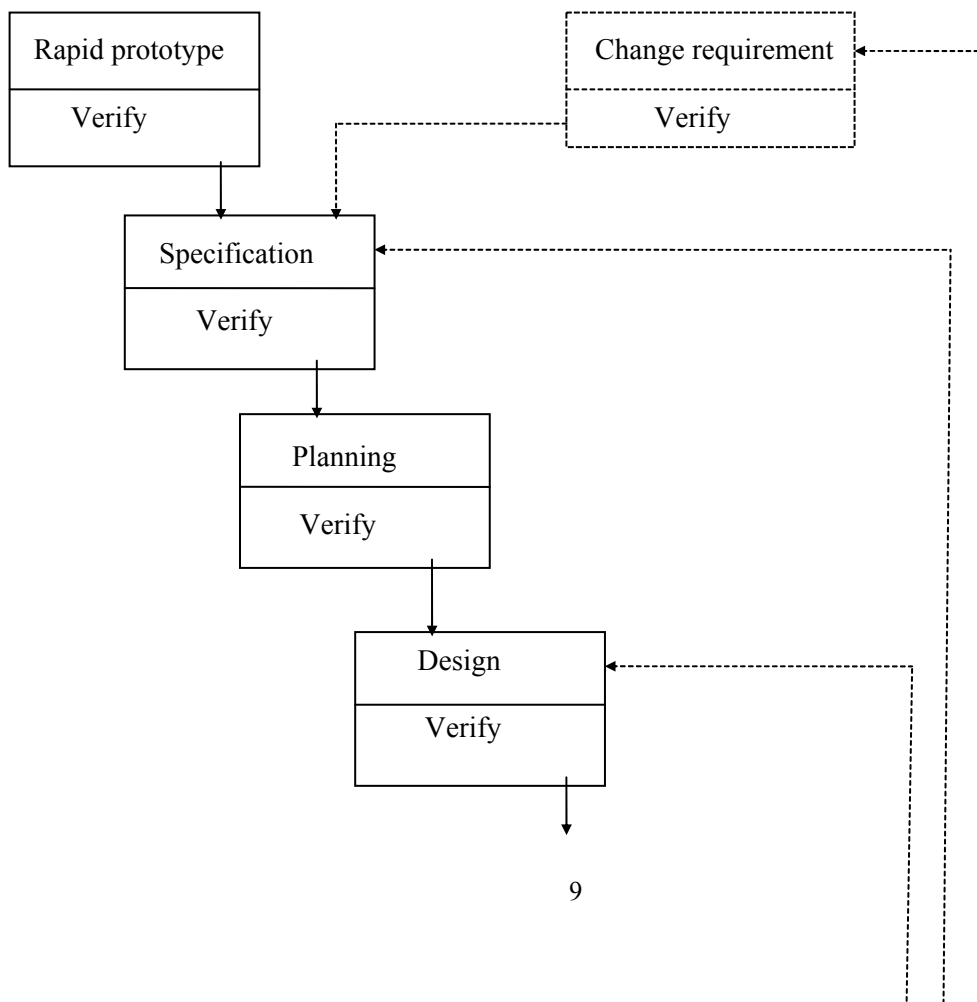
### **2.2.2 The Life Cycle Applied To The Guide Authoring System**

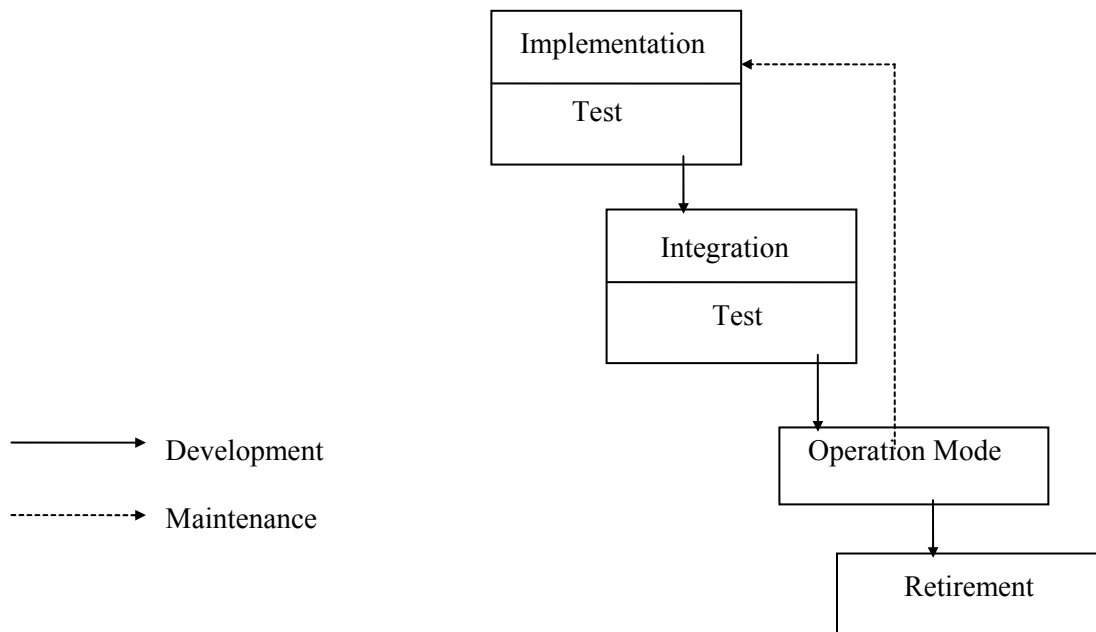
In Accordance with the rapid prototype model, we divided the whole process into 6 distinct phases: requirements, specification, design, implementation and integration, and maintenance.

1. Requirements phase: we explored and refined what the customer really needs.  
Based on that, we built rapid prototypes.
2. Specification phase: this phase handled the problem of exactly what the Study Guide Authoring System is to do.
3. Design phase: in this phase, we decided how the Study Guide Authoring System will be implemented.
4. Implementation and integration phase: the various components were coded and tested.
5. Maintenance phase: maintenance includes all changes to the product once the client has accepted it. In this phase, we discuss future enhancements to the Study Guide Authoring System.



We have now introduced some general ideas of software engineering. We will present the detailed information about how we constructed our project in the following chapters.





---

**Figure 2.1 Rapid prototyping model (From SCHACH, 1997)**

## **Chapter 3 Requirements Phase**

A complete understanding of the software requirements is essential to the success of a software development effort. No matter how well designed or well coded, a poorly analyzed and specified program will disappoint the user and bring grief to the developer. In our special case, this stage is the conceptual phase of the whole project. The fundamental issues must be resolved before an efficient algorithm can be developed, so we spent almost one whole year on this stage.

In this phase, we analyze what the Study Guide Authoring System really needs to do. For avoiding communication-based problems, we decided to build a rapid prototype of the Study Guide Authoring System. We discuss our rapid prototyping in detail in the following sections.

### **3.1 What we need to do**

The development group, which includes 3 members, began meeting regularly in fall 1996. The group discussed what the Study Guide Authoring System needs to do. The three people in this group worked on different parts of the whole project. Dr. Davis contributed the conceptual requirements, structure and content of the output, while Dr. Berleant advised graduate student Shuxuan Yan on specifications and plans. Shuxuan Yan contributed the design, implementation and integration as well as significant portions of the specifications. The prototypes were contributed to by all participants, particularly Yan and Davis.

#### **3.1.1 The functional requirements**

The Study Guide Studying System design is driven by four instructional goals: 1) engaging the student reader by building upon his/her prior knowledge of the topic of the reading selection; 2) allowing a flexible interaction with the text that ensures the level of on-line comprehension checking currently believed necessary for optimal understanding; 3) promoting active learning by providing students with meaningful tasks; and 4) developing transferable reader strategies that will be useable on or off the World Wide Web without the assistance of the program.

The design of the Study Guide Authoring System is inspired by the work of three internationally recognized foreign-language reading researchers: Elizabeth B. Bernhardt, June K. Phillips, and James Lee, as well as Dr. Davis' own research in this area.

The software will be available at its own Web Site and will be accessible for any user of the Web. Dr. Davis will archive and prepare French language documents from the Web for

use at the site. The Study Guide System will guide students to an understanding of the archived selections through active participation in several steps:

- 1) Pre-reading: students will first be led to examine the title and important sentences in a document. Students will also be guided to sentences in the passage where the important words of the title appear. An on-line dictionary will be made available to help in further understanding of the text.
- 2) Brainstorming: students will be asked to brainstorm (in the foreign language) the likely topic of the passage, in writing, on the computer.
- 3) Guided reading: as they proceed to read a selection, users will be encouraged to employ what Lee calls management strategies. (i.e., the most effective ways to approach reading the foreign language text), such as dividing the passage into smaller segments.
- 4) Comprehension checks: many activities designed to improve foreign language reading skills check comprehension only after an entire passage has been read. An important design principle of the software is that novice foreign language readers need frequent monitoring of their understanding throughout the entire process of comprehending. The proposed software will permit such an intervention with immediate feedback. For example, a user may be presented with four possible interpretations of an idea expressed in a passage. If the incorrect choice is selected, s/he will be presented with the misunderstood material again and asked to re-read the paragraph to find the correct interpretation.
- 5) Assimilation: many authorities (e.g., S. Jay Samuels and Janet Swaffar) have emphasized the importance of re-reading to comprehension. Many instructors, however, find it difficult to motivate students to re-read, when the only rationale for re-reading is to understand a passage more clearly. The program will propose meaningful tasks that will require several readings in order to be completed. The software also implements a browser

approach to reading, in which important parts of the passage tend to reappear simply because they are important.

At the end of a reading lesson, the program will also review effective comprehension strategies, thus encouraging students to use effective reading techniques independently of the program.

### **3.1.2 The interface requirement**

Our Study Guide Authoring System will have built-in GUIs (Graph User Interfaces) that are easy and intuitive to use. The interface of the Study Guide Authoring System will contain the following functions:

- Let the author retrieve foreign documents either from the WWW or from the local drive.
- Let the author select which language to study.
- Let the author specify the title or subtitle of the document s/he wants to process.
- Let the author create simple on-line dictionary for this document.
- Let the author construct multiple choice questions for some certain sentences.
- Let the author create various modification by himself by providing a text editor.

### **3.1.3 The implementation requirements**

- Work on multiple platforms such as Windows95, UNIX, MAC.
- The relationship between different modules should be high in cohesion and low in coupling.
- Make as extensible as possible (system should be extendable by adding new modules with little or no modification to existing code).

### **3.1.4 The system requirement**

IBM compatible PCs

Netscape 3.0 or Internet Explorer 3.0

UNIX (SUN Microsystems workstations)

Windows 95

### **3.2 Communication technique**

Software requirements analysis begins with members of the requirements team meeting with members of the client organization to determine what is needed in the target product. There are several techniques we can use for requirements analysis, such as interviews, scenarios, and rapid prototyping.

In the Study Guide Authoring System, we used rapid prototyping to present the result of our requirement analysis.

#### **3.2.1 Rapid prototyping**

A rapid prototype is hastily built software that exhibits the key functionality of the target product. The key point is that a rapid prototype reflects the functionality that the client sees, such as input screens and reports, but omits "hidden" aspects such as file updating (Schach, 1990).

Since the result of our system will be distributed via the WWW, we decided to use HTML (hypertext markup language) to build our rapid prototype. We spent almost one year on the requirements phase. We have built four generations of rapid prototypes to work out our requirements analysis.

#### **3.2.2 Rapid Prototype I**

We built Rapid Prototype I in fall of 1996. Rapid Prototype I was based on the HyperBrowser system, an existing software system that has already been developed at University of Arkansas, (Fatima, 1996). The HyperBrowser system takes an ordinary plaintext or .html document, producing an .html document containing numerous hyperlinks within it (Berleant and Berghel, 1996). Links are installed from each plausible keyword in a sentence to a partially or fully matching keyword in another sentence when such a match exists in the document.

This prototype was the initial stage of the Requirements Phase. We focused on how to parse the sentence and add links in it.

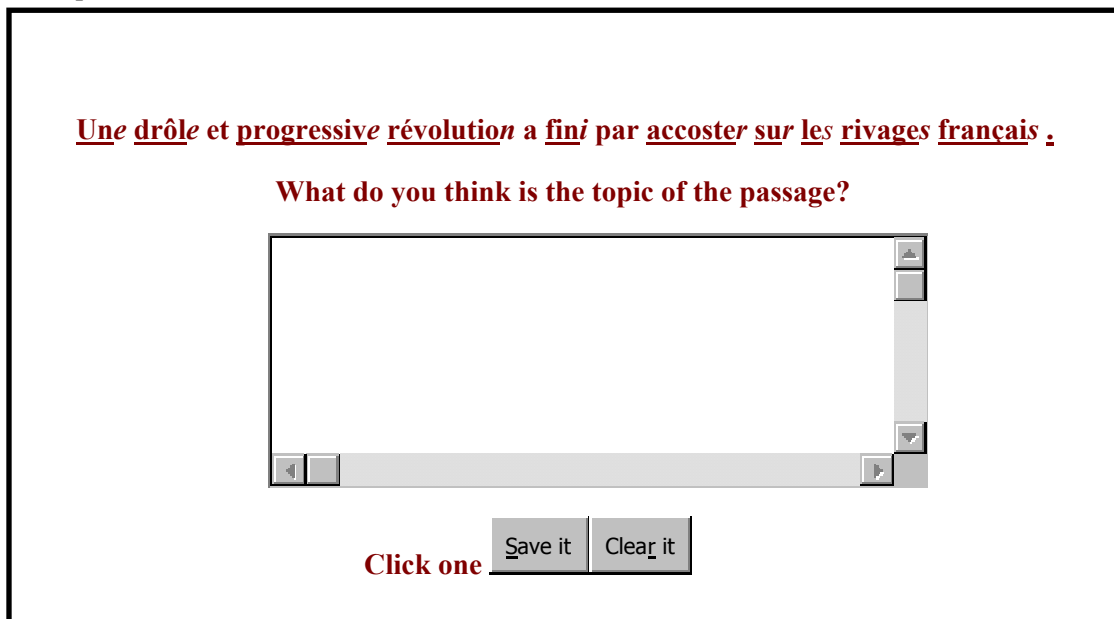


Figure 3.1 Prototype I

In the rapid prototype I (Figure 3.1), the last letter of a word in the document links to an on-line dictionary, while the rest of the word links to another similar word which is in the same document. The period of one sentence will link to a multiple-choice question. We also insert blank text areas into the document in which students can brainstorm and type in their thoughts.

The first rapid prototype was simple. It just simply processed the whole document in the same page. However, it took a lot of time to construct. That is because we tried many things in this phase and integrated the basic idea into the whole project. This prototype demonstrated the fundamental idea of whole project.

### **3.2.3 Rapid Prototype II**

After we got rapid prototype I working, we knew what we were supposed to do. Through group discussion, Dr. Davis suggested we split the whole problem into ten steps. Every step can be on a different screen.

The Prototype II is the second generation rapid prototype of the Study Guide Authoring System. We used Frames to arrange our work. Always on the left side we put the menu, and users can select the item desired. Towards the right side, we listed all ten steps. Users can select any step they want to work on. Later they can back to this page and select another step.



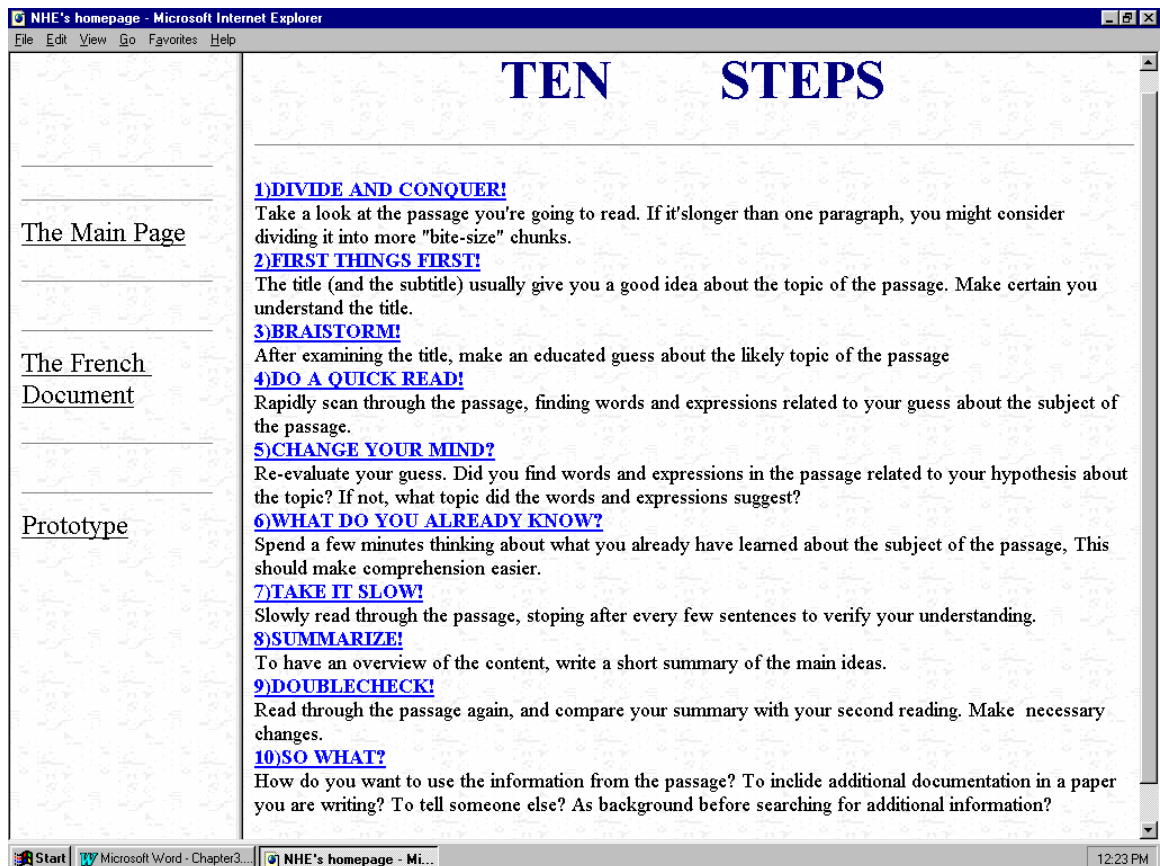


Figure 3.2 The splash page of Rapid Prototype II

### 3.2.4 Rapid Prototype III

Compared with the Rapid Prototype II, Rapid Prototype III had two big changes. First, we removed the slide menu from the left side of the main screen, and put the original document over there. The idea behind this is the student can see the changed document and the original document simultaneously. Second, we kept the ten steps study method; however we changed the technique of analyzing the words in the sentences. We remove the HyperBrowser linking strategy from our project. That means, we removed the links between similar words. Right now, if you click the whole word, it just jumps to the on-line dictionary, rather than to a similar word in the same document.

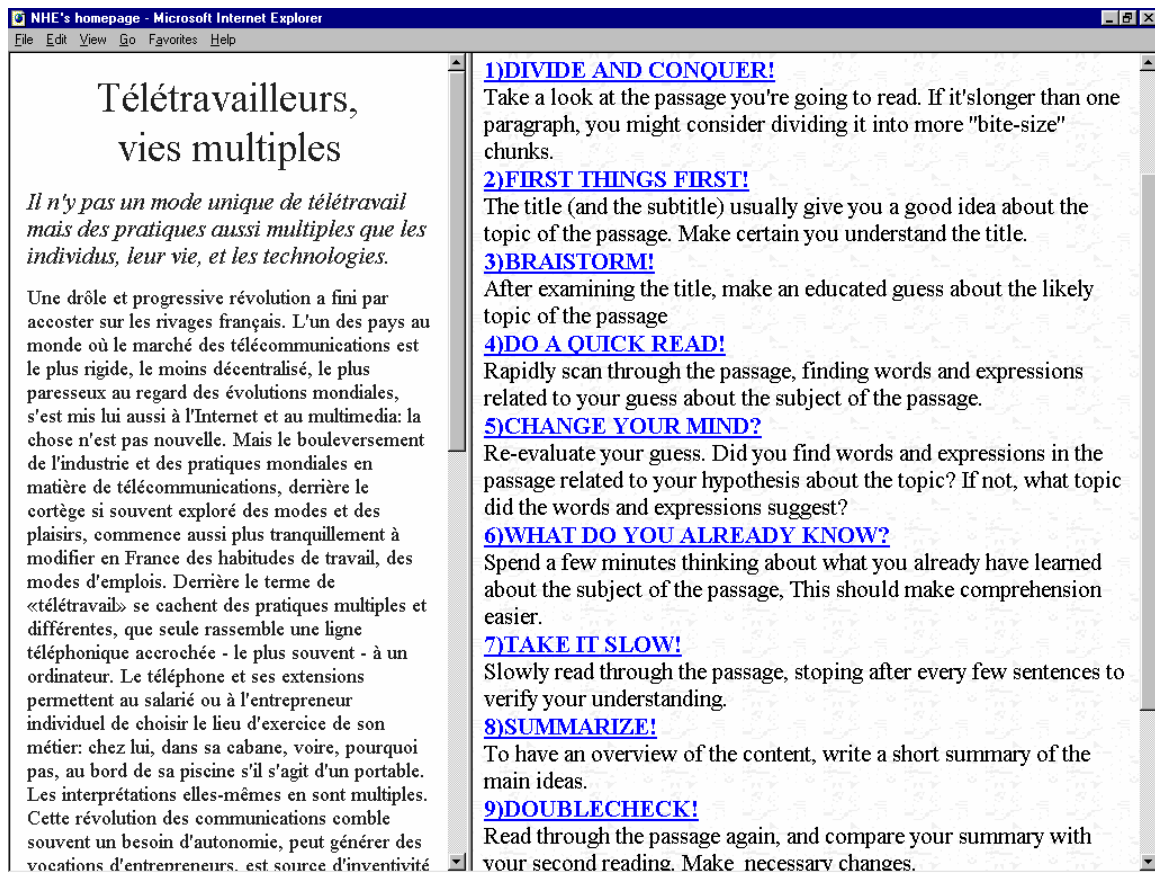


Figure 3.3 The splash page of Prototype III

### 3.2.5 Rapid Prototype IV

So far, we have constructed the basic layout of the Study Guide Authoring System. In this last Rapid Prototype, we focused on solving the following questions:

- 1) the previous version rapid prototype just can process one file in one difficulty level and one kind of foreign language. In this version, we need to construct a database which contains three challenge levels and three languages. That means that Study Guide Authoring System should allow three challenge levels for three languages, and we can

save all of that into one database. From one interface, a user can select any challenge level for any language s/he wants;

- 2) specify the emphasis of every step;
- 3) define which parts we need to link to the on-line dictionary, and which parts we need link to multiple choice question.
- 4) Insert nice images to let users feel comfortable.

## **Chapter 4 Specification Phase**

Once the client agrees with the product of the requirements phase, the specification document is drawn. As opposed to the relatively informal requirements phase, the specification document explicitly describes the functionality of a system, that is, precisely what the product is supposed to do, and lists any constraints that the product must satisfy. The specification document will include the inputs to the product and the required outputs of the product.

I used the structured analysis technique in this phase. The first step is to determine the logical data flow, as opposed to the physical data flow (that is, what happens, as opposed to how it happens). This is done by drawing a data flow diagram (DFD). The DFD uses the

basic symbols shown in Figure 3.1. After that, we will analyze the entire system step by step according to the DFD. In the following sections, we will discuss all of these steps in detail.

Compared with the requirements phase, while the requirements phase focuses on system content, the specification phase combines content with presentation and details to produce a model ready for implementation (Schach, 1993).

#### **4.1 The data flow diagram of the Study Guide Authoring System**

See Figures 4.1 through 4.5

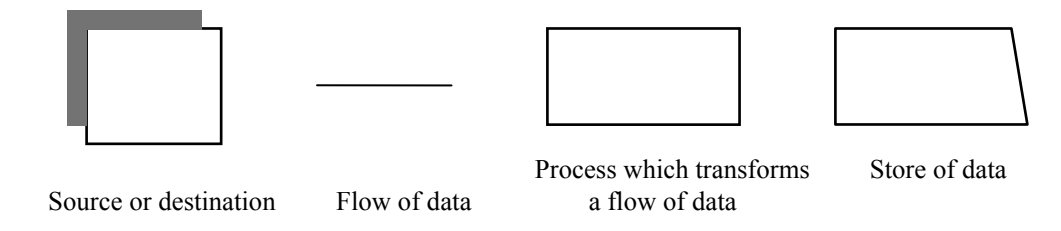
#### **4.2 Basic strategies for the Study Guide Authoring System**

Basically, the output pages of the Study Guide Authoring System will be available at its own Web site and will be accessible for any user of the Web. So first of all, we should know what these output pages are, and then we can create a better way to build these pages.

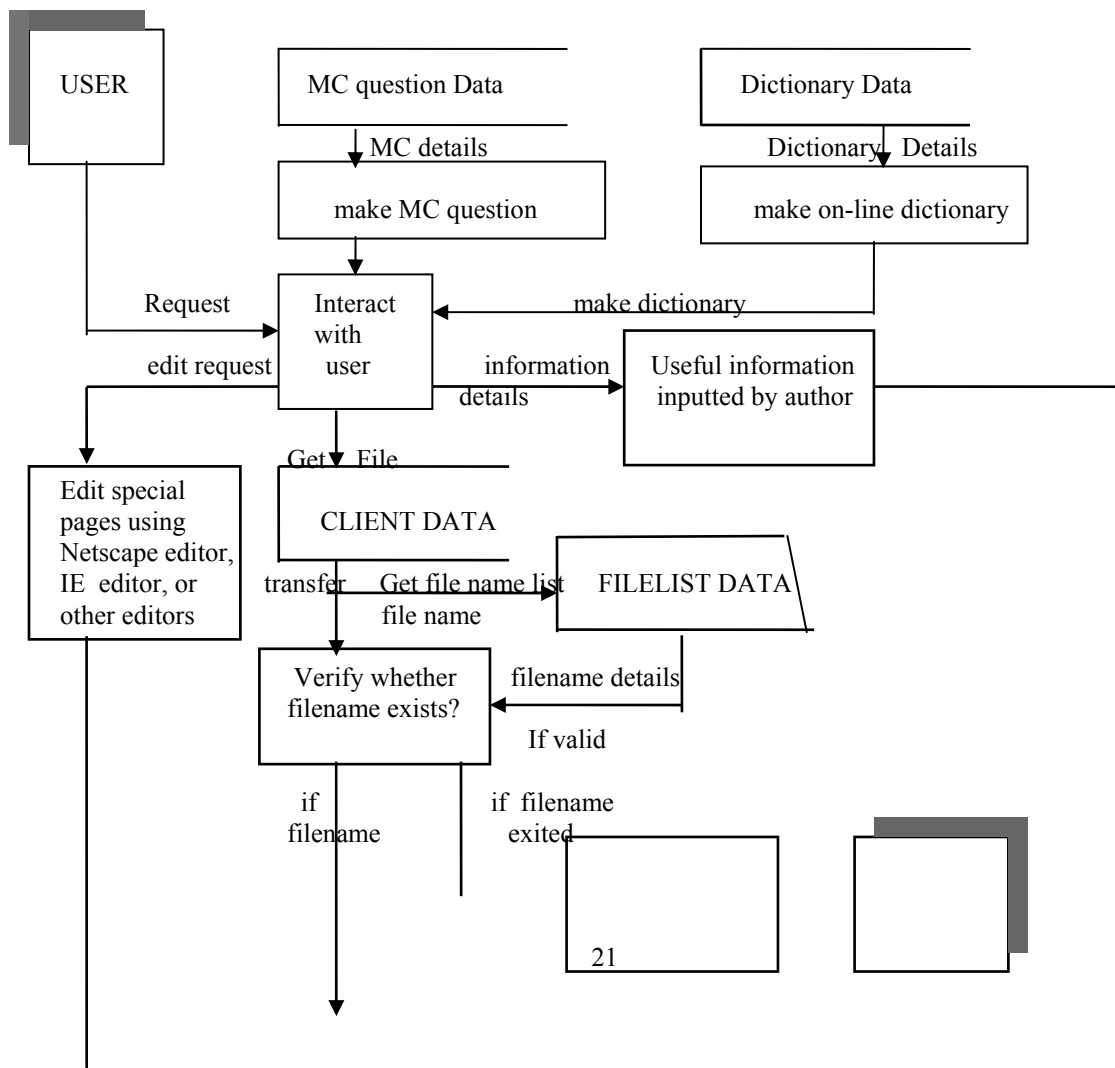
The output pages of The Study Guide Authoring System consist of a heavily annotated version of some pre-existing foreign language document. Every time the Study Guide Authoring System processes one document it automatically saves its result into a repository of processed documents. In this repository, if you want to fully study one foreign

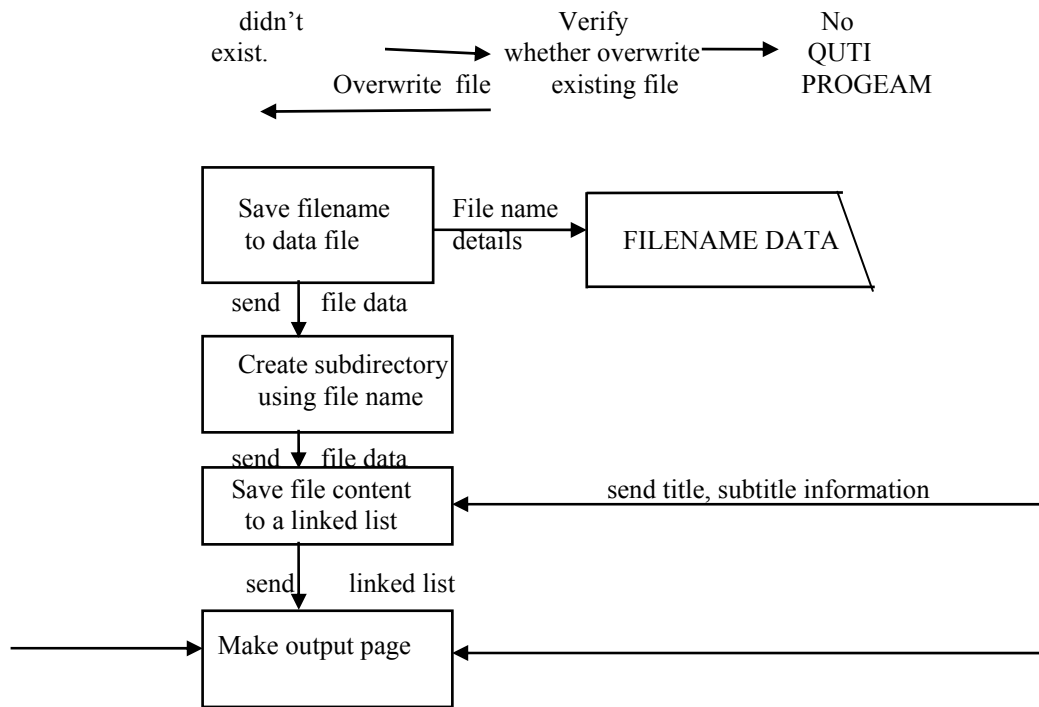
language document, you would access at least 22 pages. That means, the output of the Study Guide Authoring System consists of 22 pages, for each document it processes.

In fact, we just need to build 19 new pages in each document, because three pages are common to all processed documents. (For example, see the first page, see Figure 3.6. Every time, when you want to read a document from the repository, we can use this page as our startup page, and from this page, link to the particular processed foreign document which we want.)

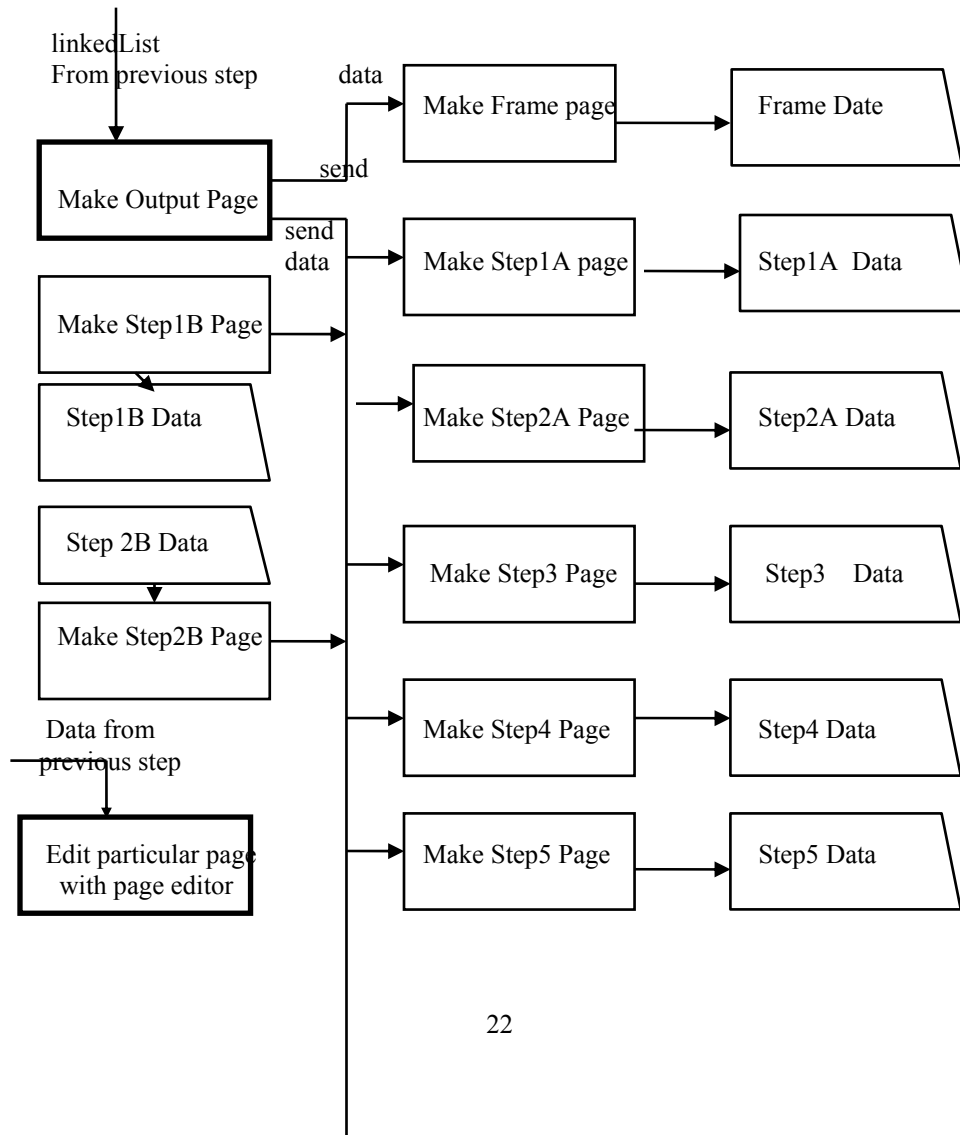


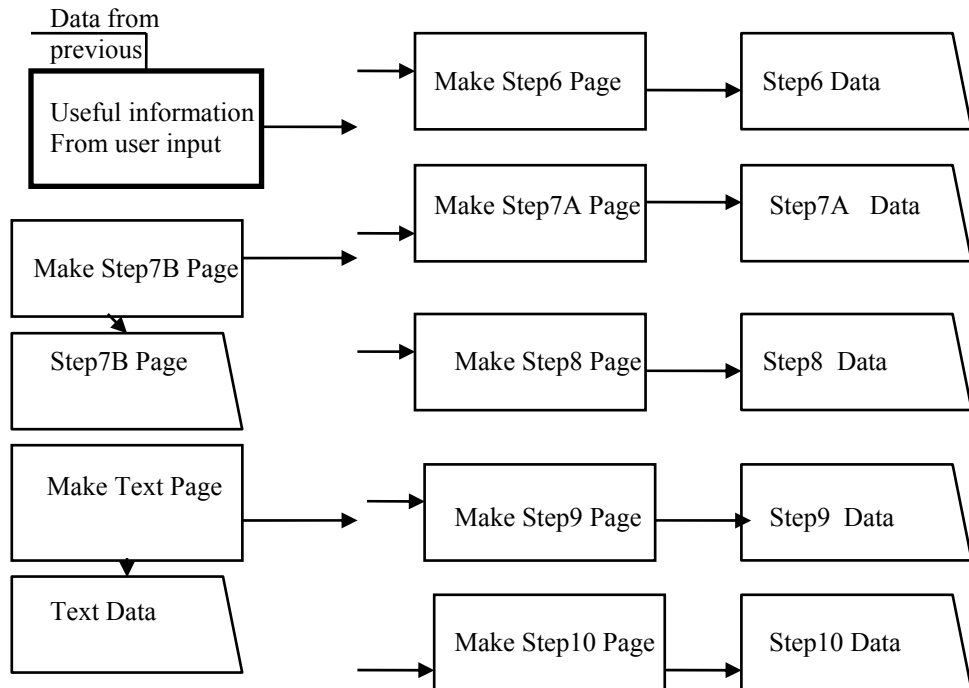
**Figure 4.1 Symbols of the Study Guide Authoring System structured systems analysis.**



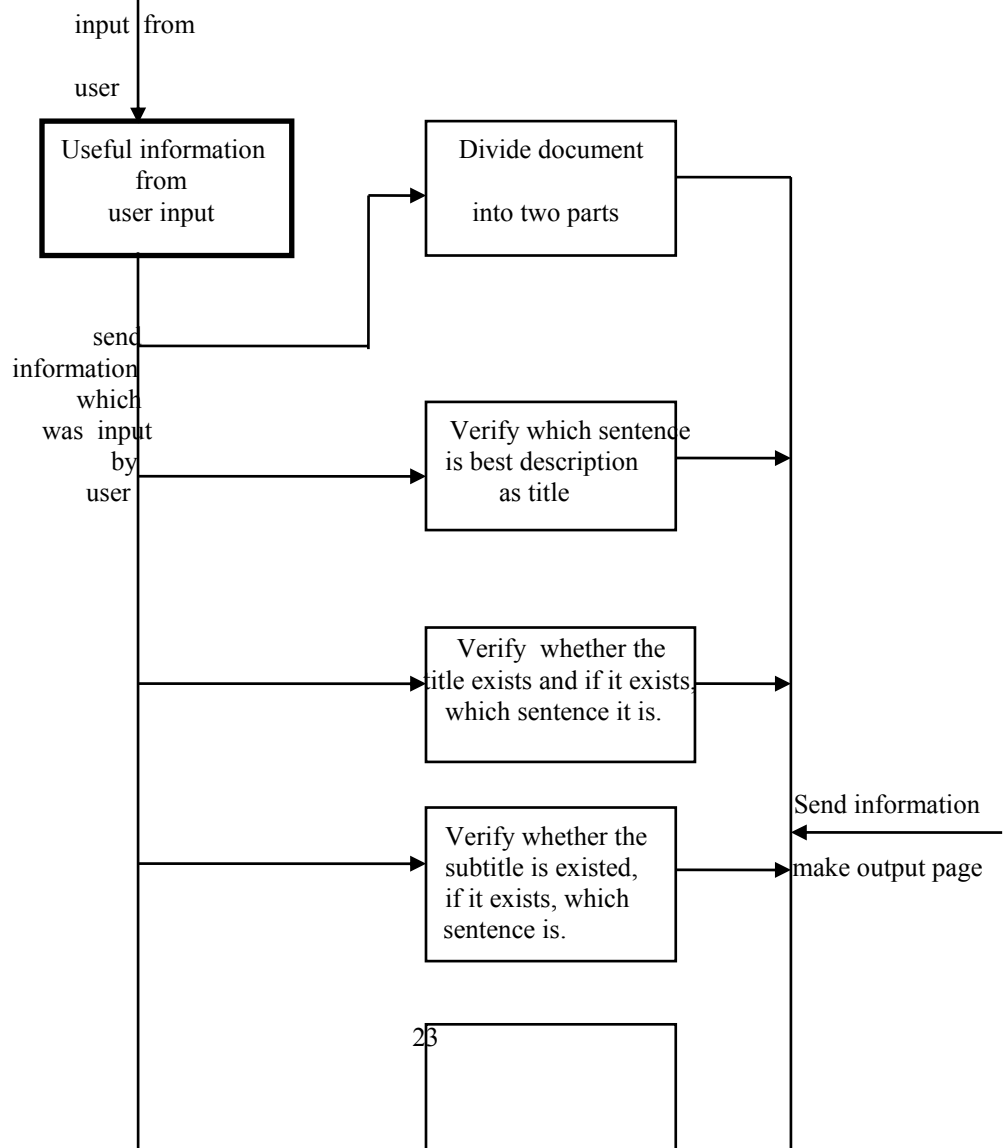


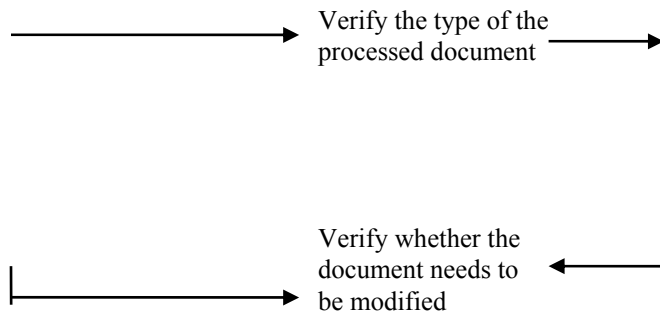
**Figure 4.2 Data Flow Diagram of Study Guide Authoring System**



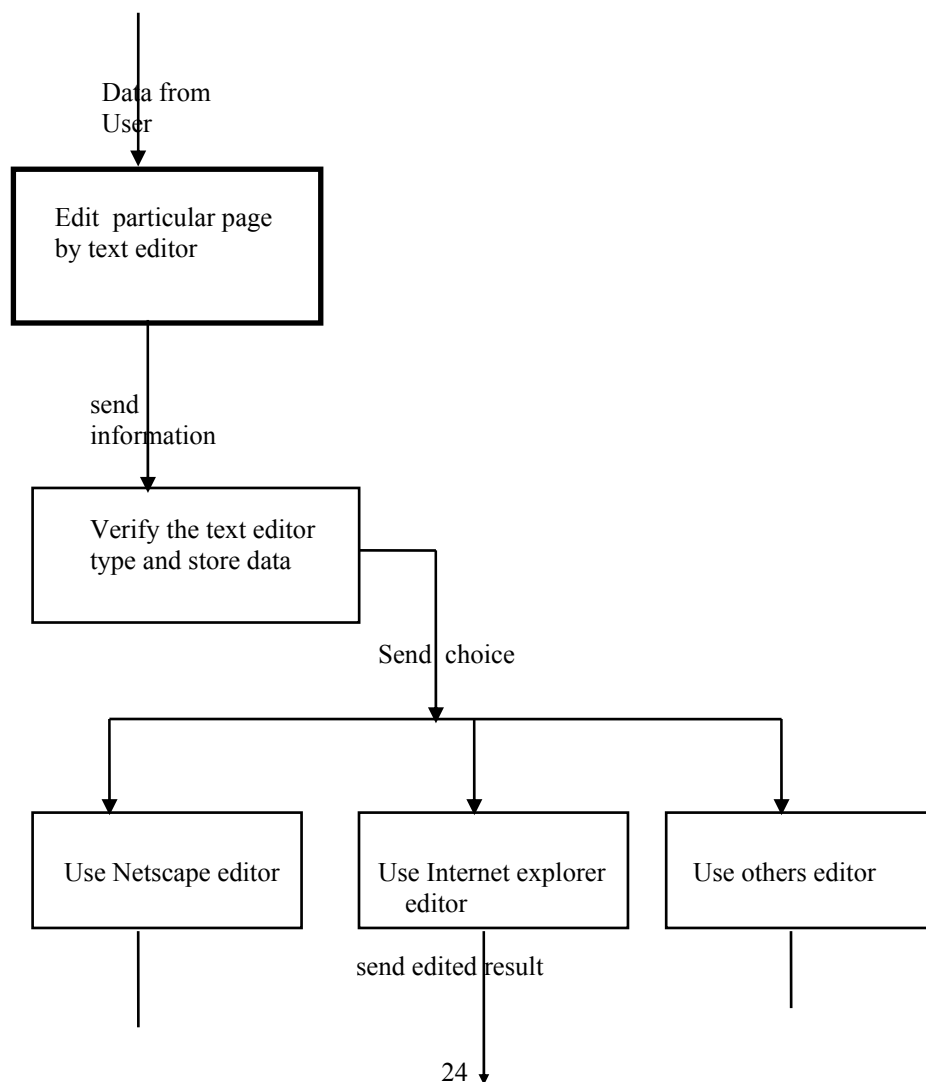


**Figure 4.3 Data flow diagram of the Study Guide Authoring System**

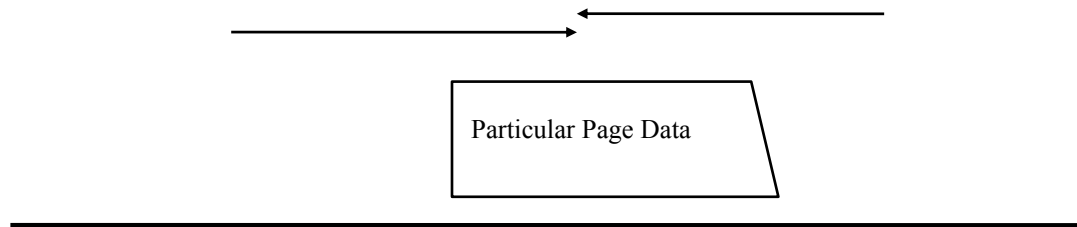




**Figure 4.4** Data flow diagram for the Study Guide Authoring System, refinement for inputted information.







**Figure 4.5** Data flow diagram for the Study Guide Authoring System, the refinement for “edit particular page”.

On the other hand, the other 17 pages we must build one by one using the Study Guide Authoring System. We can divide those 17 pages into two types.

In one type of page, for each foreign language document, these pages have similar content except for a little information. For example, PracticeSider page provides a menu which contains ten buttons associated with the ten steps. When you click on a button, the page which links to this button will show up. The information for each foreign language document in this page is the same except the link goes to a different directory. So, we do not need to use a program to build these pages. What we are supposed to do is to save these pages in a particular directory. When we build the pages of a particular foreign language document using the Study Guide Authoring System, we can just copy these pages to the directory which contain the pages of this particular document.

In the other type of page, every document contains special information concerning the particular document. We called this type “Ten Step” pages. The “Ten Step” page which describes how to study this foreign language document is one example. Every such page

contains a different thing to study about that foreign document. Further, every such page has a similar page structure (See Figure 4.7). The first part is the “header” part. The second part is the “explaining” part, which contains material introducing the study objective of the page. The third part is the “content” part. For the first two parts, we can use “boiler plate” that already exists. As to the third part, we should use the authoring program to build it step by step.

Lastly, notice that we use a lot of images in the output pages. Those images are the same for different documents. So, we can save these images in a special directory which can be linked to whenever needed.

Thus the Study Guide Authoring System will consist of three subdirectories at least. One subdirectory contains some HTML source pages. One subdirectory contains all of the images we use. And one subdirectory contain some pages which we do not need to create for each document.

After we process one foreign document, one new subdirectory will be created which contains all of the output pages about this particular foreign language document.



**Figure 4.6** The first page of the Annotated Foreign Language Document Repository

Step 1A: Divide and Conquer - Microsoft Internet Explorer



File Edit View Go Favorites Help


**NOTE:** We suggest that you **DOWNLOAD THE PRACTICE PASSAGE** and keep it beside your computer as you practice the ten steps.

---

**PRACTICE TEXT**  
**LANGUAGE: FRENCH**  
**DIFFICULTY LEVEL: CHALLENGING**


---

 **UNDER CONSTRUCTION**  Note to colleagues: The program will include an on-line dictionary. To have access to the dictionary, users will click on an unknown word or expression.

 Please send any comments about this and other pages to:  
[jndavis@comp.uark.edu](mailto:jndavis@comp.uark.edu) **THANKS!**

---

**STEP ONE.**  
**DIVIDE AND CONQUER!**  
**BRIEFLY EXAMINE THE PASSAGE BELOW AND DECIDE WHERE YOU WANT TO DIVIDE IT.**



**Libération**

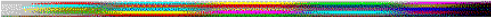
---


**Télétravailleurs, vies multiples**

*Il n'y a pas un mode unique de télétravail mais des pratiques aussi multiples que les individus, leur vie, et les technologies.*

Une drôle et progressive révolution a fini par accoster sur les rivages français. L'un des pays au monde où le marché des télécommunications est le plus rigide, le moins décentralisé, le plus paresseux au regard des évolutions mondiales, s'est mis lui aussi à l'Internet et au multimedia: la chose n'est pas nouvelle. Mais le bouleversement de l'industrie et des pratiques mondiales en matière de télécommunications, derrière le cortège si souvent exploré des modes et des plaisirs, commence aussi plus tranquillement à modifier en France des habitudes de travail, des modes d'emplois. Derrière le terme de **télétravail** se cachent des pratiques multiples et différentes, que seule rassemble une ligne téléphonique accrochée - le plus souvent - à un ordinateur. Le téléphone et ses extensions permettent au salarié ou à l'entrepreneur de choisir le lieu d'exercice de son métier: chez lui, dans sa cabane, voire, pourquoi pas, au bord de sa piscine s'il s'agit d'un portable. Les interprétations elles-mêmes en sont multipliées. Cette révolution des communications comble souvent un besoin d'autonomie, peut générer des vocations d'entrepreneurs, est source d'inventivité et de création. L'envers de la médaille, pour d'autres, peut être la précarisation, l'isolement, la version moderne de la couturière des Vosges: le travail à domicile, après tout, ne date pas d'hier. Pour tous, le troc existe entre l'immédiateté, l'instantanéité - qui permet, par exemple, à un chasseur de têtes international d'opérer dans la Creuse - et l'absence de proximité. Au point qu'on a vu se recréer des bureaux quasi fictifs, qui permettent à plusieurs **télétravailleurs** de côtoyer leurs semblables.

Mais le flux des informations captées par l'Internet a aussi pour effet de modifier, ou plutôt de compléter, le marché du travail lui-même. On peut aujourd'hui chercher un emploi - ou, à l'autre extrémité, embaucher - en usant des ressources du Net. Complément plus que substitut, pour l'instant, au marché traditionnel de l'offre et de la demande d'emploi. Mais au rythme où vont les choses, les petites annonces prennent le chemin du Web et de ses multi-sites plus rapidement que les articles d'un journal, pour ne prendre qu'un exemple. La révolution est à plusieurs vitesses.

Source: WebLibération. 

 [CLICK here to see where we divided the passage.](#)

---

**This is the first Part of Very part HEAD PART**

**This is the second Part of Very page STEP EXPLAINING PART**

**This is the third Part of Very page CONTENT PART**

**Figure 4.7 The step1A page of the "Ten Step" Method**

### **4.3 Detail logic process of the Guide Authoring System**

Actually, there are two parts in the Study Guide Authoring System: the Annotated Foreign Language Document Repository and the Guide Studying System. The Annotated Foreign Language Document Repository is used to store processed foreign language documents and let students read them via the internet, whereas the Guide Authoring System is used to construct the Annotated Foreign Language Document Repository and is used by instructors. Here, I want to introduce the detailed logic of the Guide Authoring System.

#### **4.3.1 Step1: Interact with User.**

Before processing every foreign document, the user must provide some important information about this document and how to process it to the system. The information we need the user to input is:

1. The foreign language document's URL, or location on the local machine.
2. Which language this particular foreign language document is written in.
3. Does this document have a title? If it has, where in the document is it?  
Otherwise, the author needs to type in the title.
4. Does this document have a subtitle? If it has, where is it? Otherwise, the author needs to input the subtitle.
5. How many paragraphs does this document have?
6. After which paragraph can we divide the whole document into two parts?
7. Does the original document need to be modified a little bit?
8. Which sentence in the document is the best description for the title?
9. Which text editor does the user want to use? Or, where is the location of the pre-edited page?
10. Inputting multiple choice questions for the document.
11. Inputting definitions for an on-line dictionary for the document.

First, the author needs to provide the address or location of the document, so the system can use that information to retrieve the document. We need the user to tell the system which language the document is written in. The Study Guide Authoring System can process three different languages currently, French, Germany, and Spanish. Each of the three will be processed by a different module, since each language has its own characteristic features. The first version of the Study Guide Authoring System only can process French language documents.

We need the author to input the number of paragraphs in a foreign language document. There are two advantages by using this number inputted by the author. First, it will make the author check the document at least one time, so s/he can also check if the correct paragraph boundaries were used. Second, it can let the system check itself. If the number inputted by author is different from the number automatically calculated by system, the system will provide a error message warning the author. So, that will make processing more accurate.

We need the author to input the information about the title or the subtitle information of this document, because the title and subtitle are very important in our method. Some steps need information about the title or subtitle. Of course, it is conceivable to program the system to find the title and subtitle automatically; however, we did not find a good way to do this. We considered using the period to differentiate between title and subtitle and content, because we read a lot of documents and found that the title and subtitle are often the first two sentences and are without periods. That means, if the first sentence of the document has no period, that sentence may be the title. If the second sentence of the document has no period, that sentence may be the subtitle. So we want to enable the program to differentiate the title and subtitle based on this method. The result of that method is obviously not accurate. But, the title or subtitle are really important for this system. So we decided to let the user select or input the title and subtitle by themselves, and the trade-off is to get accurate information at the expense

of requiring additional user interaction. Sometimes, there is no subtitle in the document. In that case, authors can either type in “(no subtitle)” in the subtitle box, since system does not allow to leave any text box blank, or author can type in a subtitle composed by themselves.

The system also needs the author to input where to divide the document, because in the “Ten Steps” method, sometimes we need to process one part of document in detail, and the other part just requires some simple processing.

We need to divide the document based on paragraph boundaries. Since the original document is an HTML file, we cannot use the general method of searching paragraph boundaries of plain text. Normally, the HTML file uses <P> and sometimes </P> to differentiate between two paragraph; however, we found sometimes they just use <BR> or some other tag as a paragraph boundary. If the document does not use <P> and </P> to separate paragraphs, it is hard to confirm which is a paragraph boundary. So, a better way is let the authoring system’s users check the original document. If the document does not use the normal way to split paragraphs, they can input the <P> and </P> on their own. The currently system requires that a paragraph begin with <p> and end with </p>, both tags must be present for each paragraph.

We need to allow the authoring system’s users to input definitions to the on-line dictionary. This is at the request of Dr. Davis. We also need to allow use of an existing on-line dictionary. This will allow students to look up tough words.

We also need to allow authoring system users to input multiple choice questions to check comprehension of the students.

#### **4.3.2 Step 2: Verify whether this document already exists in the repository.**

We do this in case this document was already processed by the system and saved in the database, or this document has the same name as a processed document in the repository.



In our system, when we process a foreign document, we create a new subdirectory, which contains all of the pages associated with this processed foreign language document. The name of the subdirectory is the same as the filename. So, if the document exists or the filename is a duplicate, the system would have trouble. We can either stop processing and change the filename, or overwrite the old document which is saved in the repository.

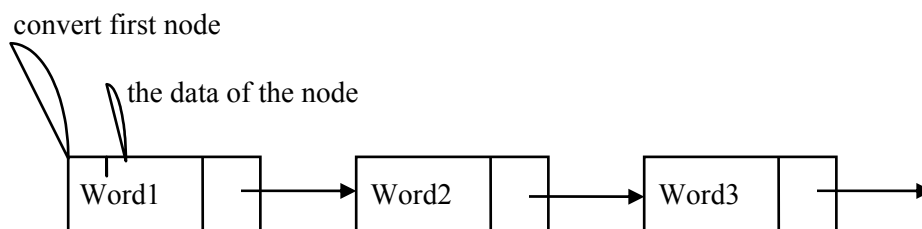
#### 4.3.3 Step 3: Save the file name to the file name list.

To conveniently check the filename to make sure it is unique, we create a file name list which contains all of the file names we have already processed. Every time we want to convert a document, we just read the file containing this list and check whether the new document's filename is already there or not. If the filename is a new one, before we convert the document, we can save the filename in this data file. If the filename exists, we can check with the user and either overwrite the old document or stop the conversion process.

#### 4.3.4 Step 4: Create subdirectory.

After checking the filename, we will use this file name to create a new subdirectory which contains all the output pages for this document. So, when students want to study this document, the database just links to this special subdirectory.

#### 4.3.5 Step 5: Save file content to a linked list.





---

### **Figure 4.8 Converting the file contents to a linked list**

The input file is read and stored into three 1--dimensional linked list data structures. One linked list will stored the title sentence, the second one will save the subtitle. The last one will save contents of document without the title and the subtitle. The title and subtitle information will be available from step 1.

In the last linked list, for each sentence in the document a 1-dimensional linked list is constructed with each word (delimited by a space or tab) in a node. Multiple spaces are neglected and considered as a single space.

#### **4.3.6 Step 6: Make output files.**

There are 19 files to be output, so this step is really a major part of the system. We will split this step into 12 parts, and discuss each part in detail; however, multiple choice questions and on-line dictionary will be discussed later.

##### **A. make PracticeStp.html file**

The PracticeStp.html page contains two Frames. This file is the opening page and lists the ten steps and provides a menu. Because this file is not affected by the content of the foreign language document, we do not have to construct it in the program. We just copy it into this particular subdirectory.

##### **B. make PracticeSidebar.html file**

PracticeSidebar.html contains links to each of the 10 steps. This file also does not need to be created by the program except to change the links to the right subdirectory.

### **C. make FrnchChallSt1A.html and FrnchChallSt1B.html files**

Step 1 is called “DIVIDE AND CONQUER”. It consists of two separate pages: FrnchStp1A.html and FrnchStp1B.html.

The FrnchStp1A.html file will let student briefly examine the whole document and decide where to divide the document into parts. This page belongs to the “Ten Step” page which is built by the program. It is discussed in section 4.2. We split the file containing the “Ten Step” page into three parts: “header”, “explaining”, and “content”. Each file has the same “header” part and different next two parts. The “header” and “explaining” parts are already contained in different files saved in the source files subdirectory. The “content” part will be constructed by the program. So, for the FrnchChallStp1A.html file, the program reads the “header” and “explaining” parts from source files subdirectory into a buffer first, and then the program constructs the “content” part for FrnchChallStp1A.html. It reads the title linked list, subtitle linked list and document content linked list into a buffer. Lastly, the program joins these three parts together and writes into an output file named FrnchChallSt1A.html.

The FrnchChallSt1B is a page which contains the author’s suggestion about how to split the whole document. The way of making this page is a little different from the FrnchStp1A.html file. First, read the first two parts into a buffer, and then access the variable containing the information which the user input about how to split this document. We can use this integer variable to split the content linked list into two parts and insert the number tag. Lastly join these three parts and write into a data file called FrnchChallSt1B.html.

### **D. make FrnchChallSt2A.html and FrnchChallSt2B.html page**

The step 2 is called “FIRST THINGS FIRST”. The exercise in this step is to ask the student which is the first thing to think about. In this step, the student will do exercises

containing the title and subtitle of the document. This is because the title (and the subtitle) usually give a good idea about the topic of the document. There are two files (FrnchChallSt2A.html and FrnchChallSt2B.html) for this step.

FrnchChallSt2A.html is a little hard to make, because the information in this page is so flexible. This file has two components. The first component deals with the title. Authors who want to make this page may want to guide the student in analyzing the title in different ways, depending on the nature of the title. So it is hard to just use a program to automatically generate this file. So the program provides a text editor and lets the author create this part on his or her own, or let the author create the file using another text editor and input to the program that location of the file. Then the program can read the file into a buffer. The other component guides a student to read the first part of the foreign language document and decide which sentence in the first part best describes the topic of the title. A student can check whether his/her decision is correct or not by clicking the period of any sentence. If his or her understanding is correct, a pop-up window will show up and tell them "You are right!". Otherwise, the pop window will tell them "Sorry, wrong guess!". The method for this is to insert the `<a...>` and `</a>` tags into the linked list representing the document around the period of every sentence. Of course, in the `<a...>` tag, we need some special attribute. We will talk about this in the implementation chapter.

Finally, we write these four parts into a data file called FrnchChallSt2A.html. For FrnchChallSt2B.html, we must solve another problem. With this file, the student will do an exercise concerning the subtitle. We must link every word of the subtitle to an on-line dictionary.

At first, we also read the "header" and "explaining" part into a buffer. For the "content" part, we must think about how to insert a link attached to every word of the subtitle.

We know that FrnchChallSt2B.html is a html file. In html, a link is created when the anchor tags `<a...>` and `</a>` are placed around some text. The anchor needs additional information to be valid. The `<a...>` has added information that identifies the destination of the link in question. This additional information is called an attribute of the tag. If the anchor tag `<a...>` has the HREF attribute added to it, we can link to the URL following the HREF attribute. For example:

```
< A HREF = "Dictionary.html"> Dictionary </a>
```

When you click "Dictionary", it will link to the Dictionary.html file.

But an anchor doesn't have to be a pointer to other data. It can be a place mark for other links to point to. In this case, the anchor is then named by using the NAME attribute with the

```
<a...> tag. An example is: <a name = "unique"> unique </a>
```

Notice that the anchor has no HREF variable in it: the anchor isn't pointing to anything else, it can be pointed to.

To reference this specific point, the reference anchor uses a # sign to indicate the named location within the document. For example, the reference anchor looks like:

```
<a HREF = "Dictionary.html#unique"> unique </a>
```

In this case, when you click on "unique", it will take you to the place which has the `< a NAME = "unique"> unique </a>` anchor in the Dictionary.html file.

This technique is important in our system. In our special case, we need to make sure the Dictionary.html file contains the necessary NAME anchors. We will talk about this later. In the file FrnchStp2B.html, we just need to insert `<a HREF="...">` and `</a>` anchors into the linked list representation of the subtitle between every word. For example, if the original linked list looks this: This is subtitle.

The converted linked list looks like this:

`<a href = "Dictionary.html#This">This </a> <a href = "Dictionary.html#is"> is </a> <a href = "Dictionary.html#subtitle">.`

Finally, we need to join all these parts together and write into a data file called FrnchChallSt2B.html.

#### **E. make FrnchChallSt3.html page**

Step 3 is called "Brainstorm". In this step, students will be asked to read the title and subtitle once again. At the bottom of the page, they get a blank textarea, where students can think about the title, and type in their guess in the blank textarea.

To build this page, first, read "header" and "explaining" parts into a buffer. Second, read the title and subtitle into a buffer, and attach every word of the title and subtitle to the on-line dictionary. Finally, join these two buffers together and write into a data file, called FrnchChallSt3.html.

In this page, we need do some work so that we can transfer the text typed in by student to textareas in other pages for later steps. For achieving that, we need to write a CGI program attached to the textarea. So, after students type in the guess and click the submit button, this text will automatically transfer to the textareas in those later pages. We will talk more about the CGI program in the Implementation chapter.

#### **F. make FrnchChallSt4.html page**

The function of step 4 is to do a quick read. That means, students are asked to quickly scan through the first paragraph, and look for words and expressions related to their guess about the topic of the passage.

In the step 4 page, we also need to plug in a blank textarea, which will contain the text the student previously typed into the textarea of step 3. So, first, we need to build the title

buffer and the subtitle buffer. Second, we need to make a linked list for the first paragraph which is split from the linked list for the entire document, and read it into a buffer. Finally, combine these three buffers together, and write into a data file named FrnchChallSt4.html.

#### **G. make FrnchChallSt5.html page**

The topic of step 5 is “Change Your Mind?”. The student can look at what he or she guessed earlier about the topic of passage again, and find the words and expressions that supported the original guess in the first paragraph one more time. If the student did not find the words and expressions, s/he needs to return to step 4. Otherwise, s/he can continue to step 6. The composition of this page is almost like for the previous page except for the second part. So, we can build this file just like building the file for step 4 file.

#### **H. make FrnchChallSt6.html**

The topic of step 6 is “What You Already Know”. In this page, we provide a blank textarea. The student can spend a few minutes to think about what s/he has already learned about the subject of the passage, and type it into the blank textarea. This should make comprehension easier.

For building this page, we also need to read the “header” and “explaining” parts into a buffer. The “content” part should include the first paragraph and title and subtitle linked lists. We combine these three parts together, and write out to a data file called FrnchChallSt6.html. Then, FrnchChallSt6.html will be done.

#### **I. make FrnchChallSt7A.html and FrnchChall7B.html**

The topic of step 7 is “Take It Slow”. This step will let student slowly read through the passage, stopping after every few sentences to verify his or her understanding. This page consists of two independent files: FrnchChall7A.html and FrnchChall7B.html.

FrnchChall7A is a complicated file. First, we need to read the “header” and “explaining” parts into a buffer. As to the third part, it can be divided into two components: one component lets students read the first three sentences and find the conjugated verb. Students also need to do some multiple choice questions to check their understanding. The other component lets students read the first part of the document. When they finish reading the document, they can click the period of any sentence, and a pop-up window will show up which contains a multiple choice question relating to this sentence. That can check whether the student understands this sentence.

To finish the “content” part of step7A, we had better work on its two components separately. For the first component, it is hard to use a program to create it, because almost all of the text in this part need the author to type it in on their own. So, we think the better and easier way to build this part is to provide a text editor, and let authors make this part by themselves. After that, the authors can save this part into a data file, and tell the system where it is. The system will read this data file in and combine it with the other parts. For the other component, we can do it just like when building the on-line dictionary in step 2 except inserting a link between every sentence rather than every word (see the make FrnchChallStp2B.html section).

Finally, we can put the three buffers together and write them into a data file named FrnchChallStp7A.html.

The FrnchChallStep7B page lets students read the other part of the document and answer multiple choice questions for checking their understanding. So we can make this page

like we made the second component of the step 7A page except writing all components into the data file named FrnchChallStp7B.html.

#### **J. make FrnchChallSt8.html page**

The topic of step 8 is “Summarize”. This step will let students have an overview of the document, and write a short summary of the main ideas. In this page, we will provide a blank textarea, and let students write what they want to write. When they finish writing, this message will transfer to textareas in other pages

We can make this page as we made step 2A. We can insert a blank textarea into the explaining part and via a CGI program transfer the message into pages for other pages.

Finally, we combine the first two parts of the page with the title, subtitle and content linked lists and write this into a data file called FrnchChallSt8.html.

#### **K. make FrnchChallSt9.html**

The topic of the step 9 is “Double Check”. That means, the student is asked to read through the passage again, and compare his or her summary with the new understanding gained by the second reading, making any necessary changes.

The way to make this page is simple. We just need to combine the “header” and “explaining” parts of the page with the title, subtitle and content linked lists and write into a data file named FrnchChallSt9.html. We need to notice that the bottom of this page should have a textarea containing the message from step 8.

#### **L. make FrnchChallSt10.html**

Step 10 is the last step of the ten step method. The topic of it is “What’s Next”. Students can think about what else they want to know about the topic. At the bottom of this



page, students will be able to click on a link to French language search engines on the Web. When they reach a search engine, they enter a keyword from the passage to find further information on the topic.

To make this page, we also can simply join the “header” and “explaining” parts and the title, subtitle and content linked list together. Lastly, we can write all these into a data file called FrnchChallSt10.html.

#### **M. make FrnchChallTxt.html page**

This page contains the whole content of the document, and every word can be linked to the on-line dictionary, which can let students read the whole document using on on-line dictionary.

So we make this page by putting a link to the on-line dictionary between every word. Finally, write the buffer into the data file called FrnchChallTxt.html.

#### **4.3.7 Step 7: make Dictionary.html page**

This page will contain information about the meanings of tough words in the document, as determined by the author. In this page, every word will have associated definition. As we discussed in describing the “make FrnchChallSt2B.html page”, we must insert `<a name = "word"></a>` before the position where the definition of “word” occurs. For Example:

Suppose the author already has input the definition of the French word “que”. Before, we stored this definition into the data file named Dictionary.html. We must insert the relative anchor before the definition. So the Dictionary.html page should store the following material:

```
<a Name = "que"></a>
```

que

the definition of que here.

This `<a name=...></a>` anchor will be relative, with `<a href= "Dictionary.html#que"`  
`> que </a>` linking to it from other files. When the student clicks the “que” word from the  
page used for various of the ten steps, Dictionary.html will be accessed and will show the  
student the definition of que.

#### **4.3.8 Step 8: make MultiChoice.html page**

MultiChoice.html page contains information for the multiple choice questions for  
every sentence in the document. We build this using the same technique as in making  
Dictionary.html, However, we use “PnSn” to stand for the sentence inside the `<a`  
`name=...>...</a>` anchor, rather than the whole sentence. For example, `<a name =`  
`”P2S2”>..</a>` is placed right before paragraph 2 (“P2”), sentence 2 (“S2”).

After the author inputs a multiple choice question, we need to add a link first, and  
then save the question in the data file named MultiChoice.html.

#### **4.4 Define the Physical Resources.**

Dictionary.html

It is a sequential file, which contains the word definitions, stored in the subdirectory  
containing the pages for the specific document.

FrnchChallSt1APart.html

It is a sequential file, which saves the explaining part of step 1A, stored in the  
SourcePage subdirectory, which contains all existing “Head” part and “Explaining” parts for  
every steps.

FrnchChallSt1BPart.html

It is a sequential file, which saves the explaining part of step 1B, stored in the SourcePage subdirectory.

FrnchChallSt2APart.html

It is a sequential file, which saves the explaining part of step 2A, stored in the SourcePage subdirectory.

FrnchChallSt2BPart.html

It is a sequential file, which saves the explaining part of step 2B, stored in the SourcePage subdirectory.

FrnchChallSt3Part.html

It is a sequential file, which saves the explaining part of step 3, stored in the SourcePage subdirectory.

FrnchChallSt4Part.html

It is a sequential file, which saves the explaining part of step 4, stored in the SourcePage subdirectory.

FrnchChallSt5Part.html

It is a sequential file, which saves the explaining part of step 5, stored in the SourcePage subdirectory.

FrnchChallSt6Part.html

It is a sequential file, which saves the explaining part of step 6, stored in the SourcePage subdirectory.

FrnchChallSt7APart.html

It is a sequential file, which saves the explaining part of step 7A, stored in the SourcePage subdirectory.

FrnchChallSt7BPart.html

It is a sequential file, which saves the explaining part of step 4, stored in the SourcePage subdirectory.

FrnchChallSt8Part.html

It is a sequential file, which saves the explaining part of step 8, stored in the SourcePage subdirectory.

FrnchChallSt9Part.html

It is a sequential file, which saves the explaining part of step 9, stored in the SourcePage subdirectory.

FrnchChallSt10Part.html

It is a sequential file, which saves the explaining part of step 10, stored in the SourcePage subdirectory.

Head.html

It is a sequential file, which contains the head part used in the pages for various steps in the ten steps.

MultiChoice.html

It is a sequential file, which will save the multiple choice question about every sentence, stored in the SourcePage subdirectory.

PracticeSidebar.html

It is a sequential file, which contains the ten steps menu, stored in the SourcePage subdirectory.

PracticeStps.html

It is a sequential file, which contains the frame structure, stored in the SourcePage subdirectory.

## **4.5 Perform Sizing of the System**

Less than 1 Megabyte of storage is need for the software. Each processed document requires about 0.1 Megabyte of disk space data storage. Of course, that depends on the size of the foreign language document.

## **Chapter 5 Design Phase**

During the design phase, the development team breaks the large project down into manageable pieces, modules. The modules are then assigned to different subgroups which then assign individuals to the different tasks. The development team should specify what the inputs to the modules will be and what the desired output values will be (James Riggs).

The development team should not in build in the design phase a cookbook recipe for the development of the procedures; rather, the development team should merely establish coding standards such as variable naming conventions and interface design.

The complete detailed design of the Study Guide Authoring System is given at Appendix A. Each module is written in language independent format describing the module's

name, type, parameters it takes, the error messages it displays, files accessed and changed and the other modules it calls.

First the methods of each class are given. They are sorted alphabetically by class name, and by method name within class name.

## **Chapter 6 Implementation and Integration Phases**

The implementation phase and integration phase are where the code is written and all the developers merge their modules together producing the desired software product. Since the developer of the Study Guide Authoring System is just one person, we describe these two stages in one chapter.

Almost all of the modules developed in the design phase were written in Java. Based on the client's requirements, we also used HTML, Java Script, and CGI programming in this project. In the following sections, we discuss the language we used, why we used this kind of language, and how we used this language.

### **6.1 Java Application**

The study Guide Authoring System is a typical windows program. We normally can use Visual C++, Visual Java, Visual Basic and so on to make this kind of program. So why did we choose Visual Java as our major tool?

#### **6.1.1 Why did we choose the Java?**

Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture, neutral, portable, high-performance, multithreaded, and dynamic language (Flanagan, 1996).

Based on the requirements phase, the Study Guide Authoring System needs to be run on different platforms. So the first thing to look at in selecting the language is to look for a platform-independent language. For this, Java is the best choice. This is because Java is an interpreted language: the Java compiler generates byte-codes for the Java Virtual Machine(JVM), rather than native machine code. To actually run a Java program, you use the Java interpreter to execute the compiled byte-codes. Because Java byte-codes are platform-independent, Java programs can run on any platform that the JVM has been ported to.

In addition, the major work of programming the Study Guide Authoring System is text processing. So we need to do a lot of work on string handling. The string in C++ is just a null-terminated array of 8-bit characters. Unlike in C++, Java Strings are first-class objects. Strings as objects provide several advantages to the programmer. First, the manner in which you obtain strings and elements of strings is consistent across all strings and all systems. Second, the programming interface for the Java String class is well-defined, so Java strings function predictably every time. Third, the Java String class does extensive runtime checking for boundary conditions. It catches errors for you. So the Java program will reliably and obviously crash, whereas the C++ program will do something obscure.

Thus, we choose Visual Java as our major language to make the Study Guide Authoring System.

### 6.1.2 Java and Study Guide Authoring System

We want to introduce several basic programming techniques using Java which we used in programming the Study Guide Authoring System.

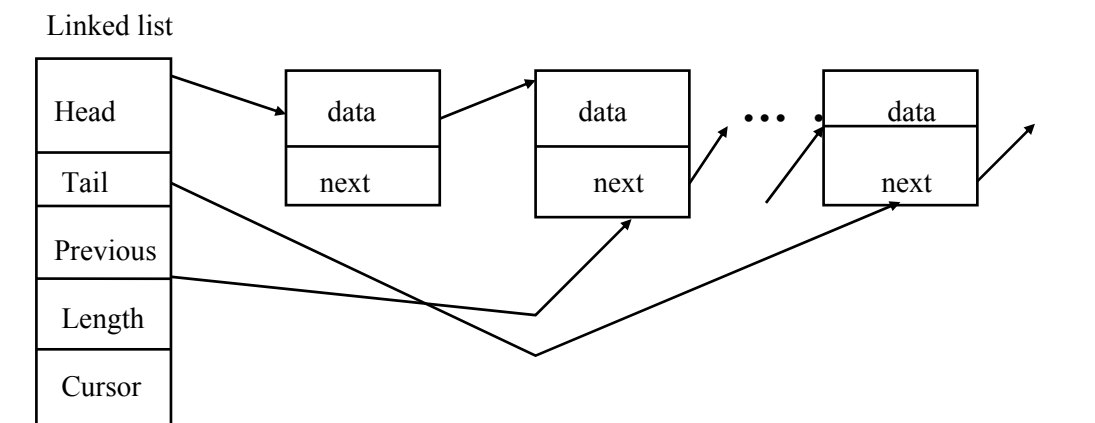
#### A. Linked list

When a document is read in by the Study Guide Authoring System, we store it in a linked list, because we need to rapidly insert and remove elements in the middle of the sequence. The structure of the linked list we used appears in Figure 7-1.

In the diagram, the list header contains a pointer to the first link, which stores a data item and a pointer to the next link. That link, likewise, stores a data item and a pointer to the next link, and so on. The last link has a null pointer to denote the end of the list.

Some people think that you cannot program a linked list in Java, because linked lists need pointers, and Java doesn't have them. Linked lists are actually easier to program in Java than in just about any other programming language. Every object reference in Java is already a pointer. This makes a basic Link class simple.

```
Class Link
{ Object data;
  Link next; //pointer to another object of the class
}
```





### **Figure 6-1      Linked list**

Moreover, we also create a List class which holds a pointer to the head link, and (in order to make rapid insertion at the tail end of the list easier) we include a pointer to the tail as well. We also keep a field for the list size called “Length”, so we do not need to traverse the entire list every time the caller wants to have an element count. Furthermore, there must be some way of inserting in the middle of the list. We do this by storing a cursor with the list object. The “Cursor” points to a specific element in the linked list. The reset method resets the cursor to the beginning of the list. The nextElement methods advances it to the next element. This permits positioning of the cursor anywhere in the linked list. The remove method removes the element under the cursor, and the put method inserts a new object before the cursor. In the List class, we have a “clone” method. This is needed because reference types are not passed by value. Assigning one object to another in Java does not copy the object. It merely assigns two references to the same object. To copy the data of one object into another object, we use the clone() method. The clone method is a protected method of the Object class, which means that your code cannot simply call it. So we create a clone method in the List class which implements the Cloneable interface (Java built-in function). The clone method can clone the linked list object.

By the way, the Java language also provides a Vector class which can also store data like a linked list. But with my own linked list, we have direct access to the node links; therefore, the node insertion and removal operations are much more efficient with a link list than with a Vector object.

#### **B. Reading a document from a remote web site**

In the Study Guide Authoring System, we need to provide the ability to read a foreign document from a remote web site. This function in Java is very easy to do. We know that Java is a distributed language. This means, simply, the URL class and related classes in the java.net package make it almost as easy to read a remote file or resource as it is to read a local file. Actually, the object referred to by the URL can be downloaded with a single call to read(). For example, in our program, we just use the following method to access file from a remote site.

```
private URL m_theURL=null;
private String m_urlString;
private byte m_buf[];
private int m_bufSize;
private InputStream m_input;
private long m_length=0;

public String read() throws IOException
{
    m_input = m_theURL.openStream();
    int ch;
    StringBuffer buffer=new StringBuffer();
    while (true){
        int n = m_input.read(m_buf, 0, m_bufSize);
        if (n == -1) break;
        buffer.append(new String(m_buf,0,0,n));
        m_length +=n;
    }
    m_input.close();
    return new String(buffer);
}
```

## 6.2 Hypertext and the Study Guide Authoring System

The operation of the Web relies on hypertext as its means of interacting with users. Hypertext is basically the same as regular text - it can be stored, read, searched, or edited - with an important exception: hypertext contains connections within the text to other documents.

For instance, suppose you were able to somehow select (with a mouse or with your finger) the word "hypertext" in the sentence before this one. In a hypertext system, you would then have one (or more) documents related to hypertext appear before you - perhaps a history of

hypertext, for example, These new texts would themselves have links and connections to other documents - continually selecting words in a text would take you on a free-associative tour of information. In this way, hypertext links, called hyperlinks, can create a complex virtual web of connections.

Because the output pages of the Study Guide Authoring System will be posted on the web, a basic component of those pages is hypertext. We use hyperlinks to connect pages.

### **6.3 Common Gateway Interface (CGI)**

The Common Gateway Interface (CGI) is an interface to the Web server that enables you to expand the server's functionality. Using CGI, you can interact with users who access your site. CGI enables you to extend the capability of your server to parse input from the browser client and return information based on the user input. CGI can provide an interface that enables the programmer to write programs that can easily communicate with the server.

After the foreign language document is processed by the Study Guide Authoring System, we get several output pages, which can guide students reading this document. The client demands that those study guide pages need to have communication functions. That means, if a student writes his brainstorm on one of those pages, this brainstorm should be displayed on other pages later. It is impossible to use HTML to do this. So we make a simple independent CGI program using the C language. The function of this CGI program is to get the message input by user, and then write this message to other pages. For detailed information about this CGI program, see the Detailed Design Phase section.

### **6.4 JavaScript**

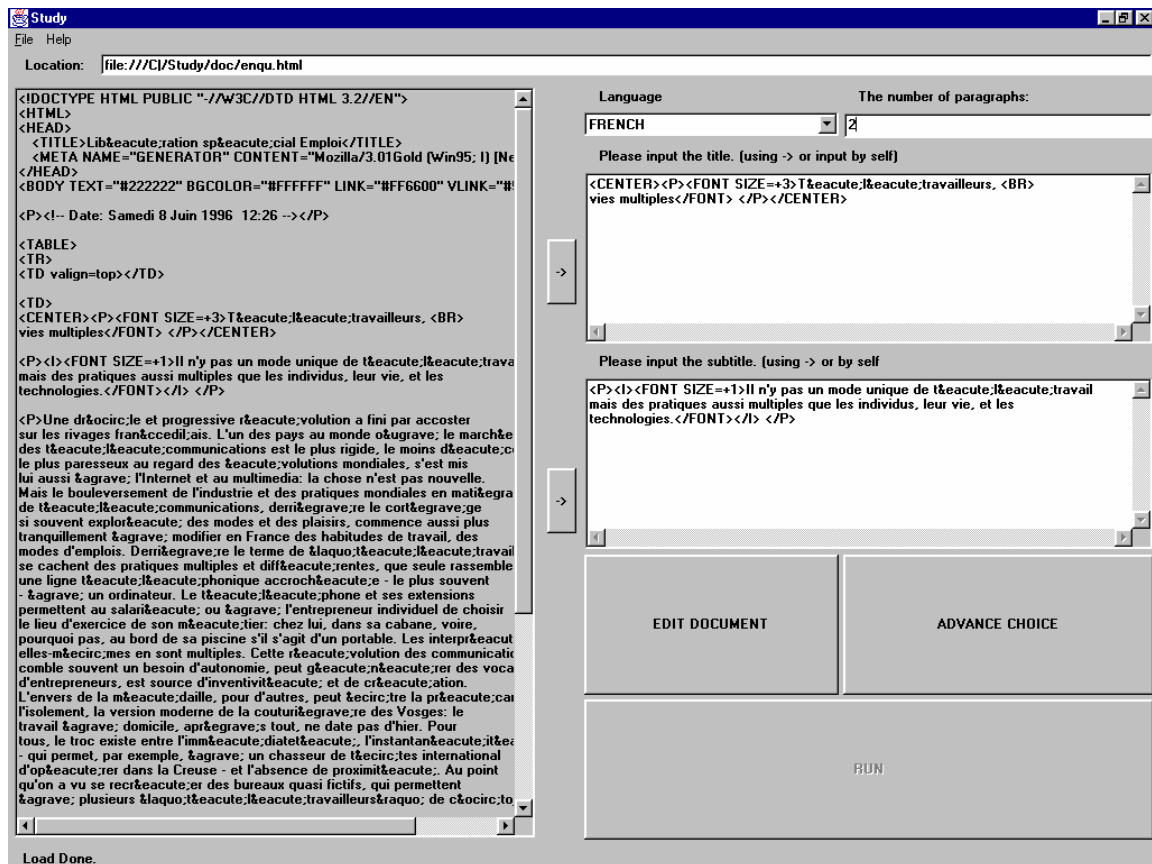
JavaScript is a new scripting language which is being developed by Netscape. With JavaScript we can easily create interactive web pages. JavaScript is not Java! Many people believe that JavaScript is Java because of the similar names. This is not true though. JavaScript is not Java. If we want to run scripts written in JavaScript, we need a JavaScript-enabled browser - for example the Netscape Navigator (since version 2.0) or the Microsoft Internet Explorer (MSIE - since version 3.0). Since these two browsers are widely used many people are able to run scripts written in JavaScript. JavaScript code is embedded directly into an HTML page.

For our project, we use JavaScript to make pop-up windows and pop-up message boxes. Of course, we can use other techniques to do this kind of work; however, JavaScript is an easy way.

## **6.5 The interface of the Study Guide Authoring System**

The user interface of the Study Guide Authoring System consists of 4 pages: "Main" page, "Advance" page, "Dictionary" page and "MC" page. I will introduce these four pages in the following sections.

### **6.5.1 “Main” page**



**Figure 6.2 the "Main" page of user interface.**

There is a menu bar on the "Main" page. The menu bar has two pop-up menus: File and Help. The "File" menu has three menuItems: "Open File", "Open Location", and "Exit". The "Open File" menuItem is used to open a local HTML file, the "Open Location" is used to open a remote file by requesting the user to input a URL, and a "Exit" MenuItem which is used to quit the program.

The "Help" menu has two MenuItem: "Help" and "About". The "Help" MenuItem opens a new window and displays a help text to help users understand know this system, and the "About" which tells the user the version of this system.

There is a "Location" TextField on the top of the "Main" page interface. The "Location" field is used to input the filename of the local HTML file or a URL of a remote Web site HTML file.

There is a bigger TextArea box at the left side of the “Main” page. This box is called the “Content” box and is used to hold the content of the file accessed by the system.

At the right side of the “Main” page, there are a Combo box and an input TextField on the first line. With the Combo box, the user can select the language that the foreign language document is written in. The TextField is used to input the number of paragraphs of the original document. There are two following TextArea fields at the right side of the “Main” Page. They are “Title” box and “Subtitle” Box. The “Title” box is used to contain the accessed document’s title, which can be input by user or be transferred from the “Content” box by the “→” button. The “Subtitle” is used to contain the accessed document’s subtitle, which can be input by user or be transferred from the “Content” box by the “←” button.

There are three buttons on the right side of “Main” page, the “Edit” button, the “Advance” button, and the “Run” button. The “Edit” button is used to enable the “Content” box, and let the user do some necessary modification on the accessed document. The “Advance” button is used to open the “Advance” page interface, which can let user input other information. The “Advance” button won’t work before the user inputs the number of paragraphs, title and subtitle information. The “Run” button is used to make the output pages after inputting all of the necessary information. The “Run” button is enable after clicking on the “Advance” button.

Near the middle of the “Main” page, there are two “→” buttons. They are used to transfer the selected text from the “Content” TextArea to the “Title” TextArea and “Subtitle” TextArea. The upper button is for the “Title” TextArea and the other is for the “Subtitle” TextArea.

There is a “Status” line on the bottom of the “Main” page, which is used to output error messages when the system resources the foreign language document.

### **6.5.2 The “Advance” page**

**Advance**

In first part of document, which sentence is the best description for the title? (Input PnSn)

Input the paragraph's number from which the document is divided into two parts. (Integer)

Please choice which editor you want to use.

☒ Netscape      ☐ Iexplorer      ☐ Load existing file

     FILE NAME FOR STEP2

     FILE NAME FOR STEP7

**Figure 6.3 “Advance” page of user interface**

The “Advance” page is used to let user input some essential information for processing the document. On the first two lines, there are two input TextFields. The first line is used to let the user input the sentence number (in the format of “PnSn”, where “Pn” stands for the paragraph number and “Sn” stands for the sentence number.), which is the best description of the accessed document’s title. The second TextField is used to let user input the paragraph number (it is a integer) after which the whole document is best divided into two parts.

There is a check box group in the third line of the “Advance” page. The user can select any one of those three. This check box determines the editor to use for the next two lines on the page. They are “Edit Step 2” button, “Edit Step 7” button, “File name for Step 2” TextField, and “File name for Step 7” TextField. If user chooses the “Netscape” check box, that means, the user chooses to use Netscape’s editor to edit the step 2 page or the step 7 page when s/he clicks the “Edit Step 2” button or “Edit Step 7” button. If the user chooses the

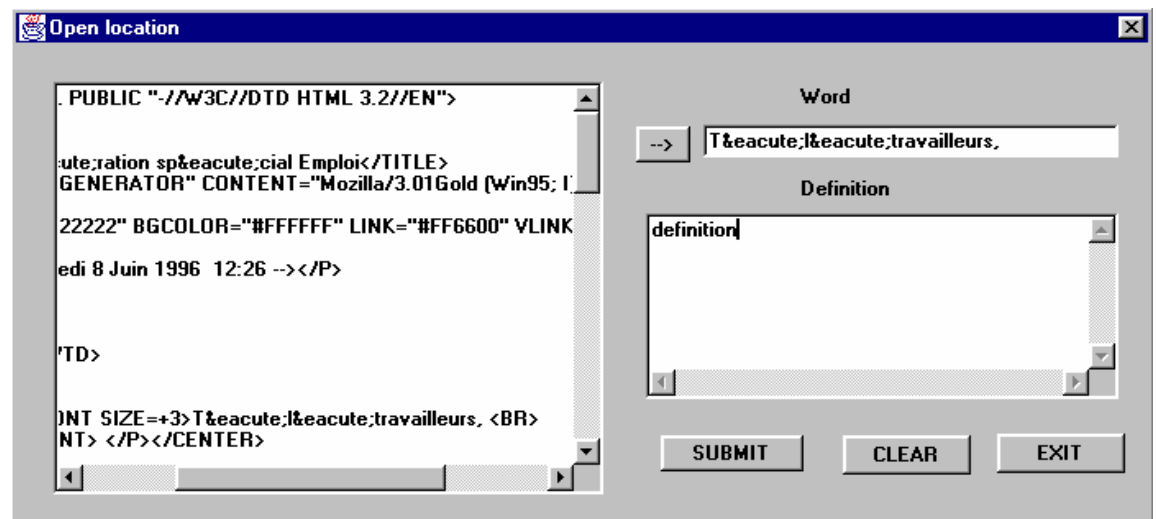
“Explorer” check box, that means, the user chooses to use the Internet Explorer’s editor to edit the step 2 page or the step 7 page when s/he clicks the “Edit Step2” button or “Edit Step 7” button. If the user chooses the “load existing file” check box, s/he can input the paths of two existing files in the “File name for step 2” TextField and “File name for step 7” separately (in the format “C:/study/step2.html”).

The “Make Dictionary” button is used to show the “Dictionary” authoring page.

The “Make Multi Choice” button is used to show the multiple choice question authoring page.

The “Submit” button is used to exit the “Advance” page and apply the information given to it.

### 6.5.3 The “Dictionary” page



**Figure 6.4 “Dictionary” page of user interface**

The “Dictionary” page is used to let the user create the on-line dictionary for a particular foreign language document.

At the left side of page is a bigger TextArea which contains the accessed document.

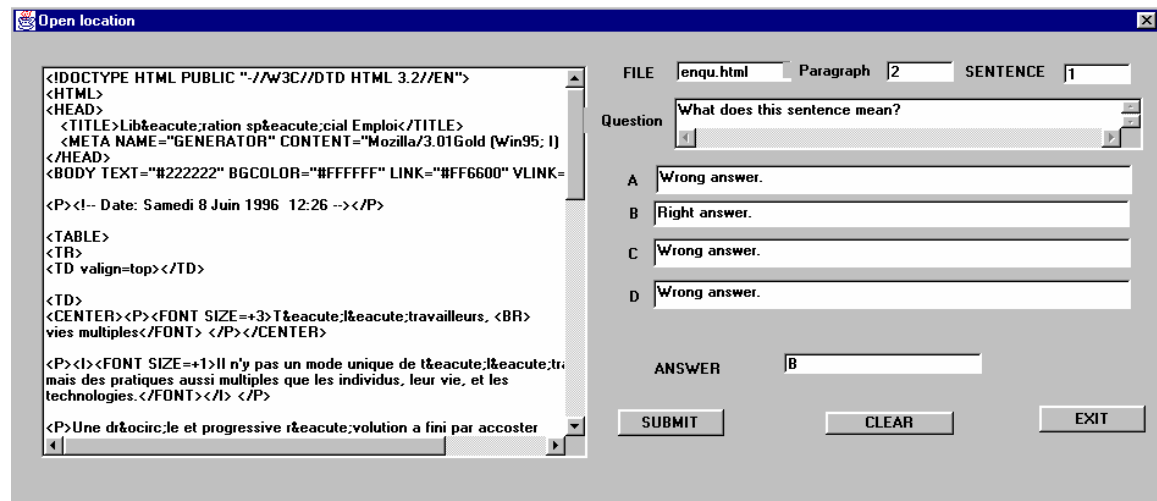


There are a “→” button and a TextField called “Word” at the first line of the page’s right side. The “Word” TextField is used to contain a tough word which the author wants to explain. The word can either be typed by author, or transferred from the large textArea using the “→” button.

There is another bigger TextArea called “Definition” on the next line of the right side. The “Definition” field is used to contain the definition of the tough word, input by the author.

There are three buttons at the bottom of the right side: “SUBMIT”, “CLEAR”, and “EXIT”. The “SUBMIT” button is used to submit the word and associated definition input by the author. the “CLEAR” button is used to clear the “Word” field and “Definition” field. The “EXIT” button is to exit this page.

#### 6.5.4 “MC” page



**Figure 6.5 “MC” page of the interface**

The “MC” page is used to let the author input multiple choice questions for checking comprehension of certain sentences in the foreign language document.

At the left side of the page, there is a TextArea field which is used to contain the content of the accessed document.

At the first line of the page's right side, there are three TextField fields: "File", "Paragraph", and "Sentence". The "File" textField is used to contain the name of the file being processed and is provided by the program automatically. The "Paragraph" and "Sentence" fields are used to let the author input the paragraph number and sentence number as integers, which indicate the position of the particular sentence associated with the multiple choice question being input by the author.

There is a TextField field called "Question" at the next line of the right side. It is used to contain the question part of the multiple choice item.

On the following four lines, there are four TextField fields which are used to contain the four answers of the multiple choice item.

After the four "Choice" Textfield fields, there is a Textfield called the "Answer" field. It is used to let the author input which choice is right (in the format of single character, such as A, B, C, or D)

There are three buttons at the bottom of the right side: "SUBMIT", "CLEAR", and "EXIT". The "SUBMIT" button is used to submit all the information input by the author. The "CLEAR" button is used to clear all the input fields. The "EXIT" button is used to let the author exit this page.

## **Chapter 7 Maintenance Phase**

Once the client has accepted the Study Guide Authoring System, any changes made to the product fall under the maintenance phase. After the client begins using the product, s/he will most likely discover various changes that would enhance the functionality of the product. The enhanced functionality would cause the product to be a more useful tool.

Given that the product has already been accepted, the maintenance phase is not grudgingly carried out but rather is an integral part of the software life cycle. The developers can reduce the amount of effort required for the improved functionality by developing flexible or extensible code; furthermore, a good software engineer will keep a constant eye on possible future enhancements (Nelson, 1995).

## **7.1 The Limitations of the Study Guide Authoring System**

As with any system, there are some problems and limitations regarding the Study Guide Authoring System.

### **7.1.1 CGI problem**

This problem is relates to the "Annotated Foreign Language Document Repository" which consists of the output pages of the "Study Guide Authoring System".

I used some CGI programming in the repository design. The function of the CGI program is to transfer the text typed by a student in a type-in window in one page to other pages. This CGI module is stand-alone and is called automatically when the student is studying, as needed..

Most Web servers are preconfigured to use CGI programs. There are generally two things that tell a server whether a file is a CGI application or not:

- A) A designated directory. Some servers enable you to specify that all files in a designated directory (usually, by default, called cgi-bin) are CGI programs.
- B) Filename extensions. Many servers are preconfigured to interpret all files ending in .cgi as CGI programs.

So, before installing the "Annotated Foreign Language Document Repository", the user must consult server documentation for instructions about how her or his server is configured for CGI programs. Right now, my CGI program uses the .cgi suffix method.

I noticed the CGI problem almost at the end of the programming of the whole system. It may cause under certain circumstances. So I recommend that this function be reimplemented in another technique, such as Java applet, in future work.

### **7.1.2 The Multiple User problem**

This problem is associated with the previous problem. When a student studies a foreign language document in the "Annotated Foreign Language Document Repository", s/he will be asked to type some words on the blank textarea in some of the pages for some of the steps. After submitting by the student, the system needs to transmit this typed-in text to other steps related to studying this particular document. Currently, what I am doing is using a CGI program associated with an html form. When a student writes down some words and clicks the submit button, the CGI program will write this text into the textareas of others page. When the student works on one of those other pages, s/he will see the text, which s/he wrote earlier. Unfortunately, if different users study the same foreign language document in the repository simultaneously via the Internet, the system will have a problem. Based on the current system, it cannot support multiple users working on the same document at the same time. Actually, it is not the problem with the Study Guide Authoring System. It is the problem with the "Annotated Foreign Language Document Repository". The text that one user types in might appear in a page viewed by the other simultaneous user. I recommend that we use database technique to solve this in the future work. The current problem is due to the CGI program, in the files transferSt3.cgi and transferSt8.cgi.

## **7.2 Future development of the Study Guide Authoring System**

### **The improvement of the multiple choice question capability**

In this version of the Study Guide Reading System, when students work on the different multiple questions, they get the same feedback, such as “Wrong Guess”, “Right Guess”. This obviously does not help student greatly in understanding the sentence. So, in the future, the system should add another function which can let authors provide explanations to accompany the multiple choice questions, so that a wrong answer comes with some explanation. To add that function, we can just add two text boxes at “MC” page screen (See figure 6.5), which can let author type in the explanations for both right and wrong answers of this particular multiple choice question, and then save them in the multiple-choice question data file using the same method currently used in saving the other parts of multiple choice question.

### **Processing multiple languages**

The first version of the Study Guide Reading System can process only French language documents in html format; however, the system is supposed to be able to process three foreign languages (French, German and Spanish) in either html format or plain text format. So, in the future, the system will be extended by adding additional classes to realize those functions.

### **Setting up the Database**

I already mentioned this in a previous section. Currently, I just save the processed foreign language documents into a repository. We call this the "Annotated Foreign Language Document Repository". If we have few foreign languages documents in it, it works fine except for the multiple-user problem. But, we will eventually store a lot of different kinds of foreign languages in it. So if we want to let the "Annotated Foreign Language Document

Repository" work efficiently, we must use other techniques to store the processed documents. I recommend that we use a existing popular database, such as MS SQL server to store those processed files. I also suggest solving the multiple-user problem based on database technique.

### **7.3 Conclusion**

The Annotated Foreign Language Document Repository is a useful system to allow students to learn foreign languages, and the Study Guide Authoring System is an efficient way to construct the Annotated Foreign Language Document Repository. This thesis is just to realize the basic function of the Study Guide Authoring System and construct the basic structure of the Annotated Foreign Language Document Repository. This work is the foundation of the whole project, and provides a good platform for future work. That is because the Study Guide Authoring System uses OOD technique, so the other advanced functions can be easily integrated into the whole system.

## REFERENCE

- |                       |   |      |
|-----------------------|---|------|
| [1] Kevin Hughes      | <<A Guide to Cyberspace>>   | 1993 |
| [2] James Davis       | <<TIG Proposal>>  | 1996 |
| [3] Tomas Green       | <<Development of Interactive, Multimedia Educational Software for Studying Native American Cultural History and Foreign Language Learning>> | 1996 |
| [4] Kathy Kozell      | <<"Crafting the User Experience" in Multimedia Producer>>   | 1995 |
| [5] Mary E. S. Morris | <<HTML for Fun and Profit>>   | 1995 |
| [6] David Flanagan    | <<Java In A Nutshell>> Second Version   | 1997 |
| [7] Marry Vampione    | <<The Java Tutorial>>   | 1997 |
| [8] Eugene Eric Kim   | <<CGI Developer's Guide>>   | 1996 |
| [9] Nikhat Fatima     | <<HyperBrowser: Information customization with automatic generation of hyperlinks>>   | 1996 |
| [10] Nielson, Jakob.  | <<Hypertext and Hypermedia>>  | 1990 |
| [11] Seyer, Philip    | <<Understanding Hypertext: concepts and applications>>  | 1991 |
| [12] Salton Gerard    | <<Automatic Text Processing>>   | 1989 |
| [13] Schach Stephen   | <<Software Engineering>>  | 1993 |



## APPEDIX A Modules of Study Guide Authoring System

<b>Class Name</b>	<b>AdvanceDialog</b>
Module Name	AdvanceDialog
Module Type	constructor method
Input argument	container
Output argument	none
Error message	none
Files access	none
Files changed	none
Methods invoked	none
Narrative	Constructs a new AdvanceDialog associated with parent Container.

Module Name	CreateControls
Module Type	public Boolean method
Input argument	none
Output argument	Boolean
Error message	none
Files access	none
Methods invoked	none
Narrative	Initiates the controls on the AdvanceDialog

<b>Class Name</b>	<b>AdvanceFrame</b>
Module Name	AdvanceFrame
Module Type	constructor method
Input argument	Frame
Output argument	none
Error message	none
Files access	none
Methods invoked	UserConstants.UserConstants, DictionaryFrame.DictionaryFrame, MultiChoiceFrame.MultiChoiceFrame, AdvanceDialog.AdvanceDialog,
Narrative	Constructs a new AdvanceFrame and initiates the private variable in the
	AdvanceFrame class

Module Name	CreateAdvanceFrame
Module Type	public void
Input argument	none
Output argument	none
Error message	none
Files access	none
Methods invoked	none
Narrative	Creates the control on the AdvanceDialog.

Module Name	HandleEvent
Module Type	public Boolean
Input argument	Event
Output argument	Boolean
Error message	none
Files access	none
Methods invoked	none
Narrative	This method is called when any event occurs inside AdvanceFrame. The method return true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to the AdvanceFrame's parent: Dialog;

Module Name	getPage2FileName, getPage7Name
Module Type	public String
Input argument	none
Output argument	String
Error message	none
Files access	none
Methods invoked	none
Narrative	Methods to query the page2 and page7 filename.

Module Name	getTitleSen
Module Type	public int
Input argument	none
Output argument	int
Error message	none
Files access	none
Methods invoked	none
Narrative	Method to query the sentence number for description title.

### **Class CommonControls**

Module Name	appendFileToFile
Module Type	protected method
Input argument	string, string
Output argument	none
Error message	Displays the exception message, when the file can not be opened.
Files access	Step page's name which follows the import argument.
Methods invoked	RandomAccessFile

Narrative	Inserts a file into another file.
Module Name	appendLinkedListToFile
Module Type	protected method
Input argument	string, LinkedList
Output argument	none
Error message	Displays the exception message, when the file can not be opened.
Files access	Step page's name which follows the import argument.
Methods invoked	RandomAccessFile, LinkedList
Narrative	Inserts a linked list into a data file at the position which has <!IPUTHERE> mark.
Module Name	appendLinkedListToLinkedList
Module Type	protected method
Input argument	LinkedList, LinkedList
Output argument	none
Error message	none.
Files access	none
Methods invoked	LinkedList
Narrative	Appends a LinkedList to another LinkedList.
Module Name	CommonControls
Module Type	constructor
Input argument	String, LinkedList
Output argument	none
Error message	none
Files access	none
Methods invoked	LinkedList
Narrative	Creates 5 new linked list relative with title, subtitle, word, sentence, paragraph, and clone title, subtitle, word with import argument.
Module Name	CopyPageToFileDir
Module Type	protected method
Input argument	string, string
Output argument	none
Error message	Displays the exception message, when the file can not be opened.
Files access	The name of source file like the first import argument.
Methods invoked	FileInputStream, FileOutputStream
Narrative	Copy a data file to a special subdirectory.
Module Name	getParagraph
Module Type	protected LinkedList
Input argument	int, LinkedList
Output argument	LinkedList
Error message	Displays the exception message, when the No. of the paragraph does not
Files access	exist.
Methods invoked	none
Narrative	LinkedList Inserts a file into another file.

### **Class DicDialog**

Module Name	CreateControls
Module Type	public Boolean method
Input argument	none
Output argument	Boolean
Error message	none
Files access	none
Methods invoked	none
Narrative	Initiates the controls on the DicDialog

Module Name	DicDialog
Module Type	constructor
Input argument	container
Output argument	none
Error message	none
Files access	none
Methods invoked	DialogLayout.DialogLayout, Container;
Narrative	Passes the parent container in.

### **Class Dictionary**

Module Name	Dictionary
Module Type	constructor
Input argument	none
Output argument	none
Error message	none
Files access	none
Methods invoked	none
Narrative	No argument Dictionary constructor

Module Name	makeDictionary
Module Type	public mehtod
Input argument	LinkedList, String
Output argument	LinkedList
Error message	none
Files access	none
Class invoked	LinkedList;
Narrative	Makes dictionary LinkedList.

### **Class DictionaryFrame**

Module Name	CreateFrame
Module Type	public method
Input argument	none
Output argument	none
Error message	none
Files access	none
Methods invoked	DicDialog.CreateControl
Narrative	Creates the control on the DicDialog

Module Name	DictionaryFrame
Module Type	constructor

Input argument	Frame
Output argument	none
Error message	none
Files access	none
Methods invoked	DicDialog.DicDialog
Narrative	Initiates a instance of DicDialog, and a instance of the based Frame. Passes the Frame caption to based Frame.
Module Name	handleEvent
Module Type	public Boolean method
Input argument	Event
Output argument	Boolean
Error message	none
Files access	none
Methods invoked	none
Narrative	This method is called when any event occurs inside DicDialog. The method return true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to the parent Dialog of the DicFrame;
Module Name	writeBufferToFile
Module Type	public method
Input argument	none
Output argument	none
Error message	Display Exceptions message, when the file doesn't exist.
Files access	Dictionary.html
Method invoked	none
Narrative	Appends StringBuffer which contains the information about word and its definition into Dictionary.html.
Module Name	writeWordToBuffer
Module Type	public method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	When the user inputs a word and associated definition, this method can insert the <A Name ="word"> </A> anchor at the front of the word. Meanwhile, it can replace the "/n" with   at the end of each line of definition.
<b>Class FileList</b>	
Module Name	appendNameToFileList
Module Type	public method
Input argument	String, String

Output argument	none
Error message	Display Exceptions message, when the file doesn't exist.
Files access	filelist.list
Method invoked	none
Narrative	Save the filename into filelist.dat file.
Module Name	existedFile
Module Type	public Boolean mehtod
Input argument	none
Output argument	Boolean
Error message	none
Files access	none
Method invoked	none
Narrative	Decides whether the file already has existed in database. If the file does
	not exist, it returns true. Otherwise, it returns false.
Module Name	FileList
Module Type	constructor
Input argument	String
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Initiates the FileList class, and passes the filename in.
Module Name	loadFile
Module Type	public method
Input argument	String
Output argument	none
Error message	Display the wrong message, if the file doesnot exist.
Files access	filelist.dat
Method invoked	none
Narrative	Loads the filelist.dat from hard drive, saves the data into string, and splits string into tokens according the filename.

### **Class FileStructure**

Module Name	cutFileHead
Module Type	public Boolean mehtod
Input argument	String
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Removes the data from <head> to </head> out of the string which stores
	data of document.
Module Name	FileStructure

Module Type	constructor
Input argument	String, String. String
Output argument	none
Error message	none
Files access	none
Method invoked	LinkedList
Narrative	Initiates three linked list which will contain title, subtitle, content of document. Pass the title, subtitle and content of document in.

Module Name	getSubTitleNode, getTitleNode, getWordNode
Module Type	public method
Input argument	none
Output argument	LinkedList
Error message	none
Files access	none
Method invoked	none
Narrative	To query the linked list of title, subtitle, content of document.

Module Name	parseFile
Module Type	public mehtod
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	removePart, removeTag, setNode, cutFileHead
Narrative	Removes the tile, subtitle , file header from the file, if title and subtitle exist, and saves the content of file into wordNode linked list. Removes the <center>,</center>,<font>,</font> tags from title or subtitle.

Module Name	removePart
Module Type	public mehtod
Input argument	String, String
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Removes the second import string from the first import string.

Module Name	removeTag
Module Type	public mehtod
Input argument	String
Output argument	Boolean
Error message	none
Files access	none
Method invoked	none
Narrative	Removes the html tag from the import string.

Module Name	setNode
-------------	---------

Module Type	public mehtod
Input argument	StringTokenizer, String
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Passes all the token stored in the StringTokenizer into a linkedlist.

### **Class Link**

Module Name	Link
Module Type	constructor
Input argument	Object, Link
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Defines the basic properties of the linked list.

### **Class LinkedList**

Module Name	append
Module Type	pubic method
Input argument	Object
Output argument	none
Error message	none
Files access	none
Method invoked	Link.
Narrative	Inserts the Object after the tail of the list
Module Name	clone
Module Type	public method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Duplicates one instance of LinkedList to another Linkedlist.

Module Name	currentElement()
Module Type	public Object method
Input argument	none
Output argument	Object
Error message	none
Files access	none
Method invoked	none
Narrative	Returns the current element under the cursor.

Module Name	cursor
Module Type	private method
Input argument	none
Output argument	none
Error message	none



Files access	none
Method invoked	none
Narrative	Returns element under current cursor, if the pervious element is not null.
	Otherwise, returns the head of linked list.
Module Name	elements
Module Type	public method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Duplicates one instance of LinkedList to another Linkedlist.
Module Name	hasMoreElements
Module Type	public method
Input argument	none
Output argument	Boolean
Error message	none
Files access	none
Method invoked	cursor
Narrative	If cursor() is not null, returns true. Otherwise, returns false.
Module Name	insert
Module Type	public method
Input argument	Object
Output argument	none
Error message	none
Files access	none
Method invoked	Link.link, cursor
Narrative	Inserts Object before the iterator position
Module Name	nextElement
Module Type	public Object method
Input argument	none
Output argument	Object
Error message	Displays exception java.util.NoSuchElementException if already at the
	end of the list.
Files access	none
Method invoked	none
Narrative	Moves the cursor to the next position. Returns the current element (before advancing the position)
Module Name	remove
Module Type	public Object method
Input argument	none
Output argument	Object
Error message	none

Files access	none
Method invoked	cursor;
Narrative	Removes the element under the cursor, return the removed element

Module Name	reset
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Put the cursor at the beginning of the linked list.

Module Name	size
Module Type	public int method
Input argument	none
Output argument	int
Error message	none
Files access	none
Method invoked	none
Narrative	Returns the number of the elements in the linked list.

### **Class ListEnumeration**

Module Name	hasMoreElements
Module Type	public Boolean method
Input argument	none
Output argument	Boolean
Error message	none
Files access	none
Method invoked	none
Narrative	Returns true if the iterator is not at the end if the list.

Module Name	ListEnumeration
Module Type	constructor
Input argument	Link
Output argument	none
Error message	none
Files access	none
Method invoked	Link
Narrative	Pass a Link node to the class.

Module Name	nextElement
Module Type	public Object method
Input argument	none
Output argument	Object
Error message	none
Files access	none
Class invoked	cursor;
Narrative	Moves the iterator to the next position. Returns the current element. (before advancing the position)

### **Class LocationDialog**

Module Name	CreateControls
Module Type	public Boolean method
Input argument	none
Output argument	Boolean
Error message	none
Files access	none
Methods invoked	none
Narrative	Initiates the controls on the LocationDialog

Module Name	LocationDialog
Module Type	constructor
Input argument	container
Output argument	none
Error message	none
Files access	none
Methods invoked	DialogLayout.DialogLayout, Container;
Narrative	Passes the parent container in.

### **Class MainDialog**

Module Name	CreateControls
Module Type	public Boolean method
Input argument	none
Output argument	Boolean
Error message	none
Files access	none
Methods invoked	none
Narrative	Initiates the controls on the MainDialog

Module Name	MainDialog
Module Type	constructor
Input argument	container
Output argument	none
Error message	none
Files access	none
Methods invoked	DialogLayout.DialogLayout, Container;
Narrative	Passes the parent container in MainDialog.

### **Class MainFrame**

Module Name	action
Module Type	public Boolean methods
Input argument	Event, Object
Output argument	Boolean
Error message	none
Files access	original document
Class invoked	LocationFrame, ReadURL, FileDialog, FileStruceture,
AdvanceFrame,	ParseFileName, LocationDialog, FileList, ProduceHtmlPage

Narrative	This method is called when any event occurs inside MainDialog such
as:	open file from either local or web side; choice which kind of
language,	select advance choice, produce html page and so on. The method
return	true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to the parent Dialog.
Module Name	addCOMBOItem
Module Type	public method
Input argument	MainDialog, MainMenu
Output argument	none
Error message	none
Files access	none
Class invoked	MainDialog, MainMenu
Narrative	Adds the relative control on the MainDialog, and passed current MainMenu and MainDialog into local class;
Module Name	appendFileToFrchList
Module Type	public void method
Input argument	String
Output argument	none
Error message	Displays exception message, if the file does not exist.
Files access	FrchFileList.html
Method invoked	none
Narrative	Add the filename to the FrchFileList.html
Module Name	getBuffer
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none
Narrative	Provides a easy way by which the other class can query the string which stores the document.
Module Name	getFileDir
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none
Narrative	Provides a easy way by which the other class can query the string which stores the current file directory.
Module Name	getFileList

Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none
Narrative	Provides a easy way by which the other class can query the string which stores the current filename list.
Module Name	getFullFileName
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none
Narrative	Provides a easy way by which the other class can query the string which stores the current file Full Name (include extension name)
Module Name	getSubTitleName
Module Type	public LinkedList method
Input argument	none
Output argument	LinkedList
Error message	none
Files access	none
Method invoked	LinkedList.LinkedList
Narrative	Provides a easy way by which the other class can query the linked list which stores the subtitle of document
Module Name	getSubTitleText
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none
Narrative	Provides a easy way by which the other class can query the string which stores the current subtitle of document
Module Name	getTitleList
Module Type	public LinkedList method
Input argument	none
Output argument	LinkedList
Error message	none
Files access	none
Method invoked	Linkedlist.LinkedList
Narrative	Provides a easy way by which the other class can query the linked list which stores the current title of document

Module Name	getTitleText
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none
Narrative	Provides a easy way by which the other class can query the string which stores the current title of document
Module Name	handleEvent
Module Type	public Boolean method
Input argument	Event
Output argument	Boolean
Error message	none
Files access	none
Method invoked	none
Narrative	Accepts the even to shut down the program.
Module Name	MainFrame
Module Type	constructor
Input argument	String
Output argument	none
Error message	none
Files access	none
Class invoked	FileDialog, AdvanceFrame, DictionaryFrame, LocationFrame, LinkedList,
Narrative	Initiates a instance of FileDialog, DictionaryFrame, LocationFrame, and three instance of LinkedList. Set up the brackground properties.
Module Name	readFile
Module Type	public Void method
Input argument	String
Output argument	none
Error message	Display exception message, if the does not existed.
Files access	Processed document whose name is the same as the imported string.
Method invoked	none
Narrative	Reads a file and saves the file content into a string.
Module Name	showFilefromLocatonFrame
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Class invoked	LocationFrame
Narrative	Reads document from web address by LocationFrame.

### **Class MainMenu**

Module Name	CreateMenu
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Class invoked	none
Narrative	Initiates the controls on the mainmenu.

Module Name	MainMenu
Module Type	constructor
Input argument	Frame
Output argument	none
Error message	none
Files access	none
Class invoked	none
Narrative	Passes the properties of the parent Frame to the MainMenu class.

### **Class MultiChoiceDialog**

Module Name	CreateControls
Module Type	public Boolean method
Input argument	none
Output argument	Boolean
Error message	none
Files access	none
Methods invoked	none
Narrative	Initiates the controls on the MultiChoiceDialog

Module Name	MultiChoiceDialog
Module Type	constructor
Input argument	container
Output argument	none
Error message	none
Files access	none
Methods invoked	DialogLayout.DialogLayout, Container;
Narrative	Passes the properties of parent container in.

### **Class MultiChoiceFrame**

Module Name	clearQuery
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Methods invoked	MultiChoiceDialog
Narrative	Resets the 5 textbox on the MultichoiceDialog.

Module Name	CreateMultiChoiceFrame
Module Type	public void method
Input argument	none

Output argument	none
Error message	none
Files access	none
Methods invoked	MultiChoiceDialog.CreateControl
Narrative	Creates the controls on the MultiChoiceDialog
Module Name	getTextFromFrame
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Class invoked	MultiChoiceDialog;
Narrative	Stores the information from the controls of the MultiChoiceFrame
into a	string.

Module Name	handleEvent
Module Type	public Boolean method
Input argument	Event
Output argument	Boolean
Error message	none
Files access	none
Methods invoked	getTextFromFrame, clearQuery, writeWordTouffer,
writeBufferToFile	
Narrative	This method is called when any event occurs inside
MultihoiceDialog.	The method return true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to the parent Dialog.

Module Name	MultiChoiceFrame
Module Type	constructor
Input argument	Frame
Output argument	none
Error message	none
Files access	none
Methods invoked	MultiChoiceDialog.MultiChoiceDialog
Narrative	Initiates a instance of MultiChoiceDialog, and a instance of the based Frame, Passes the Frame caption to based Frame.

Module Name	setChoiceTag
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Methods invoked	none



Narrative	Sets the #RightAnswer or # WrongAnser mark before very choice. So, when students click the choice, they can get the right feedback.
Module Name	writeBufferToFile
Module Type	public void method
Input argument	none
Output argument	none
Error message	Display Exceptions message, when the file doesn't exist.
Files access	MultiChoice.html
Method invoked	none
Narrative	Appends StringBuffer which contains the information about question, its four choices, and explanation into MultiChoice.html.
Module Name	writeWordToBuffer
Module Type	public method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	When the user inputs a question, associated choices and explanation, this
of	method can insert the <A Name ="PnSn"> </A> anchor at the front
	the question.(P stands for which paragraph, S stands for which sentences). Meanwhile, it can replace the “/n” with   at the end of each line of choices and explanation.
<b>Class PageStep10</b>	
Module Name	buildStep10Page
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp10.html, appends the step10 explaining page to head page, processes the step10's linked list, inserts the linked list to FrnchChallStp10.html.
Module Name	PageStep10
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols

Narrative list	Creates a instance of the based class, Commoncontrols, initiates a linked which can store step10's information.
----------------	---

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step10 into a linked list.

### **Class PageStep1A**

Module Name	buildStep1APage
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp1A.html, appends the step1A explaining page to head page, processes the step1A's linked list, inserts the linked list to FrnchChallStp10.html.

Module Name	PageStep1A
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative list	Creates a instance of the based class, Commoncontrols, initiates a linked which can store step1A's information.

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step1A into a linked list.

### **Class PageStep1B**

Module Name	buildStep1BPage
-------------	-----------------

Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp1B.html, appends the step1B explaining page to head page, processes the step1B's linked list, inserts the linked list to FrnchChallStp1B.html.
Module Name	PageStep1B
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list
	which can store step1B's information.
Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step1B into a linked list.
<b>Class PageStep2A</b>	
Module Name	buildStep2APage
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp2A.html, appends the step2A explaining page to head page, processes the step2A's linked list, inserts the linked list to FrnchChallStp2A.html.
Module Name	PageStep2A
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList

Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list
	which can store step2A's information.

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step2A into a linked list.

### **Class PageStep2B**

Module Name	buildStep2BPage
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp2B.html, appends the step2B explaining page to head page, processes the step2B's linked list, inserts the linked list to FrnchChallStp2B.html.

Module Name	PageStep2B
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list
	which can store step2B's information.

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none

Narrative	Stores the information for step2B into a linked list.
-----------	---

### **Class PageStep3**

Module Name	buildStep3Page
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile

Narrative	Copy the head page to current directory and renames to FrnchChallStp3.html, appends the step3 explaining page to head page, processes the step3's linked list, inserts the linked list to FrnchChallStp3.html.
-----------	--

Module Name	PageStep3
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list
	which can store step3's information.

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step3 into a linked list.

### **Class PageStep4**

Module Name	buildStep4Page
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile

Narrative	Copy the head page to current directory and renames to
-----------	--

FrnchChallStp4.html, appends the step4 explaining page to head page, processes the step4's linked list, inserts the linked list to FrnchChallStp4.html.

Module Name	PageStep4
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list

which can store step4's information.

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step4 into a linked list.

### **Class PageStep10**

Module Name	buildStep5Page
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp5.html, appends the step5 explaining page to head page, processes the step5's linked list, inserts the linked list to FrnchChallStp5.html.

Module Name	PageStep5
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list

which can store step5's information.

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step5 into a linked list.

### **Class PageStep6**

Module Name	buildStep6Page
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp6.html, appends the step6 explaining page to head page, processes the step6's linked list, inserts the linked list to FrnchChallStp6.html.

Module Name	PageStep6
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list
	which can store step6's information.

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step6 into a linked list.

### **Class PageStep7A**

Module Name	buildStep7APage
Module Type	public void method
Input argument	none
Output argument	none
Error message	none

Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp7A.html, appends the step7A explaining page to head page, processes the step7A's linked list, inserts the linked list to FrnchChallStp7A.html.
Module Name	PageStep7A
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list
	which can store step7A's information.
Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step7A into a linked list.
<b>Class PageStep7B</b>	
Module Name	buildStep7BPage
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp7B.html, appends the step10 explaining page to head page, processes the step7B's linked list, inserts the linked list to FrnchChallStp7B.html.
Module Name	PageStep7B
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols



Narrative list	Creates a instance of the based class, Commoncontrols, initiates a linked list which can store step7B's information.
----------------	--

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step7B into a linked list.

### **Class PageStep8**

Module Name	buildStep8Page
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile

Narrative	Copy the head page to current directory and renames to FrnchChallStp8.html, appends the step8 explaining page to head page, processes the step8's linked list, inserts the linked list to FrnchChallStp8.html.
-----------	--

Module Name	PageStep8
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols

Narrative list	Creates a instance of the based class, Commoncontrols, initiates a linked list which can store step8's information.
----------------	---

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step8 into a linked list.

### **Class PageStep9**

Module Name	buildStep9Page
Module Type	public void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrol.copyPageToFileDir, Commoncontrol.appendFileToFile, processFileLink, Commoncontrol.appendLinkedListToFile
Narrative	Copy the head page to current directory and renames to FrnchChallStp9.html, appends the step9 explaining page to head page, processes the step9's linked list, inserts the linked list to FrnchChallStp9.html.

Module Name	PageStep9
Module Type	constructor
Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	Commoncontrols.Commoncontrols
Narrative	Creates a instance of the based class, Commoncontrols, initiates a linked list
	which can store step9's information.

Module Name	processFileLink
Module Type	private void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Stores the information for step9 into a linked list.

### **Class ParseFileName**

Module Name	filename
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none
Narrative	Returns the full filename.

Module Name	fullFilename
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none

Narrative	Returns the full filename include path name.
Module Name	ParseFilename
Module Type	constructor
Input argument	String
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	Passes the filename into class, and define the path separator sign.

Module Name	path
Module Type	public String method
Input argument	none
Output argument	String
Error message	none
Files access	none
Method invoked	none
Narrative	Returns the path of the file name

### **Class ProduceHtmlPage**

Module Name	copyPageBackup
Module Type	public Void method
Input argument	String, String
Output argument	none
Error message	Displays the exception message, if the files can not be opened.
Files access	Two files whose name are the same as the import argument.
Class invoked	UserConstants
Narrative	Makes the backup file of the special file.

Module Name	copyPageToFileDir
Module Type	public void method
Input argument	String, String
Output argument	none
Error message	Display the error message, if the file does not exist.
Files access	The names of two files are the same as the import arguments.
Class invoked	UserConstants
Narrative	Copy the file to particular directory.

Module Name	makeProcess
Module Type	public Void method
Input argument	none
Output argument	none
Error message	none
Files access	none
Method invoked	none
Narrative	The major process of making all the html pages.

Module Name	ProduceHtmlPage
Module Type	constructor

Input argument	String, LinkedList, LinkedList, LinkedList
Output argument	none
Error message	none
Files access	none
Method invoked	LinkedList
Narrative	Initiates three instances of LinkedList, and passes the title linked list, subtitle linked list, content linked list into the class.

### **Class Study**

Module Name	main
Module Type	public void method
Input argument	String[]
Output argument	String
Error message	none
Files access	none
Class invoked	MainFrame, MainMenu
Narrative	The program entrance, and initiates the all controls of the MainFrame
and	MainMenu.