An Investigation into Teaching Description and Retrieval for Constructed Languages

A thesis presented in partial fulfilment of the requirements

for the degree of Master of Science in Computer Science at

Massey University

Son Hoang

2004

Abstract

The research presented in this thesis focuses on an investigation on teaching concepts for constructed languages, and the development of a teaching tool, called VISL, for teaching a specific constructed language.

Constructed languages have been developed for integration with computer systems to overcome ambiguities and complexities existing in natural language in information description and retrieval. Understanding and using properly these languages is one of the keys for successful use of these computer systems. Unfortunately, current teaching approaches are not suitable for users to learn features of those languages easily.

There are different types of constructed languages. Each has *specific features* adapted for specific uses but they have in common explicitly *constructed* grammar. In addition, a constructed language commonly embeds a powerful *query engine* that makes it easy for computer systems to search for correct information from descriptions following the conditions of the queries. This suggests new teaching principles that should be easily adaptable to teach any specific structured language's structures and its specific query engine.

In this research, teaching concepts were developed that offer a multi-modal approach to teach constructed languages and their specific query engines. These concepts are developed based on the efficiencies of language structure diagrams over the cumbersome and non-transparent nature of textual explanations, and advantages of active learning strategies in enhancing language understanding. These teaching concepts then were applied successfully for a constructed language, FSCL, as an example. The research also explains how the concepts developed can be adapted for other constructed languages.

Based on the developed concepts, a Computer Aided Language Learning (CALL) application called VISL is built to teach FSCL. The application is integrated as an extension module in PAC, the computer system using FSCL for description and retrieval of information in qualitative analysis. In this application, users will learn FSCL through an interconnection of four modes: FSCL structures through the first two modes and its specific query engine through the second two modes. After going through four modes, users will have developed full understanding for the language. This will help users to construct a consistent vocabulary database, produce descriptive sentences conducive to retrieval, and create appropriate query sentences for obtaining relevant search results.

Acknowledgements

There are a number of people I would like to thank for their support throughout this study.

First and most importantly, I would like to express a deep gratitude to my supervisor, Dr. Eva Heinrich for her encouragement, enthusiasm and invaluable advice. Without her tremendous help, this study would not be possible.

I also would like to thank my family for their continuing encouragement, support and for always being there for me.

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1

Introduction

The explosion of amount of information available in recent years has created an accompanying problem in that the large number of multimedia documents such as pictures, audios and videos has been a big challenge for information description and retrieval. This leads to requirements for creating new technologies to handle that huge volume of multimedia data. Christel et al. (1997, p23) claimed that "as digital video libraries emerge, users will need tools that handle effectively the dynamic nature of video; they will not likely invest hundreds of hours to find a single, relevant segment within a 1000-hour library". Voss (2001, p1) further confirmed that "A content provider like CNN has more than a hundred thousand hours of tape in its video archive—far too much for any human to view and annotate manually".

Natural language has widely been used for generating descriptive information as well as assisting information retrieval. However, natural language has significant drawbacks when confronted with large volumes of multimedia documents. The use of natural language in information description is significantly labour intensive (Sikora, 2001) and using of natural language for searching produces problems stemming from too many synonyms, too many meaning and spelling variations (Feldman, 1999). There are two kinds of context in which the use of natural language for descriptions may be not suitable:

- *Descriptions using in qualitative analysis in social psychology or education:* In this kind of analysis, descriptions are used to describe certain aspects of context from collected multimedia materials. The use of natural language causes difficulties for retrieving correct returns.
- *Multimedia descriptions for documents having structures:* In documents that describe multimedia contents, users need to retrieve not only entire the document content but also a single event or scene. In addition, users need to obtain very detailed information such as where, how, and when a single event should appear. When natural language is used for this kind of documents, users may get incorrect returns by using current searching techniques. In

addition, the use of natural language for this type of descriptions is time consuming.

With the requirements for new tools and new languages to handle, organise and retrieve multimedia data, many new languages are built and incorporated with computer applications to assist humans in describing and retrieving information of multimedia documents, as mentioned earlier. These languages are called *constructed languages* and can be presented in two different forms:

- *"Natural language like" languages:* These languages have many similar characteristics compared to natural language. Therefore, they are called "Natural language like" languages. These languages are used for qualitative analysis in social psychology or education and are commonly seen in studying behaviours. Examples of these languages are the Caber language (Patrick, 1985), Observer's language (Noldus Information Technology, 2003) or Flexible Structure Coding Language (FSCL - Heinrich, 1999).
- *Structured languages:* Descriptions of this kind of constructed languages are generated based on specific rules or schemas. The languages are used to describe documents that have structures such as multimedia documents like video clips or films. The most common language used for this purpose is the eXtensible Markup Language (XML - O'reilly Media, 2004).

The general purpose of these languages is to help users to attach notations or descriptions easily to multimedia documents and to retrieve correct information later on. When these descriptions are based on explicitly defined grammars, they can be easily analysed by computer systems. Therefore, the efficiency of searching information in a larger number of multimedia documents can be significantly increased.

To exploit possible advantages of constructed languages in describing and retrieving information, users have to understand the grammars of these constructed languages before using them. Currently, there is a shortage of studies and research that could provide an effective way to help users to understand those constructed languages. Therefore, the research presented in this thesis investigates issues concerning the development of learning tools to help users understand constructed languages and to facilitate the use of those languages in information description and retrieval.

This introductory chapter starts with a short review on the grammars of natural languages and constructed languages. The chapter then outlines the motivation and objectives of the research for learning constructed language grammars. Next, the chapter presents the methodology used in this research. The structure of this thesis is presented at the end of the chapter.

1.1 What is language grammar?

Language grammars can be defined into natural language and constructed language grammars. Main features of these language grammars are presented in the following sections.

1.1.1 Natural language grammar

The grammar of a language deals with the issues of how sentences are constructed called syntax. Different languages have different syntaxes. The syntax of a language includes two factors:

a. The orders of components such as subjects, verbs and objects in a sentence.

Different natural languages may have different orders of subjects, verbs and other components. For example, the orders of the Toba Batak language (an Austronesian language of Indonesia – Valin, 2001) are syntactically different from English language:

The teacher is reading a book.				*in English language*
Manjaha	buku	guru	i.	*in Toba Batak language*
Read	book	teacher	the	*the order of Toba Batak language compared to English language*

b. The combination of words into components.

A word group may be formed by several words and each word may be in a different lexical category. In the example above, "the teacher" is a word group called the noun group that plays the role of a subject. This word group includes an *article* category "the" and a *noun* category "book". Different languages have different word orders to form a word group. For example, the word order in a noun group in English and Vietnamese is different:

The	red	book	*in English language*
Mot	quyen sach	do	*in Vietnamese language*
The	book	red	*the order of Vietnamese language compared to English language*

With English language, there is one more important component called *morphology* that explains how a word is formed. For example, a word "student" has two forms: the singular form "student" and the plural form "students". Figure 1.1 presents relationships among different elements in an English sentence.

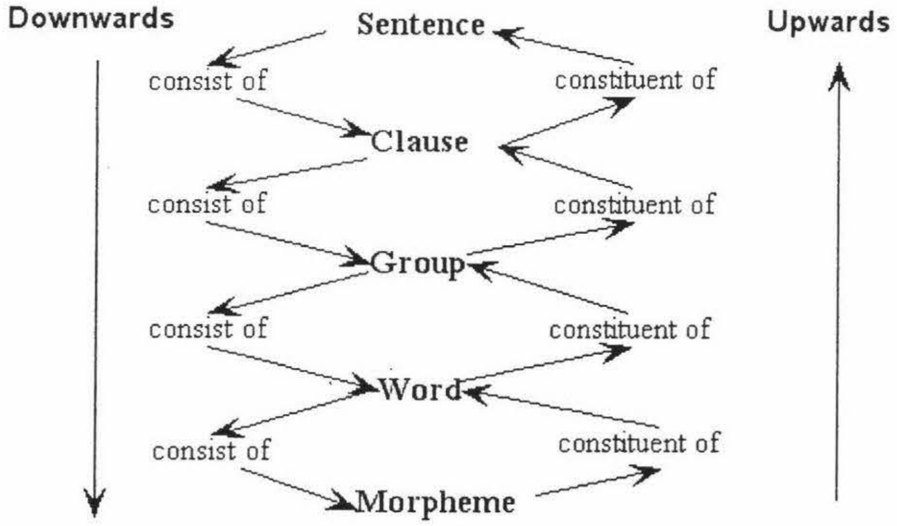


Figure 1.1 Relationship among elements in an English sentence

There are many natural languages as discussed above and each of them has specific structures. This research, from now on, will refer to the English language when using the term "natural language". In addition, the words "grammar", "structure", and "syntax" used in the research are equivalent.

1.1.2 Constructed language grammar

Constructed languages are created for incorporation with computer systems to help users to describe information easily and retrieve information correctly. In general, these languages' grammars are simplified from natural language or are structured so that the languages are logically precise and computationally tractable, but still human readable. Typical features of these languages can be summarised as the followings:

"Natural language like" languages (NLL languages): These languages have both similarities and differences compared to natural language. The differences lie on restricted vocabulary, formulated grammar and restricted structure:

- *Restricted vocabulary:* A restricted vocabulary is a list of words that is agreed on among members. Therefore, vocabulary size and meaning for a particular application domain are limited (Mitamura, 1999). In some languages, users can then organise words into hierarchies as in FSCL (Heinrich, 1999).
- *Formulated grammar:* In NLL languages, their grammars may be formulated explicitly such as LL based grammar in FSCL (Heinrich, 1999) or LR grammar in CABER language (Patrick, 1985). Documents or descriptions are written following rules set by the constructed language grammar. One important purpose of formulated grammars is to improve the performance of the retrieval process.
- *Restricted structure to reduce ambiguity:* In addition with a formulated grammar, NLL languages also employ structures that can reduce ambiguities compared with those of natural language. The reason for this is that "common" natural language structures can cause difficulties for computer application to "understand". For example, computer applications do not process the natural language accurately and efficiently when pronouns are used. Therefore, in NLL languages, pronouns are usually not encouraged (Mitamura, 1999; Pulman, 1996). A learner/user naturally understands what a pronoun stands for in a sentence. However computer applications may not know which noun this pronoun refers to.

Structured languages: Structured languages have different structures compared to natural language. These languages can be seen as "markup languages" as in XML.

Users normally have to study how to use these languages. For example, to use XML, users have to learn how to create correct XML documents following defined schemas. They also need to know how to search for information in a XML document. Simple learning tools can be developed that help users to understand basic XML structures or how to create XML schemas.

1.2 Motivation and objectives of the research

This section motivates the research by emphasising the importance of learning constructed languages, and by reviewing approaches that support learners in mastering language grammar. Following the motivation, the objectives of the research are outlined.

1.2.1 Learning constructed languages

Learning grammar is essential for learning a language. In a traditional sense, learning grammar is "the study of the principles which govern the formation and interpretation of words, phrases and sentences" (Radford, 1997, p1). To emphasise the importance of learning grammar, Lynn (2003, p1) wrote "think of English vocabulary as the bricks of the language and grammar as the mortar that hold those bricks together, without the mortar the bricks can come tumbling down! A lack of grammatical skills can cause embarrassing misunderstandings." However, many people do not want to learn grammar as it as boring and not meaningful. Experience shows that students have to do a lot of grammar exercises to learn a language but these exercises seem to have little or no meaning to them (Onestop Magazine, 2003).

Learning structures of NLL languages is important for users in view of describing and retrieving information. Even there are similarities of structures between natural language and these languages, improper understanding the specifics of these languages' structures can lead users to the problems like the following:

- *Generation of descriptive information that is difficult for querying:* Although there are advantages of NLL languages compared to natural language, they still have limitations. For example, users may create correct sentences following a language structure but it is hard to retrieve these sentences. Therefore, users should be aware of these limitations of NLL languages.

- *Misunderstanding of query engine leads to incorrect returns:* NLL languages usually employ specific query engines for searching for information. When creating improper queries, users may obtain redundant results or fail to detect descriptive sentences or documents that are appropriate to the users' search topic.

Learning structures is also essential for using structured languages, as their structures are totally different compared to natural language. Learning the structures of this kind of languages can be compared to learning the rules of a game. To be able to play a game, players must be familiar with the rules. If players do not know the rules, the game cannot be started or will soon collapse.

Therefore, to fully obtain advantages of constructed languages, users need to know the complete details of those structures. Even though different constructed languages may have different structures, there are some common principles that users are required to learn. The following sections outline important aspects that users should learn of constructed languages.

- *The structure of constructed languages*: Users must understand how elements of a constructed language can be combined into descriptions. For example, in FSCL, users must understand how words in different categories can be combined into word groups and how word groups can be integrated to create a sentence. With XML, users must understand how to create documents that are correct according the XML syntax, Data Type Definition (DTD), or XML schemas.
- *Query-matching mechanisms:* There are many different constructed languages used in many computer systems and many of these languages possess specific query engines. With NLL languages, the understanding of how the languages' query engines work will help users to set up suitable vocabularies and create appropriate descriptions to get best matches between queries and descriptions. With structured languages, users need to create correct queries to retrieve accurate returns.

There are several studies that focus on assisting users in generating correct descriptions using constructed languages. For example, there are some editor tools that help users to check the compliance of user input with the structure rules and assist them to write accurate descriptions (Schwitter et. al., 2003). In these editors, users have to stick with the predefined vocabulary and grammar rules to generate informative and consistent documents. However, these editors cannot help users to understand the query mechanisms that are embedded in many constructed languages. As a result, users may not discover the full potential of constructed languages for information retrieval.

To address these issues, the research presented here focuses on developing concepts to be applied in computer applications for teaching users constructed languages. The research then implements those concepts in a teaching tool for teaching a specific constructed language, FSCL, as a case study.

1.2.2 Factors that enhance understanding in language learning

There are several factors that can enhance users' understanding in language learning. The factors discussed in this section are: learning by using diagrams and computers, learning by example, and active learning.

Diagrams in language learning

Many authors of books or research on language theories use diagrams to present sentences structures. The benefits of diagrams in presenting language structures are, according to Ye Hedge School (2003), "Diagrams are unquestionably the best way to map the word relationships within a sentence; only when we can understand how words are related in a sentence can we study complex thought". Advantages of diagrams in enhancing users' understanding of language structures will be presented in more details in the following chapter.

Computer in language learning

Books and classroom tuition are traditional ways for learners to learn grammar. However, the invention of computers has leaded to an innovative method to teach language structures. Computer applications, called Computer Assisted Language Learning (CALL), offer powerful self-assessment facilities to learners. CALL can help learners to study many areas of languages such as grammar, vocabulary, reading and writing. As diagrams can be easily drawn by computer applications, they are used by many CALL applications as supporting tools to help users to learn natural languages. However, there are limitations of current research in exploring the advantages of diagrams in CALL applications for teaching *constructed language structures* and their specific query engines.

Learning by examples

Perry (1993) pointed out that command descriptions, syntax format and language references are not enough for new learners to learn a new language. He suggested that numerous examples could provide a better method to help users to understand the language. Addressing the same issue, Walsh (2001) emphasised that computer concepts are best taught with multiple examples and that using a variety of practical and real world examples can lead learners step-by-step on the road of learning. Pérez (2004) claimed that humans have great ability to learn concepts from a set of examples. Stauffer (2004) also concluded that learning by example allows learners to start with initial concepts (what they have known already) then understand new concepts (what they do not know yet) easily.

Active learning

Many researchers suggested that a passive learning environment in which students are involved only passively in learning, i.e., in listening to the instructor or reading the text book, generally leads to a limited retention of knowledge (McKeachie, 1998). In contrast, in active leaning, learners can obtain better knowledge as they not only merely listen or view the presentation of information but also work with the content and reflect on the process of learning (Oppermann, 2004). In an active learning environment, learners "are simply more likely to internalise, understand, and remember material learnt through active engagement in the learning process" (BYU Faculty Center, 2000, p.1). Active learning takes advantages of peoples' natural motivations, abilities and interests to solve problems, discover relationships and make comparisons. To learn language grammar, Eastwood (1999) claimed that the best way is to follow the active learning approach.

1.2.3 Research objectives

Following the motivations above, the main objectives of this research are:

- Investigate the advantages of using language structure diagrams for teaching *natural* languages.
- Identify the strategies of CALL applications to teach *natural* languages.
- Adjust the use of language structure diagrams and CALL strategies to the teaching of *constructed* languages and their queries engines.
- Implement these resulting concepts in a language-teaching tool for a specific constructed language.

1.3 Research methodology and steps

The research was carried out based on popular frameworks built by Galliers (1992) and Yin (1994). Using these frameworks, a research methodology that is suitable for fulfilling the stated research objectives has been developed:

Literature review: Examine diagramming techniques and CALL applications that assist language learning;

Concept development: Based on the literature review, develop concepts for teaching constructed languages;

Prototype implementation for a case study: Apply these concepts in an application for teaching a specific constructed language, FSCL;

Experiments to verify the concepts (due to time restrictions outside the scope of this thesis).

Applying this methodology leads to the following steps:

- Literature review, including:
  - Examining diagramming techniques using in natural language theories and constructed languages: The project will study how the diagramming techniques can support users in understanding language structures.
  - Investigating CALL applications for teaching natural language structures: The research will study features of CALL applications and their advantages for teaching constructed languages.
  - Inspecting in detail an existing constructed language, FSCL: The research examines the language structures, its structure diagrams and the query engine. This language will be used as a case study.

- Developing concepts to teach learners constructed languages using diagrams. These learning concepts are developed based on/adjusted from teaching concepts learnt from the literature review. The research employs these concepts for teaching FSCL and then outlines how those concepts can be adapted to teach other constructed languages.
- Implementing those concepts into a prototype application called Visualised Interactive Structure Learning (VISL) to teach FSCL. Specific technical issues relating to the development of this application will be presented in detail. The research will apply software engineering concepts using the Unified Modelling Language for analysis and design, and the Java programming language for implementation. VISL's interface will be developed based on principles of Human-Computer-Interaction (HCI).

1.4 Thesis structure

The thesis is divided into seven chapters. After the introduction in Chapter 1, Chapter 2 reviews the importance of diagrams in enhancing users' understanding of language structures. It also inspects typical CALL applications for teaching language structures. Chapter 3 examines a constructed language, FSCL, in detail. In Chapter 4, concepts for teaching constructed languages are developed. The chapter also illustrates how these concepts can be applied for teaching FSCL. Chapter 5 discusses the key issues relating to the development of a prototype CALL application called VISL for teaching FSCL. Chapter 6 presents the VISL development process through the three phases of analysis, design and implementation. Finally, Chapter 7 of the thesis summaries the contributions and highlights future topics arising from this research. The diagram in Figure 1.2 illustrates this structure of the thesis.

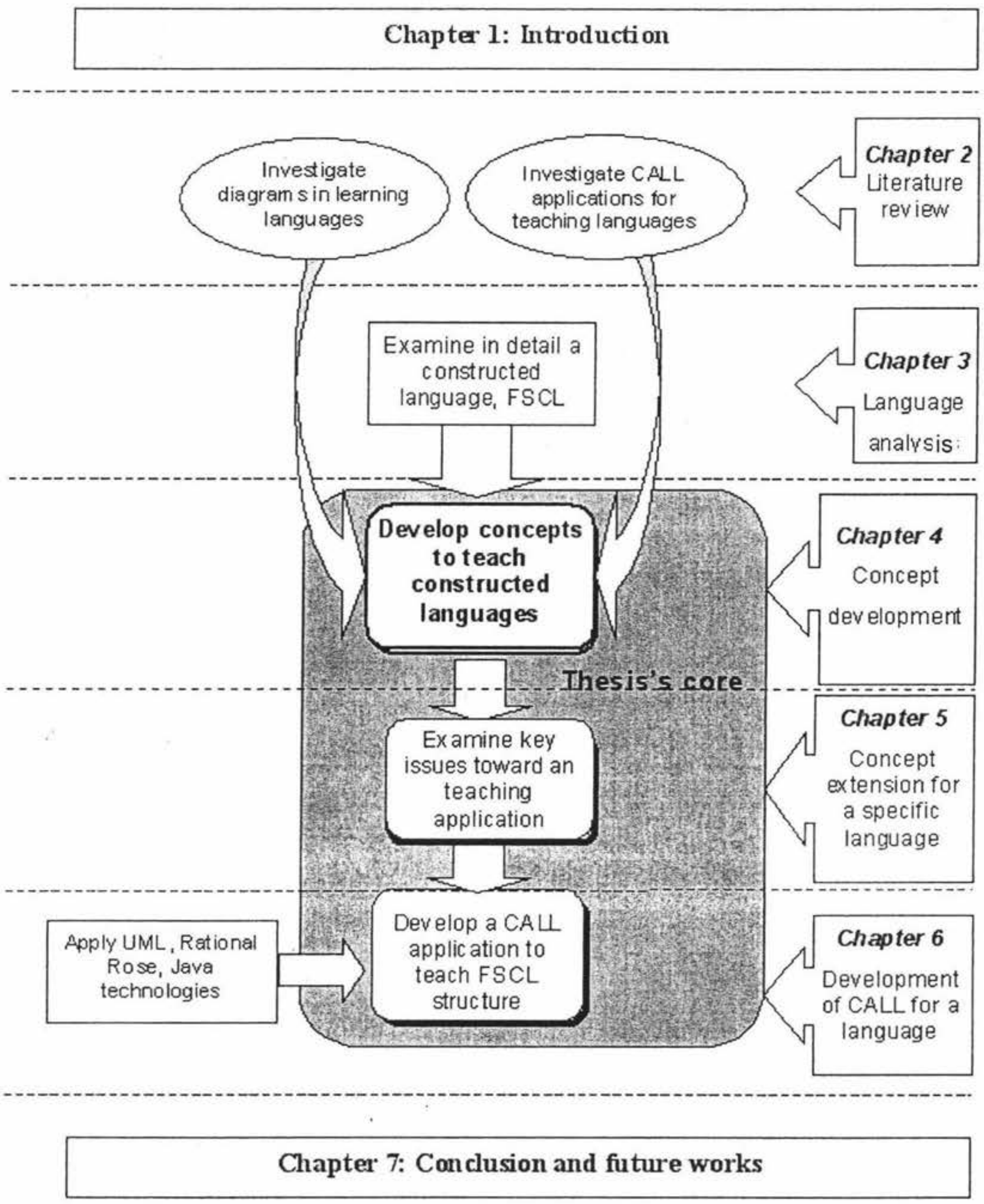

Chapter 1: Introduction
Investigate diagrams in learning languages
Investigate CALL applications for teaching languages
Chapter 2 Literature review
Examine in detail a constructed language, FSCL
Chapter 3 Language analysis
Develop concepts to teach constructed languages
Chapter 4 Concept development
Thesis's core
Examine key issues toward an teaching application
Chapter 5 Concept extension for a specific language
Apply UML, Rational Rose, Java technologies
Develop a CALL application to teach FSCL structure
Chapter 6 Development of CALL for a language
Chapter 7: Conclusion and future works

Figure 1.2 Thesis's structure diagram

Chapter 2

Language syntax learning and graphical presentation

This chapter first reviews languages used for description and retrieval of information including both natural language and constructed languages. The chapter next studies the efficiency of diagrams in presenting language structures for assisting learners to understand these language structures. The chapter then discusses how computer applications can explore the benefits of presenting diagrams in teaching natural language structures. Finally, the chapter outlines the principles drawn from those applications in teaching natural language and the possibilities of those principles in teaching constructed languages.

2.1 Language for description and retrieval of information

As briefly discussed in Chapter 1, the use of natural language is not suitable for describing and retrieving information from multimedia documents. That is because of the time consuming description process and the incorrect information achieved from the search process. Therefore, other languages were designed to overcome those limitations of natural language. One type of these languages are constructed languages. This section reviews the use of natural language and constructed languages and then discusses advantages of constructed languages compared to natural language in information description and retrieval.

2.1.1 Natural language in information description and retrieval

This part provides the main characteristics of natural language, and then presents its limitations in describing and retrieving information.

2.1.1.1 Natural language

Natural language is the language that people write and speak in everyday social communication. There are, of course, a variety of natural languages (like English, Vietnamese or German). The differences among natural languages lie in vocabulary or grammar. For example, in English language, a word "book" is written as "sach" in Vietnamese language.

As pointed out in Chapter 1, grammar in natural language includes two elements: morphology and syntax. Morphology is the way in which words are formed. For example, the plural form of the word "class" is the word "classes". Syntax, the second element in grammar, describes the way in which words fit together to form sentences or utterances. Fromkin and Rodman (1998, p28) defined that "Knowing a language includes the ability to construct phrase and sentences out of morphemes and words. The part of grammar that represents a speaker's knowledge of these structures and their formation is called syntax".

There are many English language theories. The differences among them are based on different definition of grammar or syntax for each theory. Several typical grammar theories are (Hudson, 2002):

- Phrase Structure Grammar (PSG);
- Function Grammar;
- Head-driven Phrase Structure Grammar (HPSG);
- Link Grammar;
- Systemic Functional Grammar;
- Lexical Functional Grammar;
- Word Grammar.

Although there are many categories of grammar, they all share some common elements. Following Hilferty (2002), the essential elements of grammar are word order, dependencies and categories.

- *Word order:* types of words and their ordering in regard to other elements in a phrase or sentence. When writing a sentence, a writer may have questions such as "Can an element of type X precede or follow (or both) an element of type Y".
- *Dependencies*: certain types of elements may need to co-occur (i.e., be combined in some way) with certain other elements (whether they be specific words or grammatical categories, etc.). This is known as valence.
- *Categories:* words are classified into categories called lexical categories (e.g., noun, verb, adjective, adverb, preposition, etc...).

2.1.1.2 Limitations of natural language in information description and retrieval

The advantages of natural language as a description language are "its expensiveness and flexibility, its immediate availability to every analyst with access to a word processing facility, and the familiarity with and understanding of natural language for every analyst" (Heinrich, 1999, p91). However, there are also some disadvantages when using natural language for both information description and retrieval. These disadvantages include lexical ambiguity, ambiguous sentence structure and complexity causing incorrect search results. The two example sentences below explain the ambiguity of using natural language in descriptive information:

1. The boy loves the dad more than the girl.

The sentence can be understood in two ways:

a. The boy loves the dad more than the girl loves the dad.

b. The boy loves the dad more than the boy loves the girl.

2. That house is created by a piece of glass and wood.

The sentence can be understood by two ways:

a. The house is created by a piece of both glass and wood.

b. The house is created by a piece of glass and with some wood.

The limitation of using of natural language in information description and retrieval is more obvious when computer systems are used to store and extract information. In this case, the richness, variance, and confusion in terminology of natural language cause difficulties in information retrieval (Fast et. al., 2002). Here are some significant drawbacks of using natural language in describing and retrieving information using computer systems:

- *Difficulty for computers to "understand" descriptions:* Defining the meaning of sentences is very difficult for a computer system while that meaning may be easy for a human reader to understand. The reason is that natural language processing is very complex for computer programs. Current computer applications are not able to analyse all sentences correctly (Smeaton, 1997).

This weakness of natural languages as a descriptive language is more obvious when it is used in computer systems to analyse behaviour (Heinrich, 1999).

- *Difficulty in retrieval:* Nowadays, information using natural language is stored in large volumes in computer systems. In these systems, searching techniques such as keywords and free texts are used widely. However, the use of natural language is not suitable for searching information. This is because users may get results in which descriptive information is not relevant to the search topics. Yang (2004, p1), pointed out that "Keyword search always results in low precision, poor recall, and slow response time due to the limitations of the indexing and communication methods (bandwidth)". The two examples explain the limitation of keyword search:
    - ➢ When a user wants to search documents relating to "children", he may use the word "child" to search. One can recognise that the user would miss the sentences/documents that use the word "infant" instead of "child".
    - ➢ When a user wants to search for a person named "John" doing an action "jump", he may use the query of a character sequence "John jumps" or the keyword string "John*jumps". If he uses the first query, he would miss the sentence "John runs and jumps". If the user utilises the query "John*jumps", he may obtain incorrect sentence "John runs and Lisa jumps".

Many researchers in computational linguistics have been attempting to address the problems of how to present English words, phrases, and sentences on a computing machine and how to encode information about sentence types, parts of speech and other grammatical information in a notation that a computer can digest (Dougherty, 1994). However, Microsoft, (2003, p1) pointed out that "Natural language, that is easiest for humans to learn and use, is hardest for a computer to master. Long after machines have proven capable of inverting large matrices with speed and grace, they still fail to master the basics of spoken and written languages. The challenges faced stem from the highly ambiguous nature of natural language".

To overcome those problems of natural language, constructed languages are invented. This type of languages is widely used in computer applications for the purpose of describing and retrieving information. Using these languages, users can extract the meaning of any sentence in an automated way. Therefore, the required information can be retrieved correctly and quickly through computer systems. Discussing the advantages of these languages, Schwitter et al., (2002, p1) pointed out that "Constructed languages not only make life a lot easier with technical documents... but they can also improve the whole knowledge acquisition process for any kind of intelligent system". The features of these languages are presented in the following section.

2.1.2 Constructed languages for information description and retrieval

Constructed languages for describing and retrieving information can be divided into two categories. One of these is called "natural language like" languages (NLL languages) used for qualitative analysis in social psychology or education that are commonly seen in studying behaviours. Example languages are Observer's language (Noldus Information Technology, 1998), CABER's language (Patrick, 1985), and the Flexible Structure Coding Language (Heinrich, 1999). The other category can be called *structured languages,* used to describe documents that have structures. One well-known representative of such languages is XML. The main features of these languages are discussed in detail in the following sections

2.1.2.1 NLL languages for studying behaviour

NLL languages have been developed for describing and retrieving information in qualitative research in social psychology or education. Compared with natural language, NLL languages share many common characters. The main differences between NLL languages and natural language can be seen in their grammar and vocabulary.

2.1.2.1.1 Simple grammar

A simple grammar of a NLL language is the grammar that is formulated (FSCL or CABER's language see a following section). As the grammar is controlled and formally specified, the sentences composed based on the grammar are not complex and ambiguous (Lehtola, et. al. 2003 and Mitamura, 1999). In addition, when documents are written following a set of rules that are standardized, computer systems can easily determine the structure of a sentence. Therefore, the "computer's understanding" of sentences or source texts can be improved. This leads to obtain or extract correct information from descriptions through search mechanisms.

2.1.2.1.2 Restricted vocabulary

Restricted vocabulary can be seen as the restriction of both vocabulary size and meaning for a particular application domain (Mitamura, 1999). For example, people studying the same topics can agree on a list of words used for describing or retrieving information. That list of words can be called restricted vocabulary. French et. al. (2001) and Mitamura (1999) pointed out that, compared to "normal" vocabulary, restricted vocabulary has the advantages to radically improve retrieval performance in a computer system. The reason for this is that users can define *equivalent words* and organise words into *hierarchies*.

- *Equivalent words:* Consider two descriptive sentences:
    1. Pandas like to live in large *areas*
    2. Pandas want to live in high *spaces*

  If a user inputs the keyword "*spaces*" for searching sentences which explain where Pandas want to live, he will miss the first sentence. If the words "area" and "space" are considered equivalent in a restricted vocabulary, the above problem can be avoided. In addition, a restricted vocabulary can help researchers to avoid misspellings or connect abbreviations to the full words (e.g. WS is understood as Wellington School). The benefits of searching information written by restricted vocabulary were discussed in more details in Fast, et al. (2002) and Lebanon Valley College (2000).
- *Hierarchy vocabulary:* By organising words into hierarchies, users can find information via relationships (French, 2002 and Heinrich, 1999). They then can construct detailed or aggregate levels of analysis. For example, in the case of a research study about behaviour of children in a classroom, researchers may want to know if boys or girls play games or read books during the intervals. They may also want to know if a girl named "Lisa" plays or reads. The first case constitutes an aggregate analysis and the second constitutes a detailed analysis. If the names of the children are organised into hierarchies, users can search information about "girls" by using keyword "girls" for the aggregate analysis. All descriptive sentences having the word "girls" or the

words below the level of that word in the hierarchy (e.g. Lisa, Joan or Jane) will be selected. Users can also search for an individual child (detailed analysis) such as "Lisa", "Joan" or "Jane". The hierarchy of a vocabulary can be presented as:

```
Kids
      boys
          Tommy
          Mike
      girls
          Lisa
          Jane
          Julia
          Joan
```

- *Use of pronouns is not encouraged:* The use of pronouns in natural language is very common. However, the use of pronouns is not always useful for retrieval of information. This is because computer applications may not know what noun is replaced by a pronoun. For example "Lisa helps Jane to open the book and then she starts to read the first chapter". A computer application may not know if "she" stands for "Lisa" or "Jane". For reason like this, most "natural language like" languages do not use pronouns. These languages replace pronouns by nouns such as for example done in FSCL.

2.1.2.1.3 Specific NLL languages for study of behaviours

There are several NLL languages used for information description and retrieval in studying behaviours such as Observer' language, CABER's language and the Flexible Structure Coding Language.

Observer's language

The Observer Video-pro system, called Observer, has been developed for video analysis (Noldus Information Technology, 2003). This system defines a constructed language with a strict structure for generating descriptive sentences. The language structure is identified as "actor-behavior-modifier1-modifier2" or just "behaviour-modifier1-modifier2". The "actor" is the subject who displays the behaviours. The "behaviour" describes the basic behavioural elements to be observed. The "modifiers" allow a more detailed description of behaviours.

Each behavioural element can be supported by one or two "modifiers". Descriptive sentences are built based on those structures and are attached to video segments to describe behaviours of the actor in these segments.

CABER's language

The Caber system was developed to analyse behaviours through a computer system presented in Patrick (1985). The system was built with two main features: a build-in constructed language to provide sentence descriptions of possible behaviours and a capability for capturing behavioural events in real time. The built-in language is in the form of a LR(1)-type (see Patrick, 1985, for more details about this LR(1)-type). The coding descriptions, which are describing behaviours of video segments, are stored as code instances in a database. Using the system, users can create queries to search correct descriptions in this database. The query results can be used for statistical analysis of behaviours or for viewing the video segments of the retrieved descriptions.

Flexible Structure Coding Language (FSCL)

FSCL has been developed at Massey University (Heinrich, 1999, 2000). The goal of FSCL was to create a constructed language that is suitable for describing and retrieving information contained in multimedia recordings in the context of behaviours, multimedia education systems, online teaching or e-leaning. In FSCL, descriptive sentences are created by the combination of words which are defined into different categories. The syntax of FSCL is based on an LL grammar (see Louden, 1997 for more detail about LL grammars). The basic structure of FSCL can be understood as "subject-activity- object". With that specific structure, the language allows users to "retrieve correctly and completely the subject-verb-object relationships within the descriptive sentences" (Heinrich, 1999, p.iii). This constructed language was successfully integrated with a computer system called PAC for studying behaviour. Since the language is considered as the case study for this research, the next chapter will discuss this language in detail.

2.1.2.2 Constructed languages for presenting document structures

Different from the "natural language like" languages for studying behaviour, this type of languages, called *structured languages*, is used for describing documents that have structures. These documents can be seen in the form of product catalogues, digital libraries, and scientific data in repositories. Using these languages, users are able to retrieve precise information through specific query engines. A typical language in this category is the eXtensible Markup Language (XML). In this part, XML is described in detail.

XML

XML was originally designed to meet the challenges of large-scale electronic publishing to overcome the limitations of SGML (Standard Generalized Markup Language) and HTML (HyperText Markup Language) (W3C(1), 2004). The language can be used to store any kind of structured information, and to enclose or encapsulate information to pass it between different computing systems (Flynn, 2004). In general, a XML document can be seen as a document with descriptive tags before and after the text values. The following example of a catalogue for a music store gives a simple illustration of this language.

The catalogue stores information about a music CD. The information contains details about the performer, the composer, the title and tracks and the length of each track etc. The textual presentation of this catalogue is displayed as following:

Type of document: catalogue
Category: CD
Performer: Bob Marley
Composer: John Markey
Title: Is it love
Tracks:
  Track1: length: 2.34
  Track2: length 5.32
....

The catalogue data above can be presented by using XML:

```
<catalogue>
  <CD>
    <performer> Bob Marley</performer>
    <composer> John Markey</composer>
    <title> Is this love </title>
    <tracks>
       <track1 length="2:34">
      <track2 length="5.32"
        ........
    </tracks>
  </CD>
</catalogue>
```

The descriptive tags before and after the text values are written following Document Type Definitions (DTDs) or XML schemas. DTDs or XML schemas can be seen as blueprints for describing the structure of a XML document. In general, DTDs and XML schemas define:

- The sequence in which elements appear in a XML document;
- The interrelationships between different elements, including parent and child relationship and nesting levels;
- The types of data that are contained in elements or attributes.

DTD: DTD uses markup declaration to define rules that a XML document must follow. These rules can be elements, the sequence of elements, attributes, and entities that a XML document can contain. An element in XML commonly contains Character data. Characters that can be changed (or parsed) into other formats are presented as PCDATA. Characters that are not changed are presented as CDATA (Wagner and Mansfield, 2003). The DTD used for the XML document for the above CD catalogue can be displayed as:

```
<?xml version = 1.0"?>
<!DOCTYPE catalogue
<!ELEMENT CD(performer, composer, title, tracks)
```

```
<!ELEMENT performer(#PCDATA)
<!ELEMENT composer (#PCDATA)
<!ELEMENT title (#PCDATA)
<!ELEMENT tracks(track)
<!ELEMENT track(length)
<!ELEMENT length(#PCDATA )
```

XML schema: XML schema is a XML-based alternative to DTD for describing the structure of a XML document. The XML schema is also referred to as XML Schema Definition (XSD). Following W3C(1) (2004, p1), XML Schemas "provide a means for defining the structure, content and semantics of XML documents"

Associated with XML, there are important languages used for different purposes such as XSLT (Extensible Stylesheet Language Transformations) for transferring XML documents, XLink for linking XML documents, XPointer for locating information and XPath for retrieving information.

- *XSLT for transformation of XML documents.* Following W3C(2) (2004, p1), XSLT "is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element".
- *XLink for linking XML documents*: XLink is similar to the URL links in HTML but with much greater power. For example, in HTML, a link always has one source and one destination. In XML, a link may have multiple sources and destinations.
- *XPointer for locating the exact information:* XPointer provides access to the values of attributes or the content of elements anywhere in a XML document. Using XPointer, users can link to sections of text, select particular attributes or elements, navigate through elements, and so on.
- *XPath for searching for information:* XPath language is used to scout out XML data. This language has responsibility for looking into a given XML document and retrieving particular nodes that match a profile provided to it. In

- other words, Xpath helps users to find matching nodes between queries and XML documents.

2.2 Diagrams for presenting language structures

The section reviews the benefits of diagrams in enhancing human's cognition. It then studies how diagrams can be used for presenting language structures in different language theories.

2.2.1 Diagrams for enhancing learners' understanding

Long time ago, humans have recognised the benefits of using pictures compared to text to facilitate understanding. A number of research projects have been conducted to study the efficiency of diagrams for improving understanding in many areas such as applied psychology, linguistics, visual programming, data visualisation, graphic design, education, history and philosophy of science, cartography, and decision support sciences. Several benefits of diagrams in enhancing human cognition in general are presented in the following sections.

2.2.1.1 Efficiency of diagrams for enhancing human understanding

Human, from ancient time, has been acknowledged that pictures have many advantages compared to text explanation. For example, "A picture is worth 10,000 words" is a Chinese proverb that is widely known and believed (Larkin and Simon, 1987). Many people know that Chinese characters are written by means of hieroglyphics for the ease of remembering (Jordan, 2003). A proportion of such characters actually derive from fairly literal pictures of things. Below are some examples of Chinese characters with their English meaning:

伞	umbrella	木	tree	車	cart

One may recognise that the "伞" or "木" looks like the picture of an umbrella or a tree, respectively. The word "車" looks like a picture of a cart (when seeing the word as a picture from the left; it seems that the word has one wheel at the top and one at the bottom with an axle).

Larkin and Simon (1987) suggested that diagrams could group together all information that is used together, thus avoiding large amounts of search for the elements needed to make a problem-solving inference. Moreover, "diagrams typically use location to group information into a single element, avoiding the need to match symbolic labels" (Ullman et al. 1990, p1). In addition, Larkin and Simon (1987) also concluded that diagrams automatically support a large number of perceptual inferences, which are extremely easy for humans to understand. Addressing the same issue, Winn (1987) commented that the diagrams are useful because they easily attract learners. He also concluded that, getting learners to draw diagrams and maps of content is a useful way of developing good mental models. Cheng (1996) pointed out that expert knowledge in some domains commonly use diagrams encoded in perceptual chunks, or diagrammatic configuration schema. His research also emphasised that a diagram or graphic display have greater compatibility compared to explanation text and if they were absent, it is likely that the solutions to problems would be harder because there was less information available. Blackwell (1997) suggested that diagrams provide an expressive power and specificity that assists cognition as diagrams have fewer interpretations so they are more tractable than unconstrained textual notations.

Steinberg (1991) pointed out that visuals were remembered better than words because they are more likely to be encoded redundantly than words. Noldy, et al. (1990) also suggested that pictures are invariably remembered better than words on tasks of recall and recognition. This "picture superiority effect" is an "established memory phenomenon". He proved that "memory for pictorial stimuli is extremely accurate, durable and extensive compared to that for verbal stimuli" (Noldy, et al. 1990, p417). He also explained that pictures are memorable because the processing of pictures in the brain needs "additional allocation of attentional resources or effort" (Noldy, et. al. 1990, p418).

2.2.1.2 Efficiency of diagrams in computer science

Computer science has long history of creating and using diagrams. Flowcharts, functional decomposition diagrams, data structures and input/output schemas are examples of diagramming techniques (Koning et al, 2002). A well-known example of diagram usage in computer science is the area of visual programming languages. Chang (1989) suggested that visual programming languages have several advantages compared to traditional programming languages based on the following premises:

1. People, in general, prefer pictures over words.
2. As a means of communication, pictures are more powerful than words. They can convey more meaning in a more concise way of expression.
3. Pictures do not have language barriers that natural languages have. They are understood by people regardless of what language they speak.

Diagramming is also a valuable tool to help users to understand concepts and ideas in computer science. A good picture can be much better than thousand words because it is concise, precise, and clear. It does not allow the "sloppiness and woolly thinking" that are common in text specifications (Martin and McClure, 1985). It also assists teachers in explaining their ideas easily when writing does not make sense.

Diagrams also are useful tools for the development of computer applications. For example, the Unified Modelling Language (UML), a standard modelling tool for analysing and designing computer software, use diagrams extensively. Boggs, (1999, p.5) explained why visual diagrams are widely utilised in this language "We seem to be able to understand complexity better when it is displayed to us visually as opposed to written textually. By producing visual models of a system, we can show how the system works on several levels. We can model the interactions between users and a system". Poling (2000, p1) further underlined that "A UML diagram is worth a thousand lines of code".

2.2.2 Diagrams in language structure presentation

As discussed in the previous chapter, grammar includes two inter-related elements: *morphology* which explains how words are formed out of smaller units or word structures and *syntax* that determines how words can be combined to form phrases and sentences. Syntax of a language can be presented by two forms: textual explanations or structure diagrams. This section first reviews the textual approach then looks closely at the more common approach that uses structure diagrams.

2.2.2.1 Non-diagram presentation

A very common method is the bracketing presentation. This method is used to display Phrase Structure Grammar (PSG) referring to constituency-based syntactic analyses developed by Leonard Bloomfield as reported for examples by Hilferty (2002). In this technique, higher-level constituents are shown by having lower level constituents included within "wider" brackets. The various brackets can be labelled to indicate the kind of constituent in each case. The example below illustrates this type of presentation (in similar form by Crowley et. al., 1995).

"This stupid politician will kill the country with economic rationalism"

${}_{\text{S}}[{}_{\text{NP}}[{}_{\text{DET}}$ This ${}_{\text{ADJ}}$ stupid ${}_{\text{N}}$ politician]] ${}_{\text{AUX}}$ [will] ${}_{\text{VP}}$ [${}_{\text{V}}$ kill ${}_{\text{NP}}$ [${}_{\text{DET}}$ the ${}_{\text{N}}$ country] ${}_{\text{PP}}$ [${}_{\text{P}}$ with ${}_{\text{NP}}$[${}_{\text{ADJ}}$ economic ${}_{\text{N}}$ rationalism]]]

This sentence is created by a noun phase (NP) constituent "This stupid politician", an auxiliary (AUX) "will", and a verb phase (VP) constituent "kill the country with economic rationalism". These constituents of the sentence can be divided into their own immediate constituents. The NP consists of a determiner (DET), "This", and adjective (ADJ) "stupid", and the noun (N) "politician". The V consists of a verb (V) "kill", and another NP "the country" and a prepositional phrase (PP) "with economic rationalism". The PP consists of a preposition (P) "with" and another NP "economic rationalism".

One can recognise that it is not easy to see the overall relationships among the constituents following this non-diagram presentation. With more complicated sentences, more nested bracketing is required. Thus these kinds of sentences will be difficult to interpret especially when there are a lot of small functional labels included (Crowley et. al., 1995). The only reason this bracketing method is still applied is that when a word processor is used, it is less of an effort to add brackets than to draw a tree representation (Borjars and Burridge, 2001)

The above bracketing method can be presented in a different way called Phrase Structure Grammar rules (PSG rules). This technique is presented by the example below:

S→ NP(AUX)VP
NP→(DET) (ADJ)N
VP→ V(NP)(PP)
PP→P(NP)

Then, with the simple sentence "John gives the flower to Lisa", the structure of the sentence is presented as following:

$[_{S} [_{NP}[\text{John}]] [_{VP} [_{V} \text{gives}] [_{NP} \text{the flower}] [_{PP} \text{to Lisa}]]]$

...

$[_{NP} [_{Det} \text{the}] [_{N} \text{flower}]]$
$[_{PP} [_{P} \text{to}] [_{NP} \text{Lisa}]]$

PSG rules can be written for other languages with a structure that is very different from English. For example, the following PSG rules are for Fijian language presented in Crowley et. al. (1995)

S→VP(NP)

VP→SUBJ(TENSE)V(NP)

NP→DET N (ADJ)

Note: SUBJ is the subject-indexing pronoun and TENSE is the tense marker.

2.2.2.2 Diagrams in presenting language structure

To present language structures, a more common technique is the use of diagrams. This subsection presents some typical diagrams used to display structures in different natural language theories and constructed languages.

2.2.2.2.1 Structure diagrams of different natural language theories

The benefits of diagrams in studying language have been emphasised by numerous linguists. Many grammarians and English instructors suggested that analysing a sentence and presenting its structure with a visual scheme is very helpful—both for language beginners and for linguistics. By placing the various parts of a sentence in relation to the basic subject-verb relationship, the learner can see how these parts fit together and "how the meaning of a sentence branches out, just as the branches of a plant ramify from the stem in place and time" (Crowley et. al., 1995, p123). Bernstein (1992, p1) pointed out that, "When you are at school and learn grammar, grammar is very exciting. I really do not know that anything has ever been more exciting than diagramming sentences. I like the feeling the everlasting feeling of sentences as they diagram themselves". Capital Community College Foundation (2003, p1) gave another plausible quote: "Sentence diagramming is one of the best analytical techniques I ever learned".

As outlined in the previous discussion (Section 2.1.1), there are many grammar theories. Unlike the non-diagram presentation, which may be only used for Phrase Structure Grammar (PSG), a grammar theory is commonly accompanied by its own structure diagram to present its specific grammar structures. The next section will review some typical grammar theories and their associated syntax diagrams.

Diagrams in Phrase Structure Grammar

Following Phrase Structure Grammar theory, sentences are broken into smaller constituent parts step by step following the steps below:

- Take a whole and subdivide it into individual parts.
- Once this is done, the parts can in turn, be treated as wholes to be subdivided into further parts.
- The process can be continued until the parts become words.

Following these steps, a sentence as a whole is broken down into phrases or parts at the highest level of analysis. At next level of analysis, the phrases, which are wholes, can be broken down into smaller phrases or individual words. There are several versions of diagrams that show this type of structures including Phrase Structure Tree, Diagramming and Chinese box.

Phrase Structure tree

This is one of the most popular ways to present sentence structures. The method is considered as equivalent to the bracketing method discussed previously (Section 2.2.2.1). In the Phrase Structure trees, the immediate constituents of the sentence are like limbs of a tree while their constituents are like branches. The diagram is clearer when the nodes are labelled with the types of the constituents. There are two types of tree diagrams: bottom up and top down. Figure 2.1 shows a syntax tree for the sentence “The bear runs over the mountain” that is displayed as a bottom-up diagram.

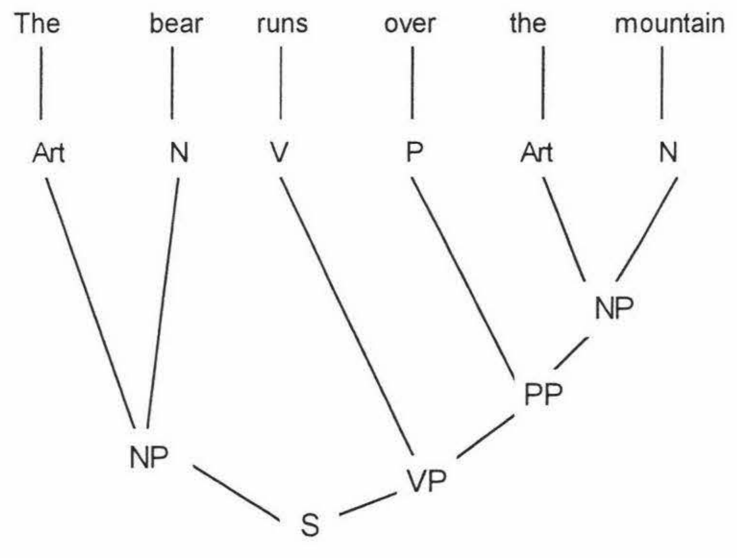


Figure 2.1 Bottom-up syntax tree of a simple sentence “The bear runs over the mountain”

In this tree diagram, the bottom “S”, which dominates the whole string of words, is claimed for the sentence. The NP comprises the noun phrase “The bear”. VP presents for the verb phase, “runs over the mountain”. The initial NP itself comprises an article (Art) “The” and a noun (N) “bear”; the VP comprises a verb (V) “runs” and a Prepositional Phrase (PP) “over the mountain”. This PP itself is made up of a preposition “over” and an NP “the mountain”, which is made up of an article “the” and a noun “mountain”.

However, Crowley et al. (1995) said that it is more conventional in linguistics to show the diagram as an upside down tree. The reason for this is that these diagrams look like a family’s hierarchy, organization, or files in a folder that are very familiar in common life. Figure 2.2 shows this type of trees:

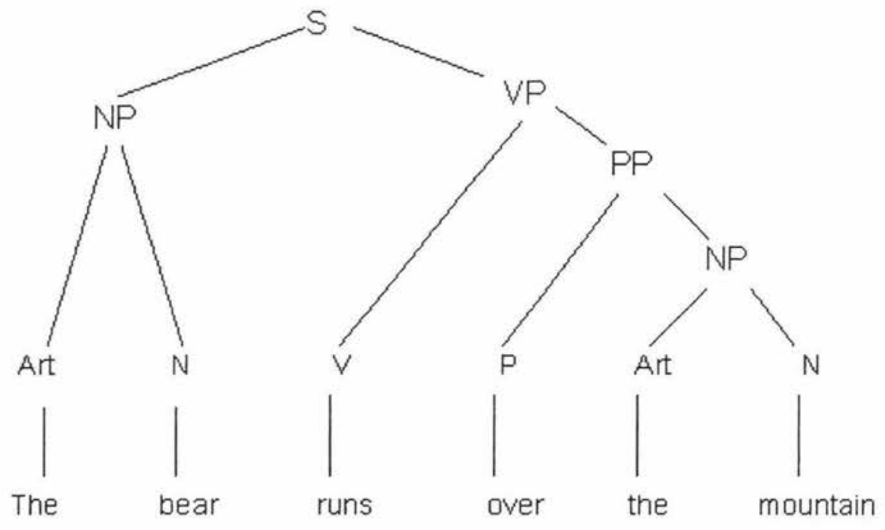


Figure 2.2 Top-down syntax tree of the sentence "The bear runs over the mountain"

It can be seen that the tree provides very important information: the word class or category of each word, the phrase structure of the whole sentence (including what the word groupings are and their hierarchical structure - how they are nested or not nested inside each other) and the phrasal category of each phrase. Every branch in a tree ultimately ends in a word and every word is at the bottom of just one path of branches starting from the "S" at the top ("root") of the sentence. When tree diagrams are initially seen, they may look unfamiliar and difficult to understand. However, when users get used to them, trees give a good overview of the structure of a sentence (Borjars and Burridge, 2001).

To highlight phrases in a sentence, the Phrase Structure tree of a sentence can be displayed with triangles presenting phrases or constituents. This kind of diagram is called the abstract syntax diagram. Figure 2.3 displays this kind of diagram for the sentence "The bear runs over the mountain". By using this way, learners can easily recognise that the sentence includes two main phrases: noun phrase and verb phrase. The noun phrase includes two words "the" and "bear". The verb phrase includes two sub-verb phrases: verb phrase and preposition phrase. The verb phrase contains only one word "runs". The preposition phrase includes three words "over", "the" and "mountain".

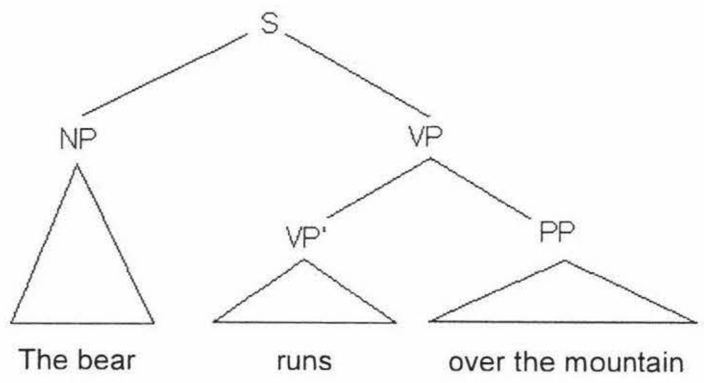


Figure 2.3 An abstract syntax tree for the sentence "The bear runs over the mountain"

Diagramming

Reed-Kellogg diagramming techniques are visual maps used in the schoolroom to display the structure of sentences. This kind of diagrams, along with some other notations, was developed during the nineteenth century. However, only Reed-Kellogg diagrams are still used in many schools today (Thomas and Muriel, 1996). Figure 2.4 presents a Reed-Kellogg diagram of the sentence "This new Panda makes new friends easily". In this diagram, the main horizontal line represents the core of the sentence. A vertical line intersects the horizontal between the main word "Panda" of the subject "new Panda" and the main words of the predicate "makes new friends easily". A vertical line following the verb, "makes", separates it on the horizontal line from the main word of its direct object, "friends". Diagonal lines point towards words they modify. The noun modifiers "This" and "new" are on diagonals below "Panda" and the adverb "easily" is on the diagonal below the verb "makes".

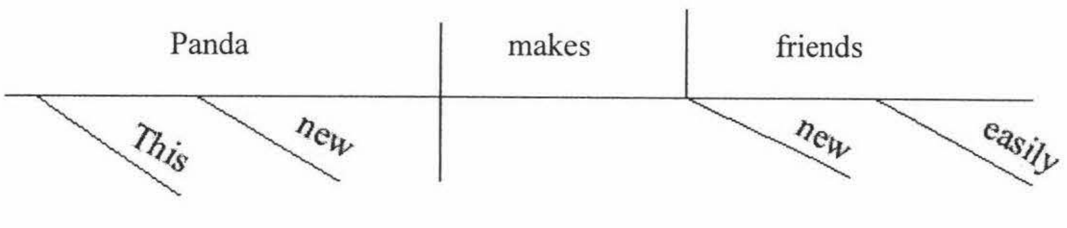


Figure 2.4 Diagramming of the sentence "This new Panda makes new friends easily".

Figure 2.5 shows the structure of a more complicated sentence – a compound sentence "Forecasting technologies are more sophisticated and today's forecasters are better trained, but weather predictions are still not very reliable" (in similar form from Capital Community College Foundation, 2003, p1):

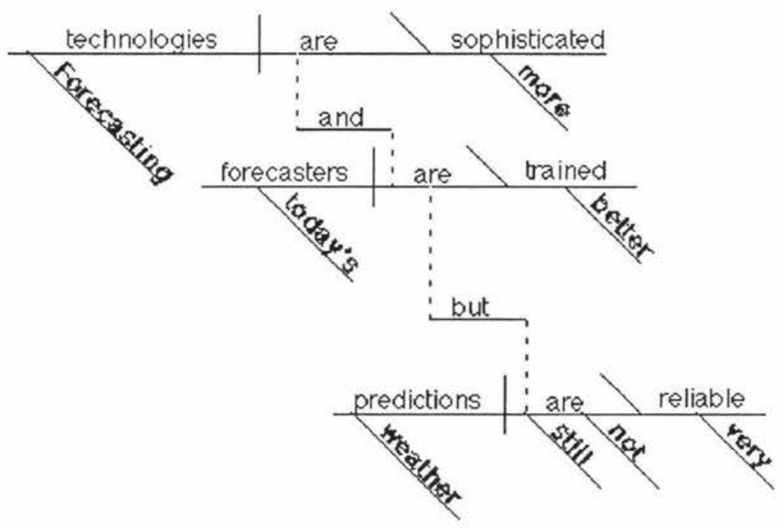


Figure 2.5 Diagram for a complex sentence

IC Analysis Diagram

Immediate Constituent (IC) is a system mainly used by American structural linguists for analysing the hierarchical structure of sentence and their parts (Thomas and Muriel, 1996). The system uses the Analysis Diagram to present results of IC analysis. In this type of diagrams, arrows are used to show sentence elements including words and phrases that modify these words. Figure 2.6 presents an IC Analysis Diagram for an example sentence "These new students learn the grammar lessons easily".

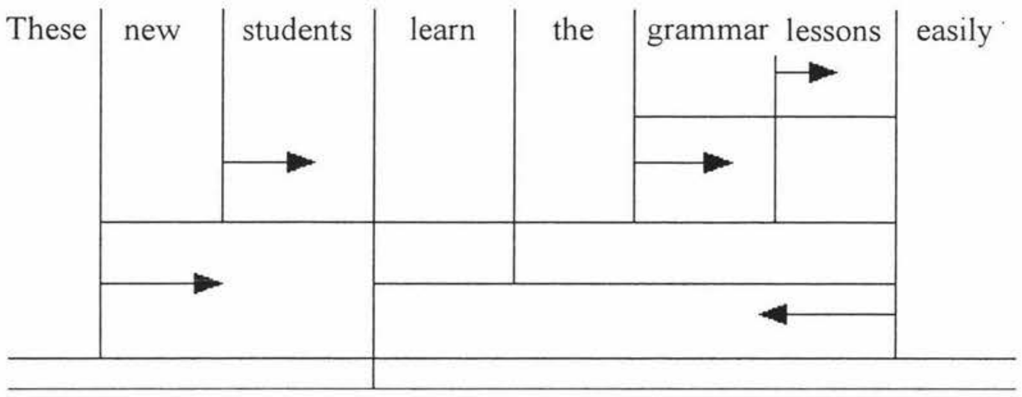


Figure 2.6 IC diagram of the sentence "These new students learn the grammar lessons easily"

Studying the diagram from bottom-up, one can see how phrases contained in the sentence form its parts. The third vertical line divides the entire subject, "These new students", from the entire predicate, "learn the grammar lessons easily". Each of these is divided into the parts that constitute it and so on. For example, the subject has two major constituents, "These" and "new students". The advantage of this kind of diagram is that it can show how there is a hierarchy in modifiers with one modifying another that modifies yet another (Thomas and Muriel, 1996). One can see that, in this diagram, lexical categories of words in the sentence are not presented.

Chinese boxes

Another way to show the grammar structure of a sentence is the use of Chinese boxes (also known as nested boxes). It looks like "hats and shoes" (Hilferty, 2002). The Chinese box presenting the grammar structure of the sentence "John gives the flower to Lisa" is displayed in Figure 2.7.

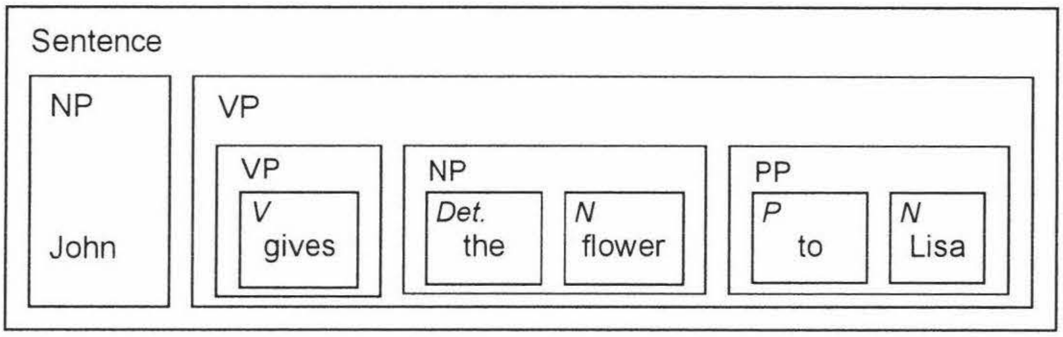


Figure 2.7 Chinese box for the sentence "John gives the flower to Lisa"

In this diagram, the sentence includes two main phrases: noun phrase (NP) and verb phrase (VP). The noun phrase has only one word "John". The verb phrase includes three smaller boxes of its constituents: "verb", "noun" and "prepositional" phrases. Each of these smaller boxes includes constituent boxes and so on. The diagrams are easy to understand. However, if a sentence is long, its Chinese box diagram will be very complicated as it will have many nested boxes and require lots of space.

Diagrams in Dependency Grammar

Dependency Grammar (DG) is a very popular grammatical theory in linguistics. One of the best-known approaches developed based on Dependency Grammar has been constructed by Richard Hudson (Hilferty, 2002). The grammar does not rely on constituency but on relationships between individual words and in some cases, the linear ordering of words. There are two main elements of a dependency: the head and its dependents. The meaning of the head is usually completed by a complement or modified by an adjunct. In a DG diagram, the verb is considered as root. Figure 2.8 shows a DG diagram representing the simple sentence "Lisa reads a book".

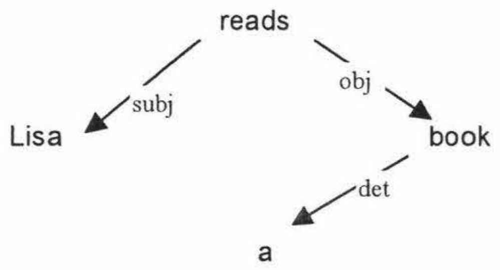


Figure 2.8 DG diagram for a simple sentence

In this diagram, the links with labels refer to the syntactical function of the dependent element. Following DG structure, the sentence above can be interpreted as following:

"Lisa" depends on "reads";

"a book" also depends on "reads";

"a" depends on "book";

"reads" depends on nothing.

Therefore, "reads" is considered as the "root" of the sentence in Dependency Grammar and is placed at the top of the diagram.

Figure 2.9 shows a diagram of a more complicated sentence analysed by this type of grammar (adapted from Tapanainen, 1997).

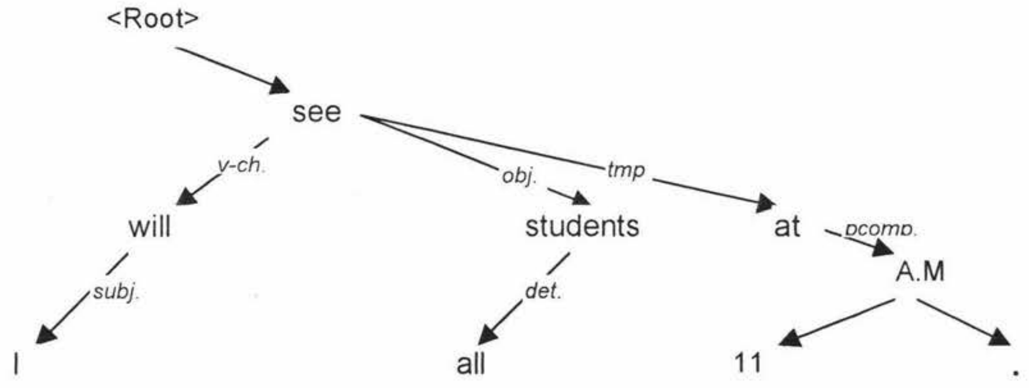


Figure 2.9 Dependency Grammar tree for the sentence "I will see all students at 11 a.m."

Diagrams in Word Grammar

Word Grammar theory (WG) was developed by Richard Hudson from the early 1980's (Hudson, 2002). This grammar uses word-to-word dependencies and a noun is the subject of a verb. The main difference between the Word Grammar and the popular language theory, Phrase Structure Grammar (PSG), is that the Word Grammar does not recognise a noun phrase as the subject of a clause as the PSG does. In addition, in Word Grammar, grammatical relations/functions between words are presented by using labels. Figure 2.10 presents a Word Grammar structure for the sentence "Syntactic dependencies make phrase structure redundant". Each word in the sentence is in the centre of a small network of links to other words; and these networks combine into a bigger network for the whole sentence. This network is not equivalent to a phrase structure tree because there are double dependency links for the word "structure" from the words "make" and "redundant". Figure 2.10 also displays the relations/functions among words such as "subject", "object" or "adjunction".

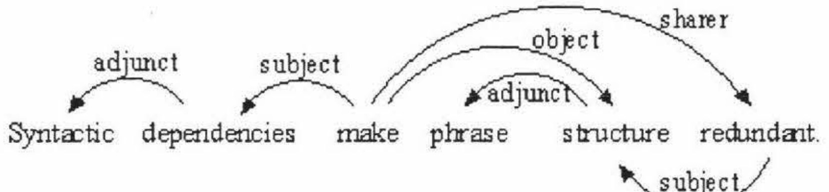


Figure 2.10 Sentence structure diagram for a sentence using the WG theory

Diagrams in Lexical Function Grammar

Lexical-Functional Grammar (LFG) is a unification-based linguistic formalism which is suitable for computation (Wong, 2001). Comparing with PSG, LFG uses different structures to represent different levels of linguistic information of a sentence. While PSG captures the external structure of a sentence, the Lexical Functional Grammar represents the internal structure of a sentence. This includes the representation of the higher *syntactic* and *functional information* of a sentence. The higher *syntactic* information of a sentence shows the grammatical information of a lexical item. For example the word "books" is in plural form of the word "book", and the word "played" is expressed in past tense of the word "play". The *functional information* of a sentence stores the information about functional relations between elements of sentences and how elements of the sentence affect each other. The relationship is presented by links drawn between them. The *syntactic* and *functional information* of a sentence is shown in a functional structure diagram as a set of "attribute-value pairs". In an "attribute-value pair" of a functional structure diagram, the attribute corresponding to the name of a grammatical symbol (e.g. Numb, Tense) or a syntactical function (e.g. Subj, Obj) and the value are the corresponding feature possessed by the concerning constituent. For example, the functional structure diagram representation of the NP (Noun Phrase) "John" which functions as the subject in a sentence is presented in Figure 2.11 (in this diagram, Pred. and SG are the abbreviations of the word "Predicator" and "Single" respectively).

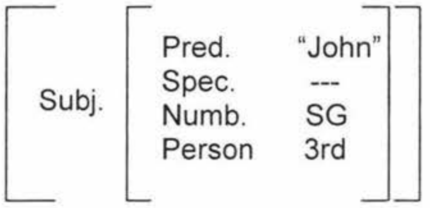


Figure 2.11 Functional structure representation for a noun as a subject

The functional structure diagram can be a multi-levelled tree-like structure as it may contain subsidiary functional structures. In the functional structure, some of the attributes appear in different places and sometimes they are linked with each other. For example, in the sentence "John tried to play the guitar", the subject of the sentence "John" is also the subject of the complement clause "to play the guitar" (Figure 2.12). The value of the attribute "Subj." ("John") is linked to the value of the same attribute in the functional structure of the complement (see Wong, 2001 for more details).

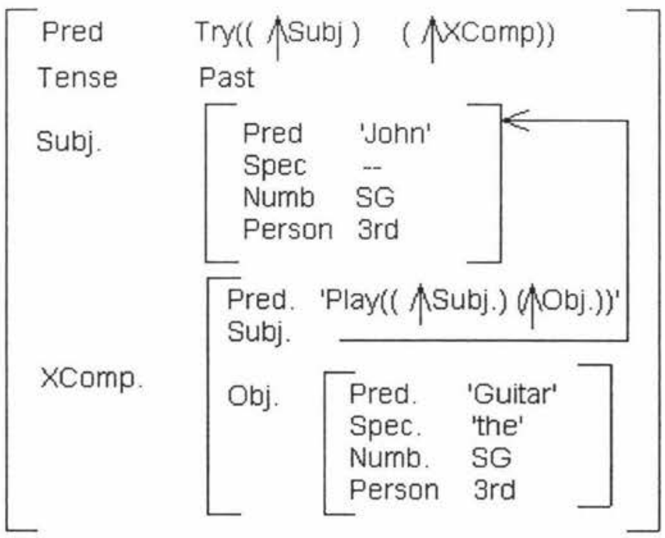


Figure 2.12 Functional structure for the sentence: "John tried to play the guitar"

2.2.2.2.2 Structure diagrams of constructed languages

Diagrams are not only commonly used in natural language to present natural language structures. They are also utilised to display constructed language structures extensively. This section presents diagrams used to display language structures for FSCL and XML.

Diagrams in presenting FSCL structure

FSCL uses a structure tree called semantic tree to present the structure of FSCL sentences (Heinrich, 1999). This kind of tree can be seen as a general tree that has an unlimited number of nodes and lines. A tree may have a minimum level of two presenting "subject-verb" structure or a maximum level of six presenting "subject-verb-direct object-verb-indirect object" structure. Figure 2.13 presents a semantic tree diagram for the sentence "John gives a book to Mike".

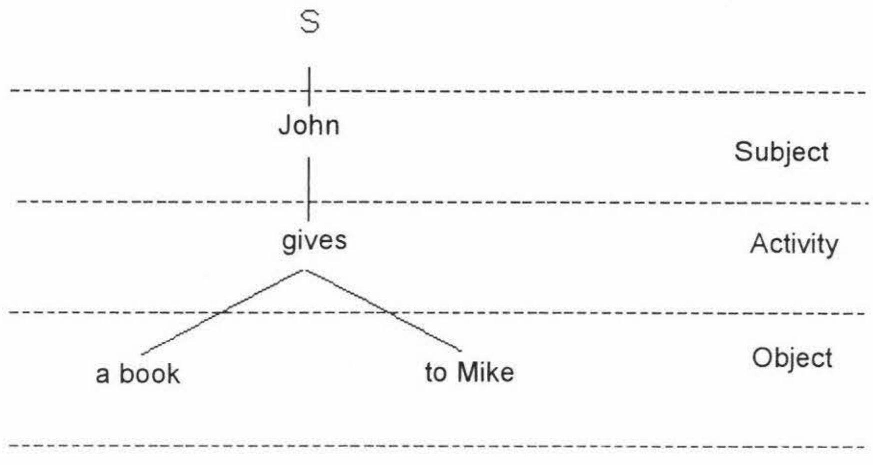


Figure 2.13 The FSCL semantic tree of a FSCL sentence

The semantic tree in this diagram is divided into four levels. The highest level is the root "S" presenting for Sentence. The subject "John" and the verb "gives" are in the second and third level respectively. The two objects "a book" and "to Mike" are displayed in the fourth level. A FSCL sentence is read by going through the semantic tree from top to bottom, left to right direction. As this language is used for the research's case study, the language is discussed in more details in the next chapter.

Diagrams presenting XML structure

XML documents can be displayed through structure trees easily. For example, the structure diagram of a catalogue document (first mentioned in Section 2.1.2.2) using XML can be presented in Figure 2.14. In this figure, the textual format of the document is located in upper left hand side and its mapping diagram is displayed in the lower right hand side. The node in the top of the diagram presents the whole structure of the document and nodes below it present sub-divisions of the documents. These nodes may connect to other nodes or be leaves.

```
<Catalogue>
   <CD>
      <performer> Bob Marley</performer>
      <composer> John Markey</composer>
      <title> Is this love </title>
      <tracks>
          <track length="2:34">
            ........
      </tracks>
         ...
   </CD>
</Catalogue>
```

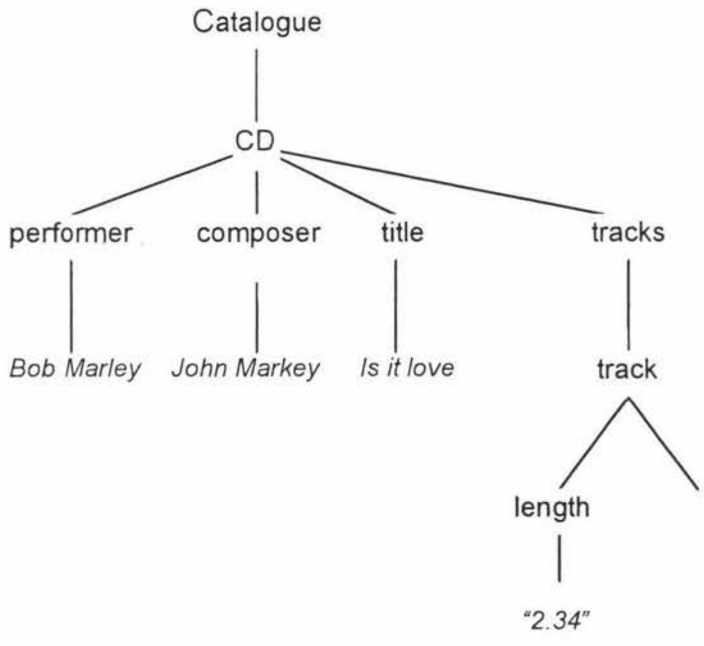


Figure 2.14 Mapping tree of a XML document

2.3 Computer Aided Language Learning

Users can learn language in different ways, such as by attending classes or by reading books. Nowadays, with the invention of computers, users have one more choice:

using computer applications to learn language. These applications are called Computer Aided Language Learning (CALL) applications. Compared with other methods of learning, computer applications have many advantages. This section first reviews advantages of using computers in language learning. It then examines elements that affect the development of computer applications for language learning. Towards the end, the section reviews computer applications for learning natural language with and without using graphical diagrams.

2.3.1 Benefits of CALL in language learning

Traditional language learning takes place in classrooms where one teacher addresses a larger number of students. This brings some difficulties for students. Because there are many students in the class, each student can get only a limited amount of attention. The speed of the course or its difficulty is independent of the individual student's ability. Now, with the invention of computers, those limitations of teaching students through classrooms are not longer an issue.

Many computer applications have been developed to teach learners various aspects of natural languages. These applications (Computer Aided Language Learning applications - CALL) offer powerful self-assessment facilities. They generate learner-centred and self-paced activities. The CALL applications can be sensitive to the level of proficiency of the leaner and also can self-adjust in real-time in response to the learner input. Moreover, learning lessons can be called up by students at will. In addition, computer applications can do everything very quickly. They never tire and nor does their attention falter. Moy and Eliëns (1994) pointed out some advantages of computer applications in learning language as following:

- The student is able to get attention from a computer as a tutor.
- The course material and level of the lesson can be adjusted to the needs of each student.
- The course can be held at a time and place as required by the student.
- The student can get direct feedback from exercises.
- The student can repeat anything without much effort. For example, if a student does not understand a lesson, he can replay that lesson until he completely understands it.

2.3.2 Areas of CALL in Language Learning

Computers can assist in teaching students in many fields: Grammar, Writing, Vocabulary and Reading. The following section presents some benefits of CALL in language learning.

- *Grammar:* Many researchers state that the first computer applications for teaching languages focused on teaching grammar (Cameron, 1999). To teach grammar, computer applications present grammar rules, display structure of sentences or provide exercises that help students to learn language syntax. The following points indicate that why a computer application can be a good tool to support teaching grammar (Cameron, 1999).
    - Grammar rules can be presented in a manner, which takes account of individual preferences.
    - The computer provides plenty of practice, combined with instant feedback that prompts learners to reconsider their understanding of the rules as soon as they make a mistake, instead of leaving them with an incorrect rule for some time as so often happening with ordinary homework.
- *Writing:* Dowling (1999) pointed out that CALL could help learners of all ages for writing. She said that computers have made possibly a relationship between the act of writing and the process of thinking and learning. Writing using a computer, a learner can be free to explore and develop ideas, concepts and relationships through the language. Therefore, the learner's thoughts can be tentatively put into words, modified, rearranged and discarded if necessary. This helps learners' understanding to grow. Moreover, writing using computers can be particularly helpful to those students identified as having learning difficulties or for whom the act of handwriting physically is difficult. These learners gain benefits from the new freedom and flexibility offered by word processing. Spelling is no longer an issue because the computers provide convenient facilities such as spelling checkers.
- *Vocabulary:* Studying vocabulary is one important part of learning language. It has been common to give learners lists of words to memorise together with periodic tests. This method was successful to a certain degree, but computers have the potential for more effective techniques. Computers now can teach

learners to learn vocabulary in exciting ways. A computerised word game is an example. In this kind of game, a computer has many roles. For example, it serves as a delivery medium, displays the puzzle to be solved and provides clues. In some cases, computers act as an opponent for the leaner or even facilitate for learners to compete against each other. The computer also records the learners' guesses and can check their spelling.

- *Reading skills:* Computers can help learners to improve reading skill by using many techniques. For example, FReader application (ReadingSoft, 2004) provides various reading modes that offer an impressive acceleration of reading pace within each training session. In another example, the ACEREADER application improves learners' reading skill (both speed and comprehension) through "the use of Drills and Games that are based on fundamental and sound pacing principles" (Learning Center, 2004, p1).

2.3.3 Elements which influence CALL development

CALL is inherently multidisciplinary. A joint policy statement of the international research organisations CALICO, EUROCALL and IALLT (CALICO, 1999, p1), stated that "CALL applies research from the fields of second language acquisition, sociology, linguistics, psychology, cognitive science, cultural studies, and natural language processing to second language pedagogy, and it melds these disciplines with technology-related fields such as computer science, artificial intelligence and media communication studies. In integrating these disciplines, CALL's work requires a huge range of complex activities and initiatives in development, pedagogical innovations and research". Addressing the same issue, Levy (1997) described the relationship of CALL with other major fields by using a diagram which is presented in Figure 2.16.

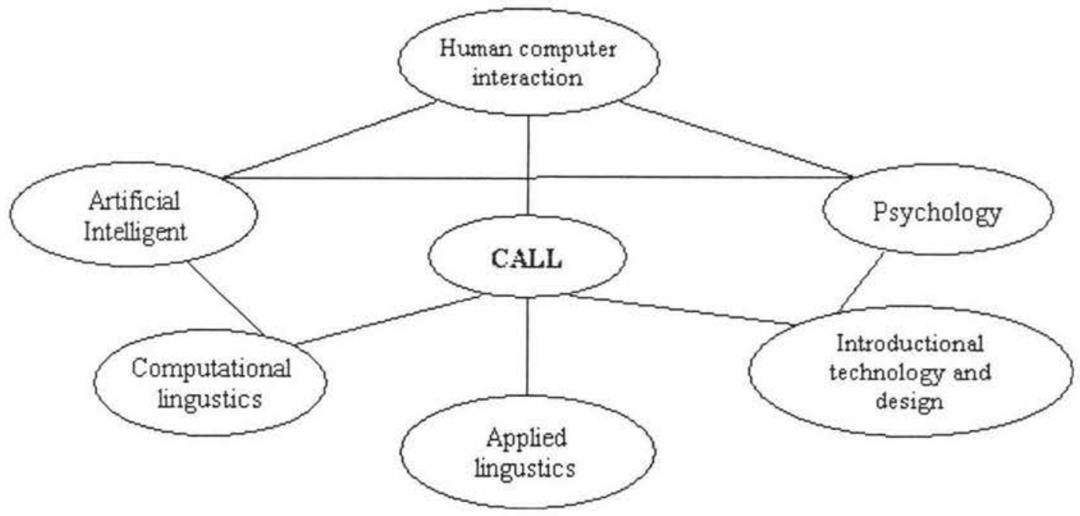


Figure 2.16 CALL and related disciplines (adapted from Levy, 1997)

Among those major fields, two fields, Human Computer Interaction and Computational Linguistics, are closely related with the development of CALL applications. Details of these two fields are discussed in the following sections.

2.3.3.1 Human Computer Interaction

Human Computer Interaction (HCI) focuses on all aspects that relate to the interaction between users and computers so that users can "use friendly" or "easily use" computers. Preece et al. (1994, p1) defined that "Human Computer Interaction (HCI) is about designing computer systems that support people in such a way that they can carry out their activities productively and safely".

HCI focuses on the design of interactive systems considering user requirements and the natural elements of the tasks that users have to complete. HCI includes discussions of the design aspects of menu selection systems, command languages and interaction styles and so on. Thus, knowledge of HCI design can provide CALL developers with a sense of the range of design options and the implications that are inherent in each of them. To develop a CALL user interface, Chapelle (1991) recommended that designers must consider three aspects of interactions: how the learners input data; how the learners interact with the input and how the system produces outputs. By analysing these interactions, designers can identify the important elements for the development of computer applications that help learners to improve their abilities of languages.

2.3.3.2 Computational Linguistics

Computational linguistics is the study of computer systems for understanding and generating natural languages. Computational linguistics needs an understanding of the entire process of natural language comprehension and generation (Grishman, 1986). Achieving high quality Natural Language Processing (NLP) has been one of the central goals of computational linguists (Levy, 1997). NLP is used for analysing natural languages automatically with the help of software programs called parsers. Most CALL applications include a NLP part which analyses natural language input by using a parser that judges the input on the basis of linguistic or grammar rules. The main functions of a parser are (Hausser, 1999, p18):

- Decomposition of a complex sign into its elementary components;
- Classification of the components via lexical lookup, and
- Composition of the classified components via syntactic rules in order to achieve an overall grammatical analysis of the complex sign.

2.3.4 CALL applications for natural grammar learning

CALL applications can be used to explore and solve issues experienced in relation to grammar learning. Key purposes of CALL applications for learning grammar are (Levy, 1997):

- Building parsers that take a text and try to analyse it according to grammar rules.
- Exploring different kinds of grammar and their suitably for presenting the variations of natural language.
- Exploring ways of representing the meaning of sentences so that an appropriate response may be made.
- Integrating the syntactic and semantic components in terms of an effective control structure.

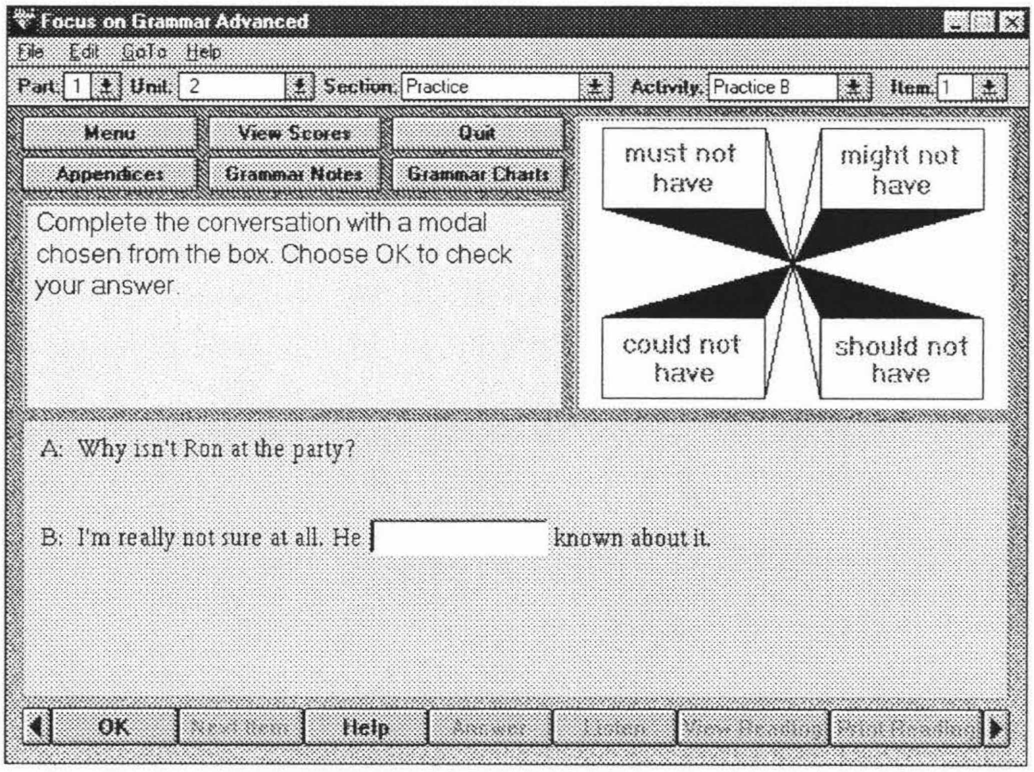
In this section, a range of CALL applications to teach natural language grammar is presented. The research separates CALL applications into non-diagram and diagram applications.

2.3.4.1 Non-diagram applications

There are applications that do not use diagrams to present language structures. Instead, these applications use textual explanations to teach language grammar. The applications reviewed belonging to this category are: Focus On Grammar, English Sentence Analysis and Constraint Grammar Parser of English.

2.3.4.1.1 Focus On Grammar

Focus On Grammar (FOG) software was developed by Exceller Software Corp (Exceller Comp., 2003). FOG offers extensive grammar practices in the form of grammar exercises. It can be used in or outside classrooms for learners to practice areas where they have problems, or to improve their grammar skills. Learners first learn language structures by viewing Grammar Notes. They then do exercises provided by the application. FOG provides a number of exercises leading gradually from isolated, simple tasks to more complex ones. The application can assess knowledge of learners through a scoring system. Figure 2.17 presents a screenshot of this application showing a practical grammar exercise. In this exercise, a learner has to select a word group to fill a blank space in a sentence.



Part: 1 Unit: 2 Section: Practice Activity: Practice B Item: 1

Complete the conversation with a modal chosen from the box. Choose OK to check your answer.

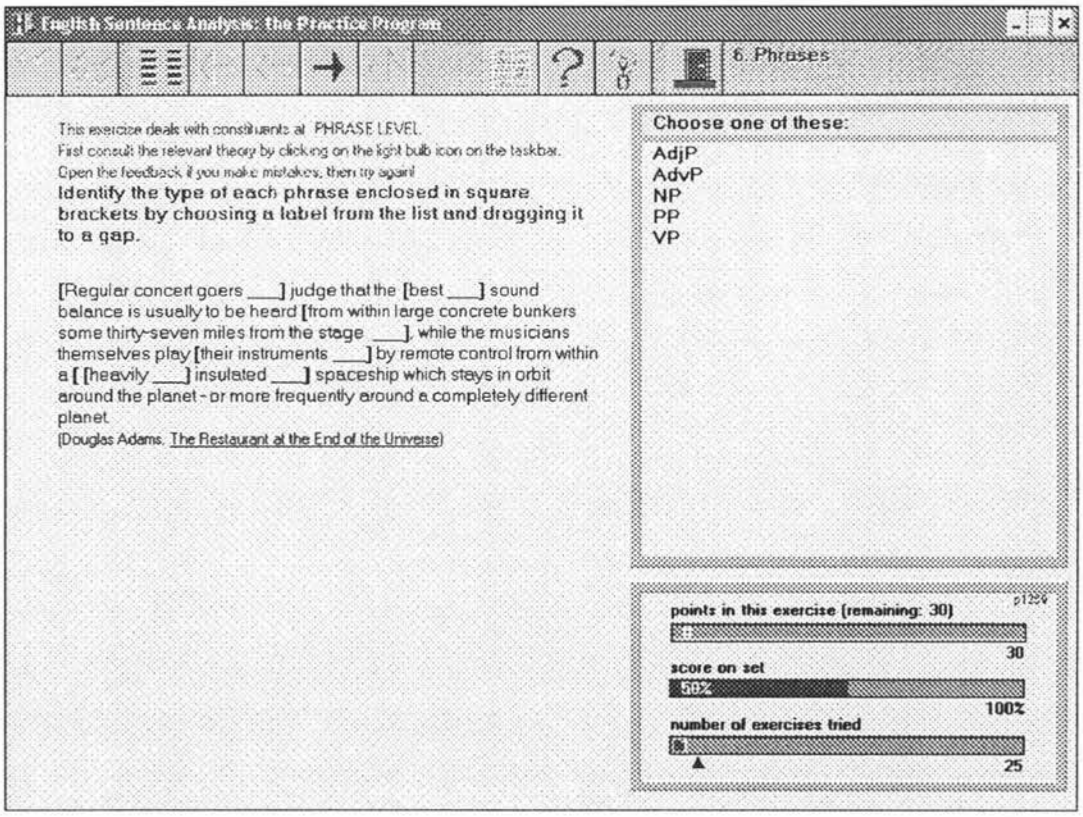
A: Why isn't Ron at the party?

B: I'm really not sure at all. He ______ known about it.

Figure 2.17 A grammar exercise of "Focus on Grammar" application

2.3.4.1.2 English Sentence Analysis

The English Sentence Analysis (ESA) application has been developed at the University of Groningen in Netherlands (Verspoor and Sauter, 2000). The main purpose of this application is to make students aware of different levels of analysis in sentence, clause and phrase level. It also helps students to familiarise themselves with the traditional terminology of *sentence constituents* such as subject, predicate and direct object; *word classes* such as noun, verb and object; and *phrase constituents* such as head, pre-modifier and post-modifiers (Verspoor and Sauter, 2000). The application contains many exercises including multiple-choice, drag & drop and fill-in-the-blank questions. After each lesson, the application will assess the level of the learner's grammar understanding. To get a "pass", a learner must score at least a certain mark on each sub-topic in a set. An interesting feature of this application is that if a user does not reach the pass mark for a certain period of time, a new set of questions will be created focusing on the topics the user did not master. Figure 2.18 shows the application's user interface during an exercise.



This exercise deals with constituents at PHRASE LEVEL.
First consult the relevant theory by clicking on the light bulb icon on the taskbar.
Open the feedback if you make mistakes, then try again!
Identify the type of each phrase enclosed in square brackets by choosing a label from the list and dragging it to a gap.

[Regular concert goers ___] judge that the [best ___] sound balance is usually to be heard [from within large concrete bunkers some thirty-seven miles from the stage ___], while the musicians themselves play [their instruments ___] by remote control from within a [[heavily ___] insulated ___] spaceship which stays in orbit around the planet - or more frequently around a completely different planet.
(Douglas Adams. The Restaurant at the End of the Universe)

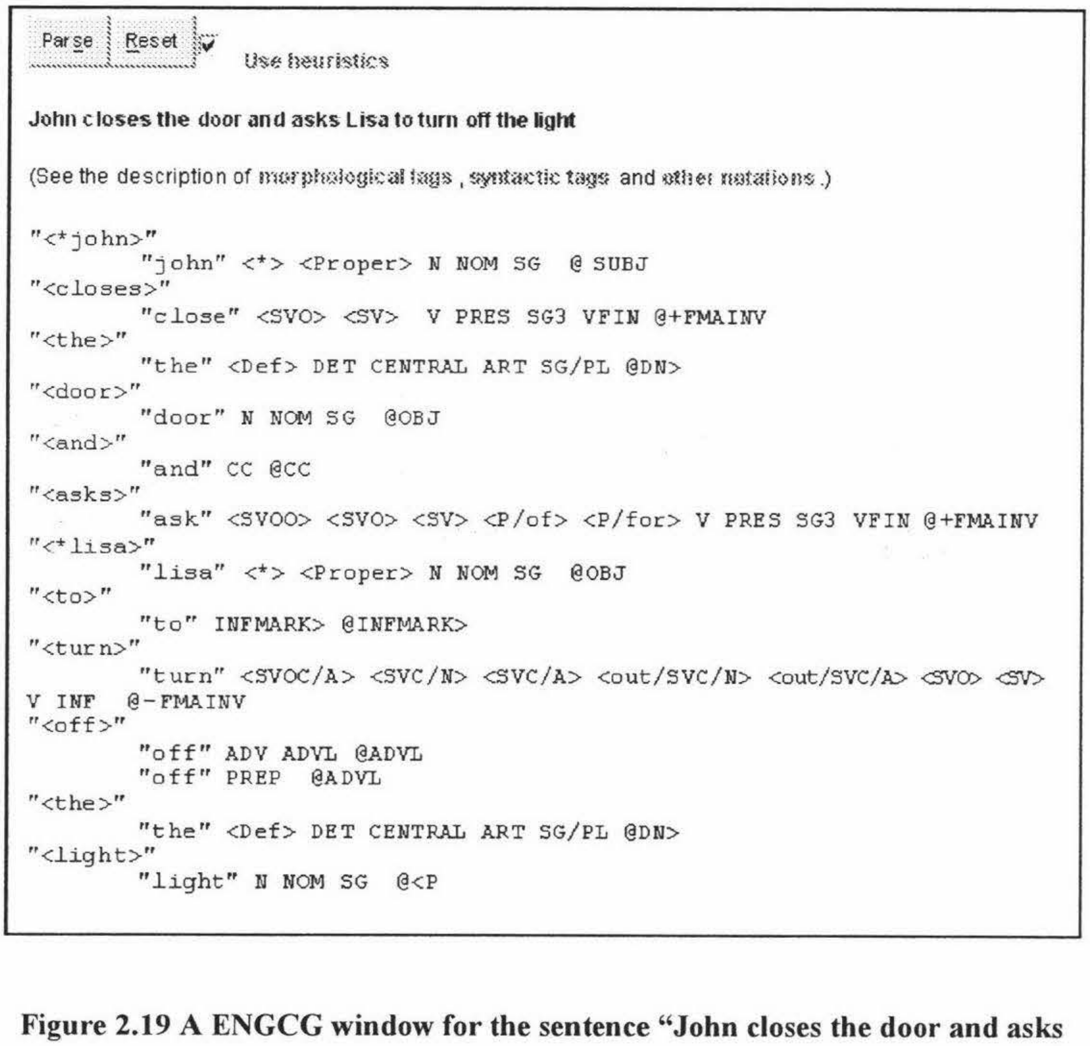
Choose one of these:
AdjP
AdvP
NP
PP
VP

p1254
points in this exercise (remaining: 30)
30
score on set
50%
100%
number of exercises tried
8
25

Figure 2.18 A phrase exercise example of "English Sentence Analysis"

2.3.4.1.3 Constraint Grammar Parser of English

Constraint Grammar Parser of English (ENGCG) was developed by the Lingsoft team in Finland (Lingsoft Comp., 2003). ENGCG performs morph syntactic analysis (tagging) of English text. Learners can enter a sentence and the application will return a mark-up for the grammatical function of the words in that sentence. These words are not only assigned multiple tags but also multiple syntactical functions. Figure 2.19 shows the output of the application when a user enters the sentence "John closes the doors and asks Lisa to turn off the light".



John closes the door and asks Lisa to turn off the light

(See the description of morphological tags , syntactic tags and other notations .)

```
"<*john>"
        "john" <*> <Proper> N NOM SG  @ SUBJ
"<closes>"
        "close" <SVO> <SV>  V PRES SG3 VFIN @+FMAINV
"<the>"
        "the" <Def> DET CENTRAL ART SG/PL @DN>
"<door>"
        "door" N NOM SG  @OBJ
"<and>"
        "and" CC @CC
"<asks>"
        "ask" <SVOO> <SVO> <SV> <P/of> <P/for> V PRES SG3 VFIN @+FMAINV
"<*lisa>"
        "lisa" <*> <Proper> N NOM SG  @OBJ
"<to>"
        "to" INFMARK> @INFMARK>
"<turn>"
        "turn" <SVOC/A> <SVC/N> <SVC/A> <out/SVC/N> <out/SVC/A> <SVO> <SV>
V INF  @-FMAINV
"<off>"
        "off" ADV ADVL @ADVL
        "off" PREP  @ADVL
"<the>"
        "the" <Def> DET CENTRAL ART SG/PL @DN>
"<light>"
        "light" N NOM SG  @<P
```

Figure 2.19 A ENGCG window for the sentence "John closes the door and asks Lisa to turn of the light"

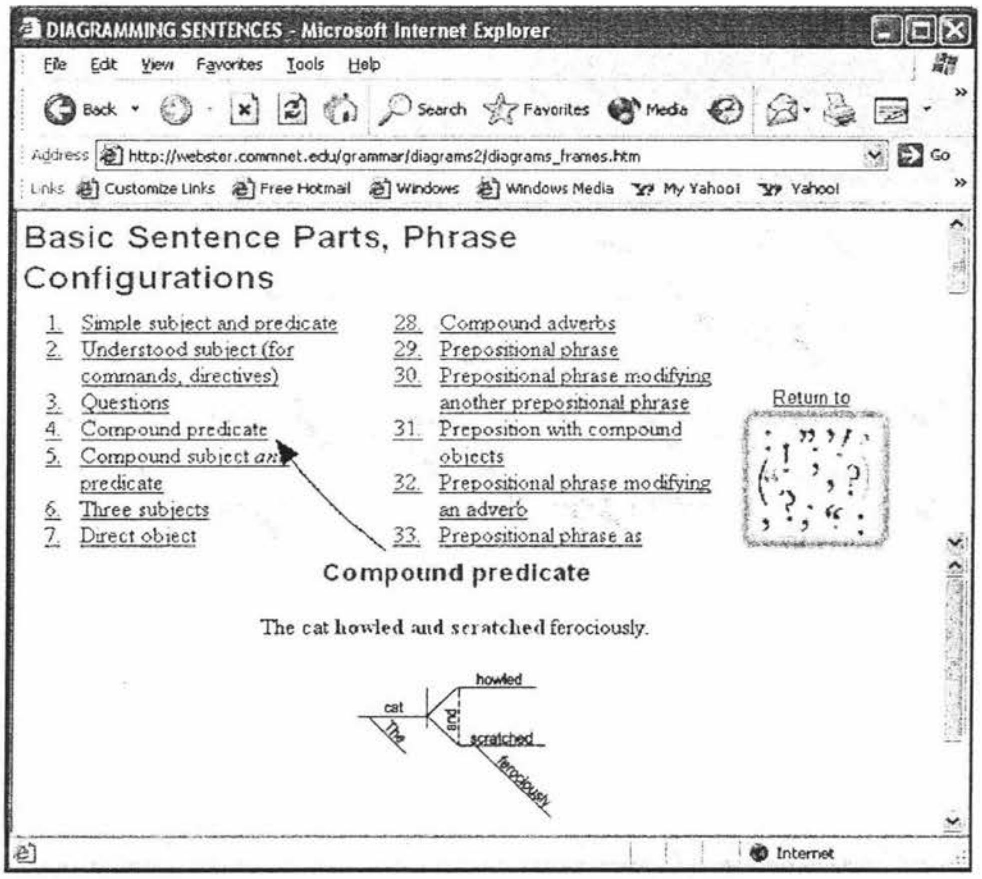
It can be seen that each word is assigned all of its potential syntactical functions and constraints to reduce ambiguities. However, the output of this looks complicated and some words still remain ambiguous (Granger, 1998).

2.3.4.2 Diagram applications

In this type of applications, diagrams are used to presents language structures. The users are allowed to view or interact with the diagrams to learn language syntax. Some applications are used for kids and some applications are used for beginning linguistic students to learn language structure. Several diagram applications are reviewed in the following sections.

2.3.4.2.1 Diagramming Sentences

To learn language grammar, there is a simple method developed by Charles Darling in "Guide to grammar and writing" lessons regarded as an online-learning application (Darling, 2003). This teaching approach is suitable for young kids or beginning students who want to learn natural language structures. In his website, Darling teaches learners to learn language syntax by using diagrams to present different categories of sentence structures. Each page in this website is divided into two areas. The top area displays categories and the bottom area displays a sentence of the selected category and its structure diagram in the form of a Reed-Kellogg diagram (see Section 2.2.2.2.1 for a definition of Reed-Kellogg diagrams). The application allows learners to select any category and then presents any sentence in that category with its structure diagram. Figure 2.20 shows the structure diagram of an example sentence in the category "Compound predicate". By seeing categories with example sentences and their detail structure diagrams, learners can understand simple language structures without difficulties.


DIAGRAMMING SENTENCES - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://webster.commnet.edu/grammar/diagrams2/diagrams_frames.htm
Go
Links Customize Links Free Hotmail Windows Windows Media My Yahoo! Yahoo!
Return to
cat
The
and
howled
scratched
ferociously
Internet

Basic Sentence Parts, Phrase Configurations

1. Simple subject and predicate
2. Understood subject (for commands, directives)
3. Questions
4. Compound predicate
5. Compound subject and predicate
6. Three subjects
7. Direct object
28. Compound adverbs
29. Prepositional phrase
30. Prepositional phrase modifying another prepositional phrase
31. Preposition with compound objects
32. Prepositional phrase modifying an adverb
33. Prepositional phrase as

Compound predicate

The cat howled and scratched ferociously.

Figure 2.20 A diagram for a sentence of the subject "Compound predicate"

2.3.4.2.2 Syntax Student's Companion

The Syntax Student's Companion is a learning program created to help students to practice syntax exercises by visually building and manipulating syntax trees, and writing simple grammars (Max, 2002). The program was developed as the result part of a project at McGill University, London. The main features of the program are (Max, 2002):

- *Visual tree drawing:* Users can draw trees without any limitation of the number of sub-trees for a node. Trees can be downloaded over the network or loaded from the local drive, and saved. Visual tree drawing also provides a simple way of getting tree bitmaps for assignments and handouts.
- *A context-free grammar editor:* Learners can define new simple grammars then check the validation of the learners' input trees following these new

grammars. The application can classify grammars by name, language, and author.

Figure 2.21 presents the user interface of the application (adapted from Max, 2002).

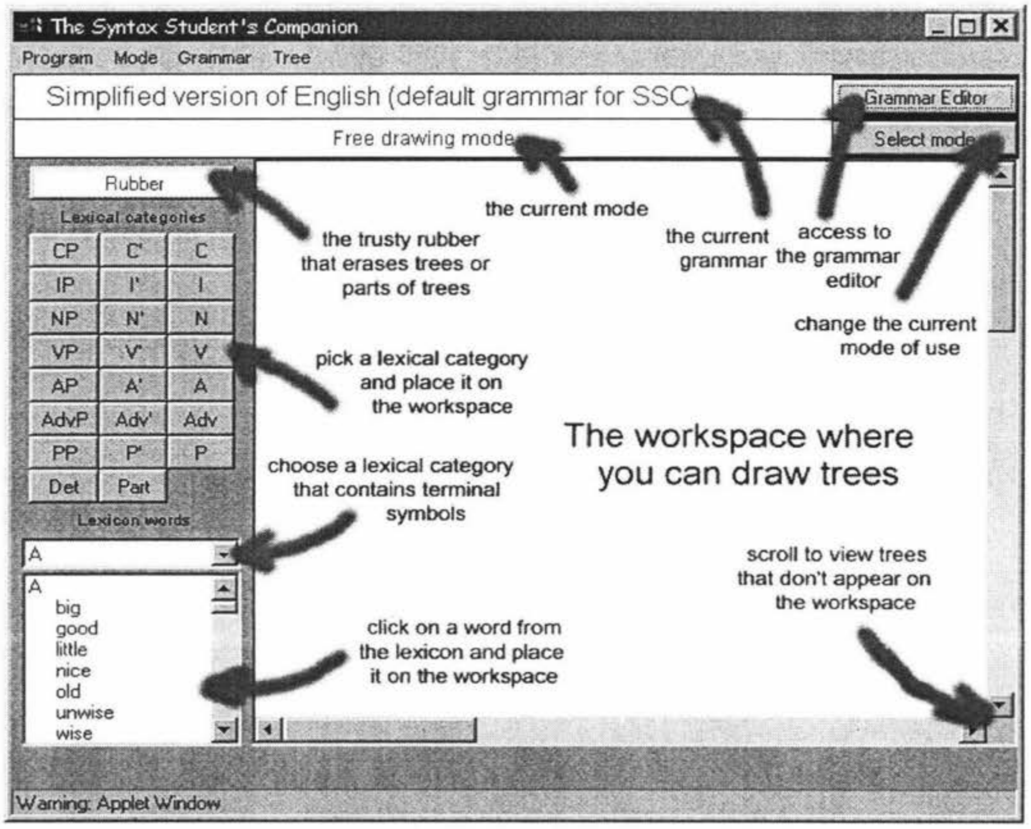

The Syntax Student's Companion
Program Mode Grammar Tree
Simplified version of English (default grammar for SSC)
Grammar Editor
Free drawing mode
Select mode
Rubber
Lexical categories
CP C' C
IP I' I
NP N' N
VP V' V
AP A' A
AdvP Adv' Adv
PP P' P
Det Part
Lexicon words
A
A
big
good
little
nice
old
unwise
wise
Warning: Applet Window
the current mode
the trusty rubber that erases trees or parts of trees
the current grammar
access to the grammar editor
change the current mode of use
pick a lexical category and place it on the workspace
The workspace where you can draw trees
choose a lexical category that contains terminal symbols
scroll to view trees that don't appear on the workspace
click on a word from the lexicon and place it on the workspace

Figure 2.21 User Interface of the Syntax Student Companion

Figure 2.22 displays the application's user interface while a user is doing an exercise of building the syntax tree for the sentence "a boy may drink milk when he is thirsty".

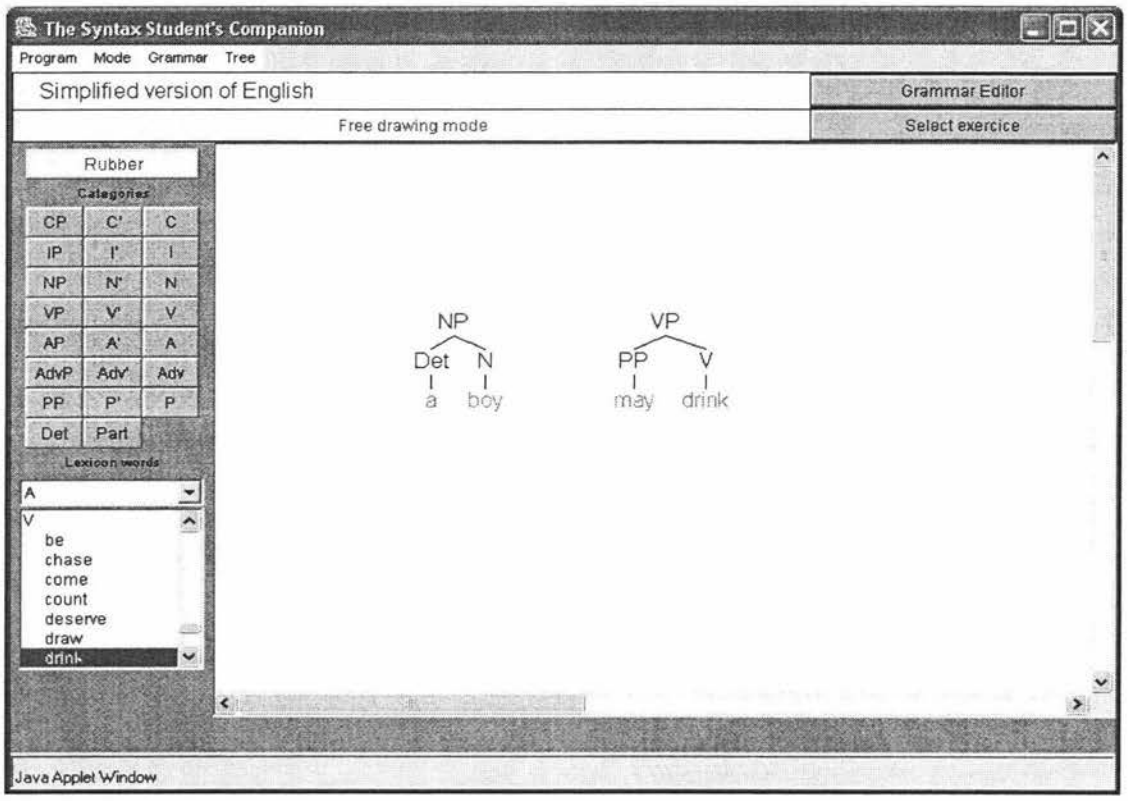


Figure 2.22 Building a sentence diagram in Syntax Student's Companion

2.3.4.2.3 Tree Drawing Tool Animation

Tree Drawing Tool Animation software (TDTA) was developed at the University of Pennsylvania for displaying and manipulating syntactic trees and derivations (Kroch, 2003). It is recommended for the introductory level of syntax learning. The application allows teachers or learners to:

- Demonstrate syntactic structures and derivations in a computer-equipped classroom.
- Construct interactive exercises for students as homework assignments.
- Model syntactic analysis to demonstrate and informally test their descriptive coverage.
- Build trees and paste them into word processing documents or web pages.

The salient point of this application is that it provides a range of available tree "fragments" and lexical items. It then allows learners to select "fragment" trees from a lexicon and combine them to form phrase sentences in a workspace (Figure 2.23). After assembling, the trees can be saved, printed or inserted into any documents. The application can also create grammars following learners' specifications. Learners then can see how the new grammar works by inputting new sentences and observing interpreted trees. Phillips (2003) commented that the program is very useful for learning grammar because of the huge range of available tree fragments and lexical items. He observed that the students who did not use the program in his class for learning grammar lessons could make six times more mistakes compared to students who did use the program.

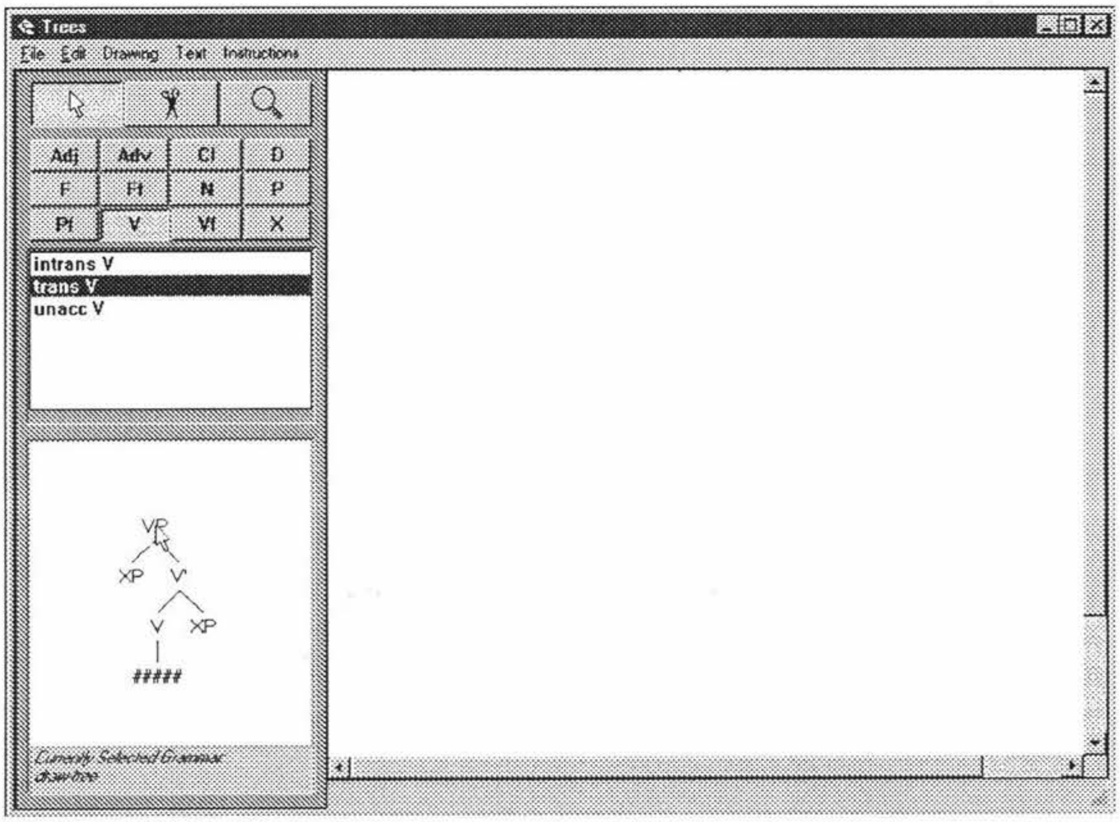


Figure 2.23 TDTA in a process of building a structure tree

2.3.4.2.4 Visual Interactive Syntax Learning

Visual Interactive Syntax Learning application has been developed as the part of a research project at the Institute of Language and Communication in Denmark (Syddansk Universitet, 2003). The project started in September 1996, with the purpose of implementing Internet-based grammar tools for education and research. At the start of the project, four languages were involved: English, French, German, and

Portuguese. Since then, several additional languages have joined the project such as Dutch, Japan and Italian.

The application provides a range of tools for an active learning environment to help learners to learn syntactical functions of sentences. It allows learners to draw syntax trees or to convert sentences into structure trees. Learners can expand or collapse structure trees to see abstract or more detailed presentations of a sentence structure. Each node is a combination of the function of a word, its category and its value (Figure 2.24 in the following page). In general, the application can perform the following tasks:

- Inspecting the tree in abstract or in detailed presentation. Depending on learners' selection, the structure trees of pre-defined sentences can be presented in abstract or detailed presentations.
- Allowing learners to draw new structure trees based on pre-defined grammars;
- Permitting learners to select various tree notations;
- Allowing learners to check the correctness of their input with the pre-defined grammar;
- Calculating the "grade" of learners' performance. The application assesses the learners' understanding of the learning topics after each exercise.

Figure 2.25 shows the interface of the application when a user is doing an exercise. In this exercise, a user inputs a sentence. The application displays the structure tree of the sentence and asks the user to insert the correct categories for each word as well as its function. The application sets a time and calculates the percentage of correctness of the user's answers so that the user can assess his level of grammar knowledge.

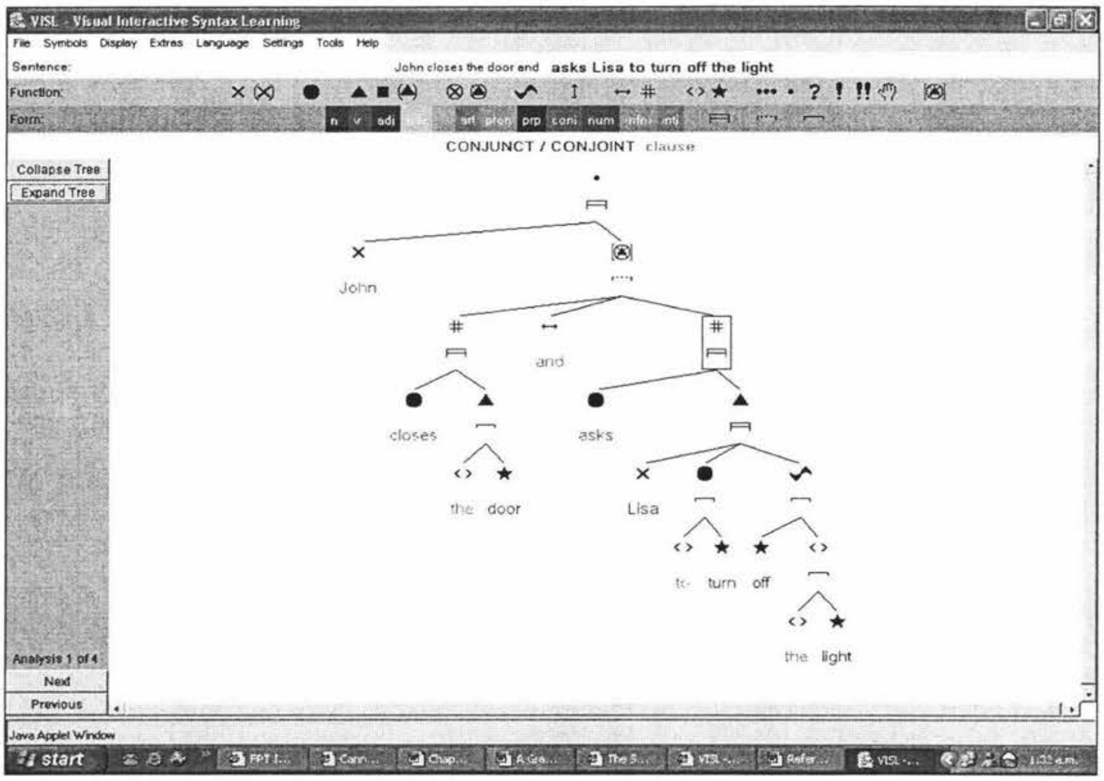

VISL - Visual Interactive Syntax Learning
File Symbols Display Extras Language Settings Tools Help
Sentence:
John closes the door and asks Lisa to turn off the light
Function:
Form:
CONJUNCT / CONJOINT clause
Collapse Tree
Expand Tree
John
and
closes
asks
the door
Lisa
to turn off
the light
Analysis 1 of 4
Next
Previous
Java Applet Window
start

Figure 2.24 Diagram to display functions of the sentence "John closes the door and asks Lisa to turn off the light"

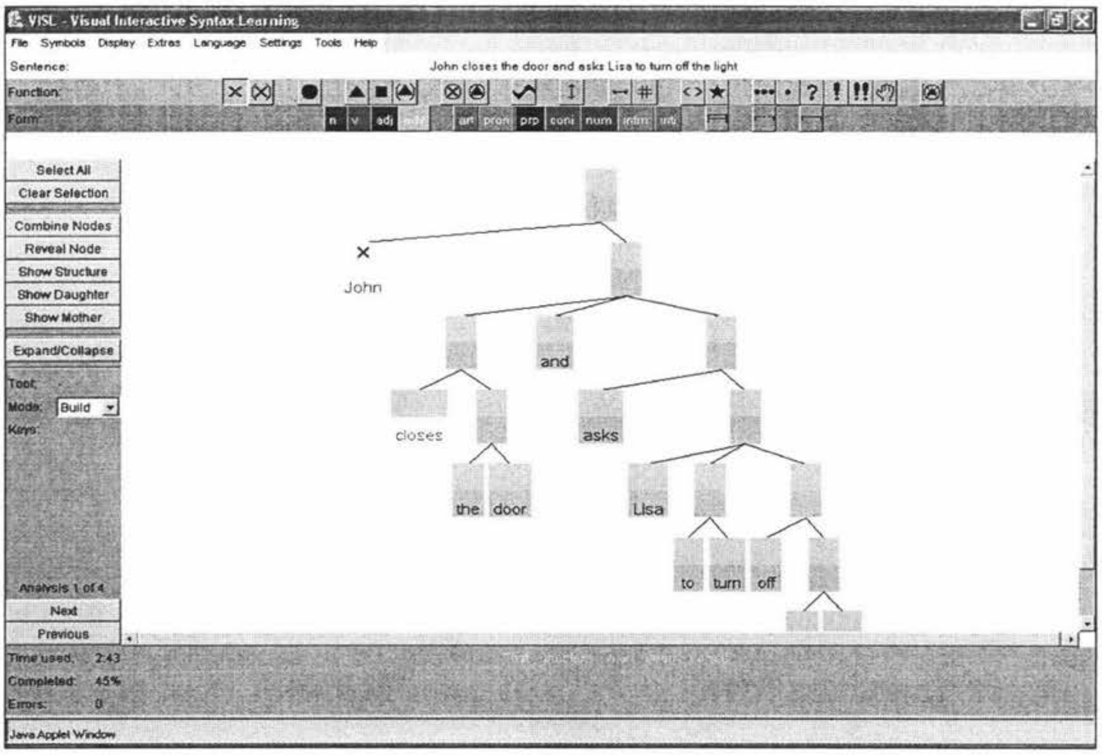


Figure 2.25 A learner is required to insert functions for each word of the sentence "John closes the door and asks Lisa to turn off the light"

2.3.4.2.5 TOSCA Tree Editor

The Tree Editor was designed by the TOSCA Research Group, and implemented as a commercial product by the MOOSE Language Technology Company (Tosca, 2003). This application can take several input formats (texts, SGML-Standard Generalized Mark-up Language) and represent them in the form of structure trees. Differently from trees previously discussed, trees of sentence structures in this application are displayed in a left to right format. The trees use complex nodes to present functions of each word. Figure 2.26 presents the tree created by the TOSCA Tree Editor showing the structure diagram of the sentence "The moose ate the bark" (in a similar form from Tosca, 2003).

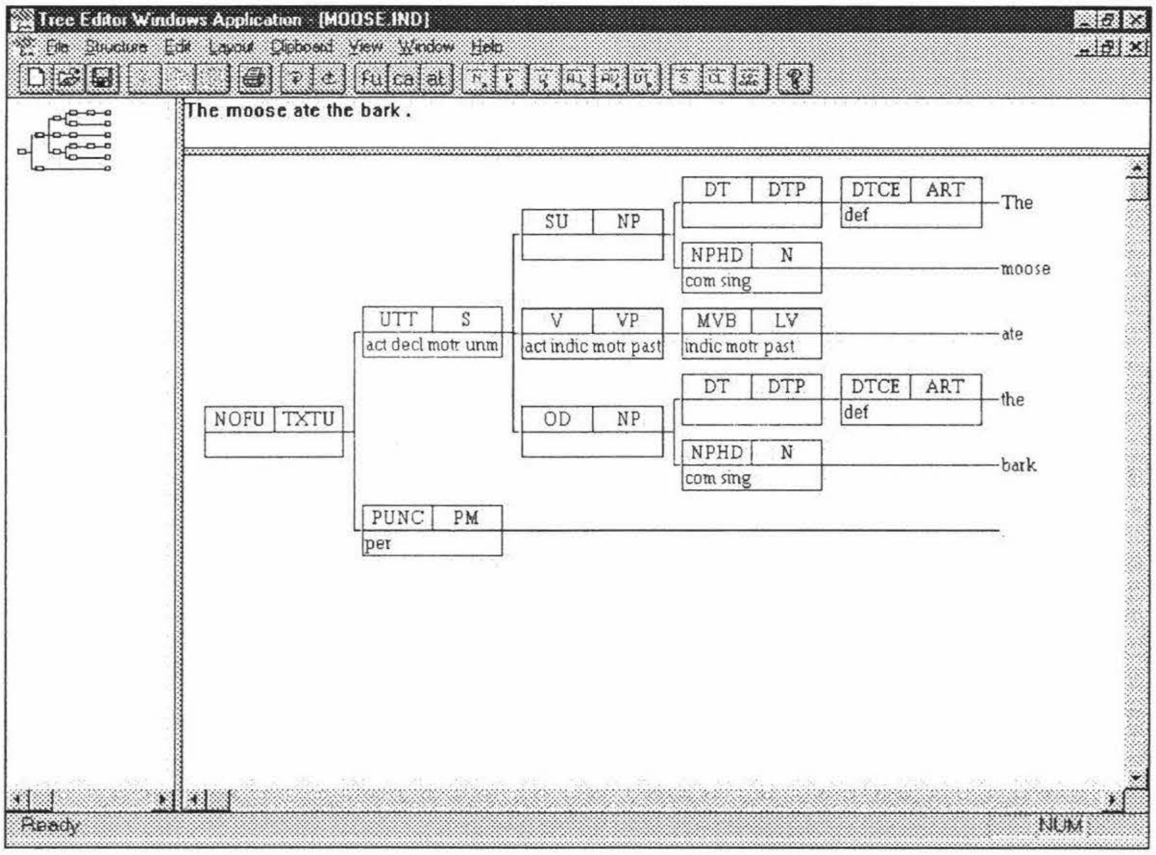
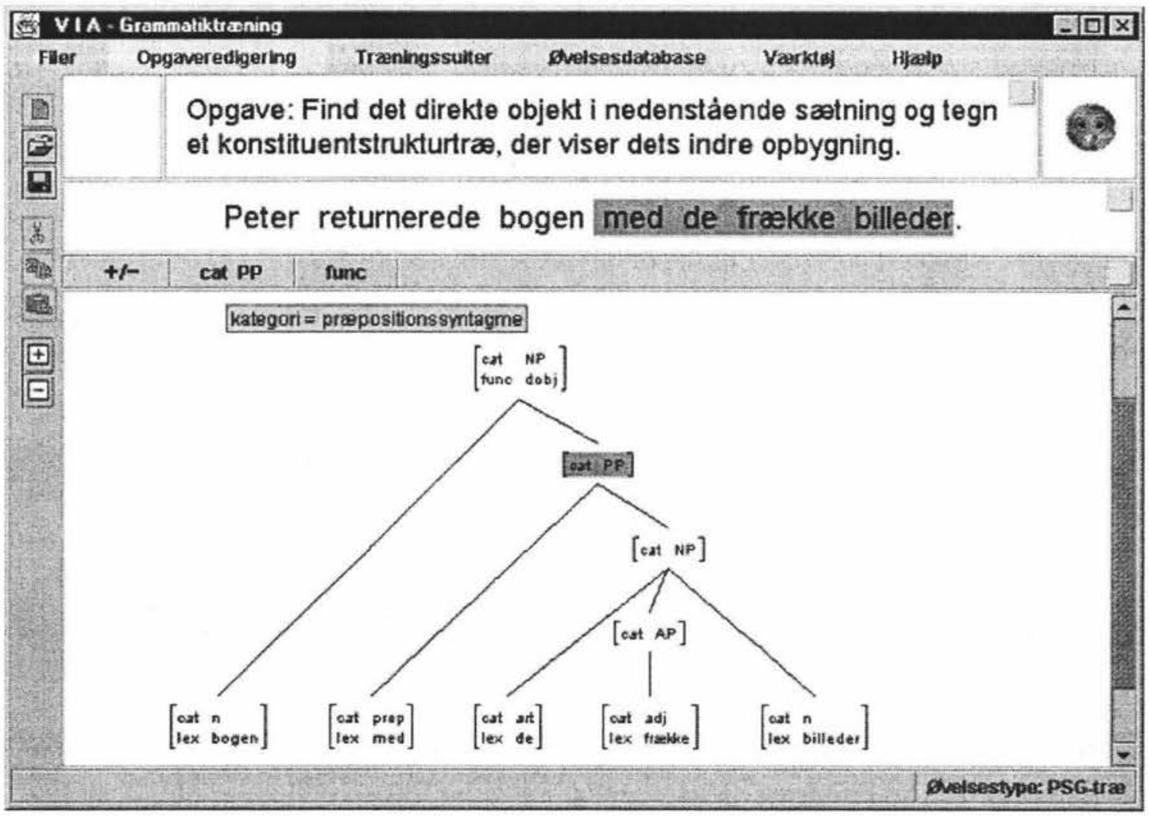


Figure 2.26 Tree Editor Windows for "The moose ate the bark"

The application also provides many exercises to help learners to undertake language structures exercises. For example, the application provides a structure tree and asks learners to assign functions, categories and attribute labels for the selected leaves. The application then checks the correctness of the learner's input and informs them possible errors.

2.3.4.2.6 VIA Grammar Practice

Visualized Interactive Analysis (VIA) is the outcome of a project led by the Department of Computational Linguistics at the Copenhagen Business School, Denmark (Kromann, 1998). The program supports learners in practicing Danish, English, Italian, French and Spanish by providing many exercises. One of the interesting features of the program is that it provides structure diagrams for several grammar theories such as Chomskyan syntax trees, Head-driven Phrase Structure Grammar (HPSG) trees or Phase Structure Grammar (PSG) trees. Figure 2.27 shows the structure tree of a sentence in Danish. Each node in the tree displays the category of a word and its function and content. Learners can interact with the tree by adding, deleting or modifying any node in the tree and then check the validation of the tree following pre-defined grammars.



Opgave: Find det direkte objekt i nedenstående sætning og tegn et konstituentstrukturtræ, der viser dets indre opbygning.

Peter returnerede bogen med de frække billeder.

Figure 2.27 Constituent structure analysis of a Danish noun phrase using VIA

2.3.4.3 Computer application for learning constructed languages

No computer application could be found that teaches "natural language like" languages such as FSCL, CABER or the Observer language. All the applications using these languages provide are syntax checking mechanisms that can verify the correctness of user input by using parsers as in PAC (the application using FSCL) or CABER.

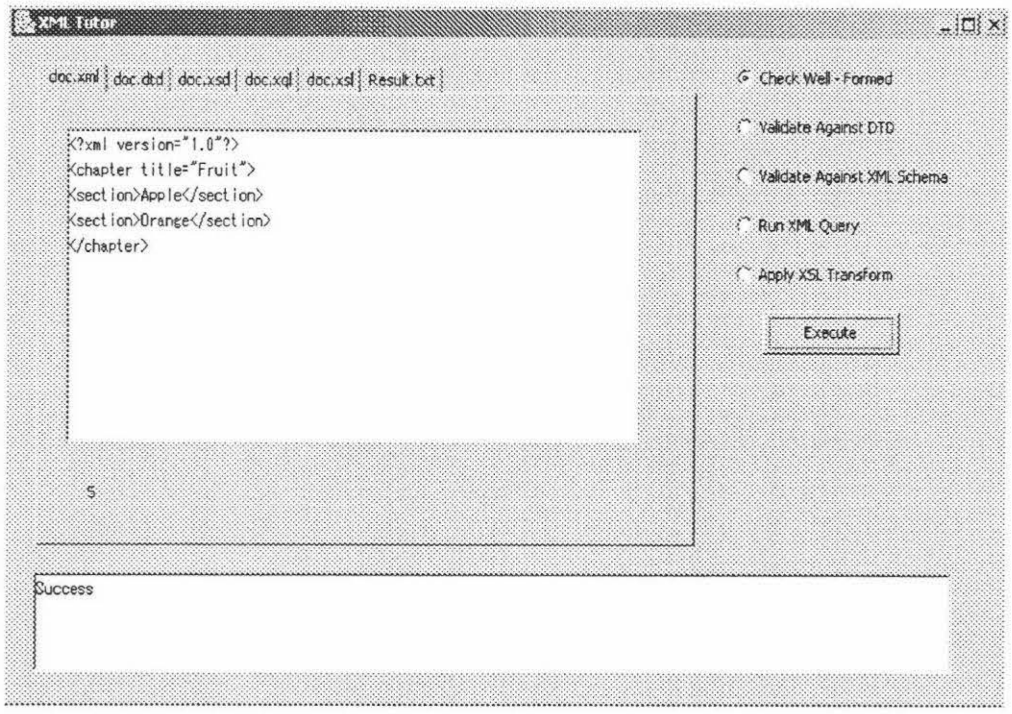
With structured languages such as XML, there are a number of applications that help users to learn XML structure and its associated technologies. This section reviews some applications that assist users to check XML structures, to understand DTD constructions or to be familiar with the query engines, XPath.

XML tutor

The application is developed in Massey University for teaching beginner XML (Massey University, 2004). Some of features of the application are:

- Check XML documents for well-formedness: Learners input XML documents and then verify the correctness of the inputs.
- Validate XML documents against DTD;
- Validate XML documents against XML Schema;
- Perform XSLT transformations: Users can enter a XML document then view how the document can be transformed to another document by using a pre-defined format.

The application utilises the Oracle XML parser, and the Oracle experimental implementation of the XML Query language. Figure 2.28 presents an example of XML tutor application for checking an entered XML document for well-formedness:



```
<?xml version="1.0"?>
<chapter title="Fruit">
<section>Apple</section>
<section>Orange</section>
</chapter>
```

```
Success
```

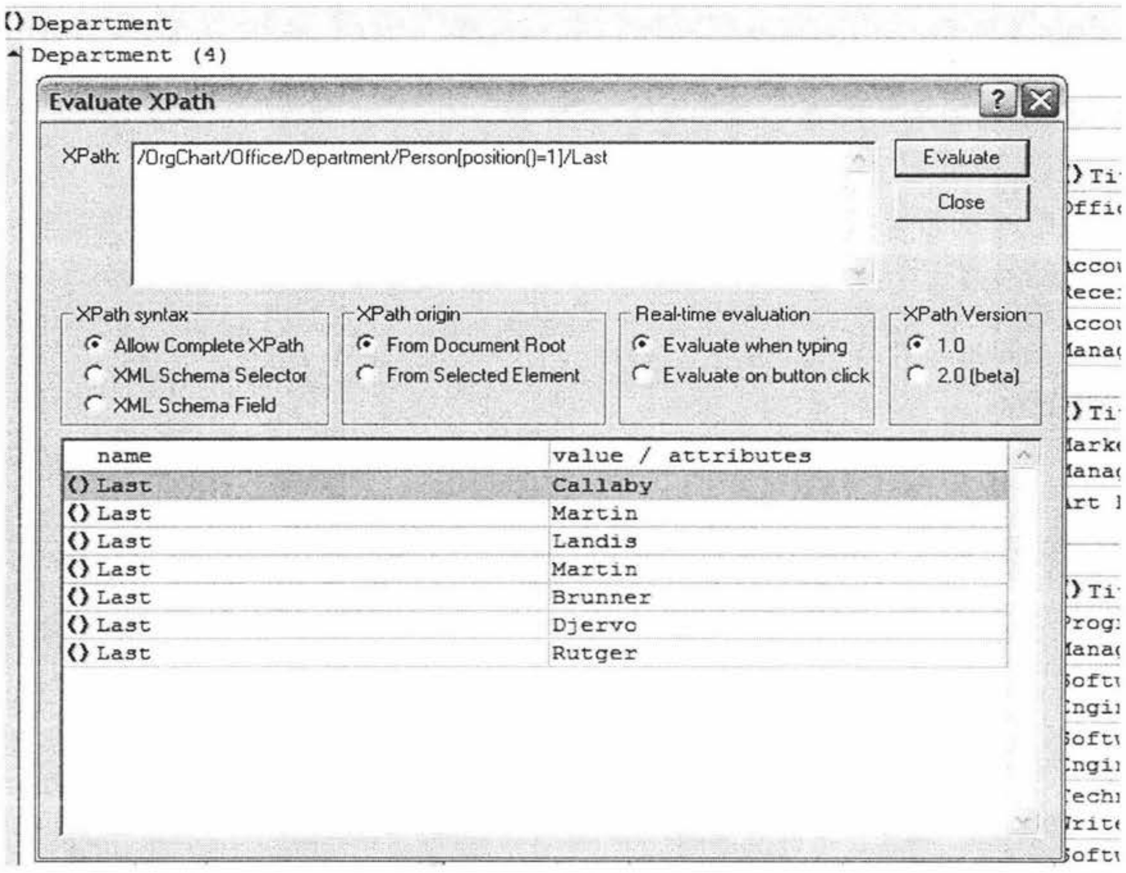
Figure 2.28 An example of checking XML structure

XMLSpy

XMLSpy was developed by the Altova company for "standard XML development environment for modelling, editing, transforming, and debugging all XML technologies" (Altova, 2004, p1). The application provides many powerful functions for both beginners and experts:

- XML checking and validation;
- Well formatedness checking;
- Validation for both DTD and XML schemas;
- Text view with syntax-colouring;
- DTD and XML schemas editing;
- DTD editor, generating XML instance documents based on DTD or XML schemas;
- Generation of DTD/Schema from use-cases.

XMLSpy contains a XPath Analyzer that assists in the building and verifying of XPath expressions. Users can enter an Xpath query and then the application will display matching nodes. Figure 2.29 presents the matching nodes of a XPath query for a pre-defined XML document.



XPath: /OrgChart/Office/Department/Person[position()=1]/Last

XPath syntax: [✓ Allow Complete XPath] XML Schema Selector; XML Schema Field

XPath origin: [✓ From Document Root] From Selected Element

Real-time evaluation: [✓ Evaluate when typing] Evaluate on button click

XPath Version: [✓ 1.0] 2.0 [beta]

name	value / attributes
Last	Callaby
Last	Martin
Last	Landis
Last	Martin
Last	Brunner
Last	Djervo
Last	Rutger

Figure 2.29 An example of a XPath query and matching nodes for an XML document

DTDChart

The DTDChart application is used to draw graphical structure charts for any DTD. The main goal of the application is to assist XML document analysts and DTD designers to visualise the document structures and DTD designs. The major functions of the application are (Intelligent Systems Research, 2004, p1):

- Draws any XML element declaration within a DTD;
- Vertical or horizontal chart display;
- Handles multi-page charts, with page layout shown on underlying grid;
- Optional display of leading comment for each element;
- Optional display of attribute declarations for each element;

To present each element declaration, DTDChart utilises a node with name (and optional comment). Within each node, standard DTD symbols are used to indicate optional element repetition within a document such as * for element may be present 0 or more times; + for one or more times; or ? for 0 or 1 times. The content specifications and optional attribute declarations for each element are shown underneath with appropriate links. Figure 2.30 displays the screenshot the tree structure for a DTD:

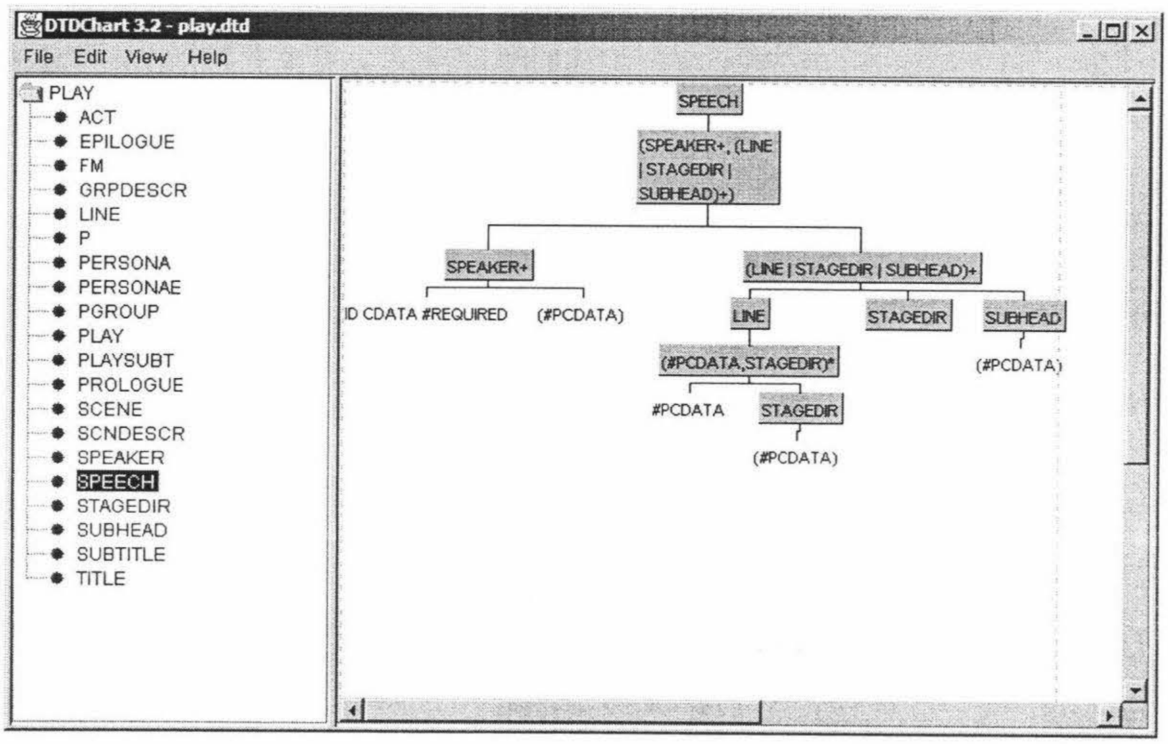
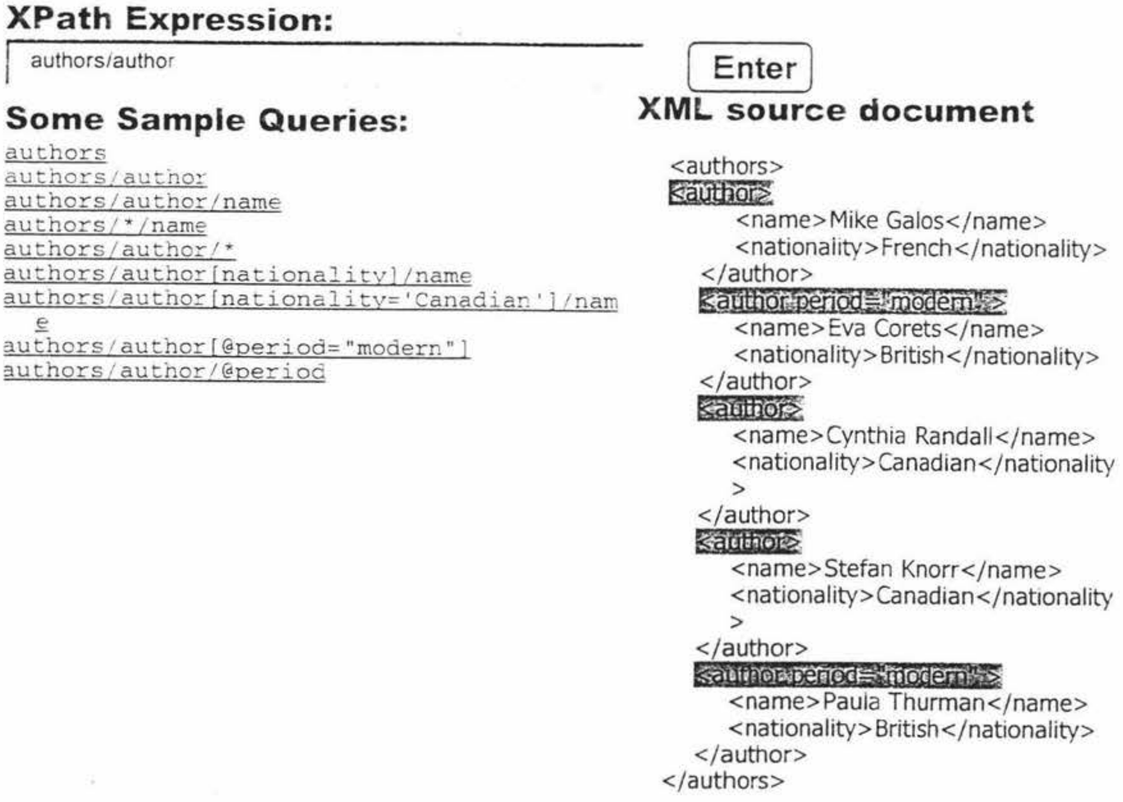


Figure 2.30 An example for DTD chart

MSDN XPath

To help learners to understand how the XPath query engine works, Microsoft Corporation has built an XPath tutorial application (Microsoft Corp., 2004). In this tutorial, users can select any pre-defined XPath query and view the matching pattern in the XML document presented nearby. In addition, users can enter new XPath queries and view matching nodes. By studying the matching pattern, learners will understand how the XPath engine works. If learners enter an error query (following XPath rules), an error message will be displayed. Figure 2.31 presents an example of the input query and the matching nodes.


XPath Expression:

authors/author

Enter

Some Sample Queries:

```
authors
authors/author
authors/author/name
authors/*/name
authors/author/*
authors/author[nationality]/name
authors/author[nationality='Canadian']/nam
  e
authors/author[@period="modern"]
authors/author/@period
```

XML source document

```
<authors>
<author>
    <name>Mike Galos</name>
    <nationality>French</nationality>
  </author>
  <author period="modern">
    <name>Eva Corets</name>
    <nationality>British</nationality>
  </author>
  <author>
    <name>Cynthia Randall</name>
    <nationality>Canadian</nationality
    >
  </author>
  <author>
    <name>Stefan Knorr</name>
    <nationality>Canadian</nationality
    >
  </author>
  <author period="modern">
    <name>Paula Thurman</name>
    <nationality>British</nationality>
  </author>
</authors>
```

Figure 2.31 Demonstration of quering against an XML document using XPath expressions.

The application is a good example explaining how the XPath works. However, it still has limitation as it can only display the first nodes that are the result of the query. For example, the application does not display the node <name>Mike Galos</name> that also is a result node.

2.4 Conclusions about current language learning systems

Based on the reviews presented in this chapter, the following conclusions can be drawn:

- *Wide use of diagrams in presenting language structures*: Language structures can be explained through textual descriptions alone. However, compared with textual descriptions, diagrams have many advantages in enhancing learners' understanding. Therefore, diagrams are usually used in combination with textual descriptions to present language structures for different language theories.
- *Use of computer applications to learn language structures:* Computer applications for language learning (CALL) are used widely to assist users to

understand natural language structures. These applications use diagrams extensively to provide friendly and flexible environments to teach learners' language structures.

- *No application is used for teaching constructed languages:* Most of the reviewed applications for constructed languages only focus on checking syntax, presenting language structures or verifying correctness of user input. No application for teaching constructed language was found.
- *Different strategies to teach language structures:* The reviewed teaching applications use various approaches to help learners the learning of language structures, including:
  1. *Simple applications using passive learning:* With simple teaching applications, learners are only allowed to view structure diagrams from pre-defined sentences. Learners can select any sentence following a language structure's rule to view its details but they are not allowed to create their own sentences. Diagramming Sentences or Constraint Grammar Parser of English is in this category. This type of learning is considered as passive learning.
  2. *Complicated applications using active learning*: The more advantaged applications not only present sentence structure diagrams but also allow learners to interact with these diagrams in different ways such as to build new structure sentences or add, delete and modify current trees to explore possible sentence structures (TDTA, TOSCA and VIA applications). In some applications, learners can choose either textual input or graphical input to create graphical structure trees. Most of these applications are able to check the validity of learner input and notify learners of syntax errors. Many applications give the learners various ranges of exercises (TDTA or VIA application). These exercises can cover different levels of difficulty. The applications also provide feedback or supply correct solutions. Some applications allow learners to set up their own grammars (e.g. Syntax Student's Companion application). Learners then enter new sentences and use the new grammars to validate these sentences. This type of learning is considered as active learning.

2.5 Suggestions for a teaching constructed languages

The reviewed applications are very useful learning tools for learning language structures. The approaches for teaching constructed languages drawn from examining these applications include:

- *Using diagrams and computer applications to teach constructed languages:* Diagrams can present constructed language structures easily. Therefore, they can be used for teaching constructed languages. A combination of presentation of language structure rules, examples and illustration diagrams can be used to teach users language structures.
- *Providing an active learning environment that allows users to interactive with structure trees:* Applications for teaching constructed languages should be able to provide various exercises to create new descriptions (using both textual input and graphical input). They should also provide extensive feedback that supplies critical information such as where there is error, what is wrong and how to fix it.
- *Providing adaptable concepts:* Each structured language is commonly used in one specific computer application for specific purposes. For that reason, one constructed language can be very different from other constructed languages. Therefore, the concepts to learn constructed languages may be similar but the implementation of those concepts for an individual constructed language may be different. For example, the structure of "natural language like" language such as FSCL is very different from the structure of languages presenting metadata such as XML. For this reason, the focuses of teaching applications for different constructed languages are not the same as the following discussion shows:
  - "Natural language like" languages have some similar structures compared with natural languages. This may cause confusion between the two. Therefore, teaching tools for this kind of constructed languages must be able to highlight the differences between the structures of natural language and those for "natural language like" languages such as the FSCL and Observer languages. Teaching tools for NLL languages also need to teach query engines embedded in those languages.

➢ The structured languages such as XML are used for describing documents that have structures. Teaching tools for this kind of constructed languages will teach users how to create correct descriptions or queries following structure or query rules.

2.6 Summary

As so many applications are using diagrams to assist users in the grammar-learning process, it can be concluded that diagrams are one of the important tools to facilitate grammar learning. Compared to the non-diagramming methods such as the bracketing of Phrase Structure Grammar, sentence structure diagrams have many advantages. Exploring efficiencies of diagrams in grammar learning, CALL applications use diagrams extensively to help users to understand language syntax. These programs can be simple applications which parse descriptions into tree diagrams or more complicated applications which allow users to interact with structure diagrams in exciting ways such as building new structure trees or modifying existing structure trees. Most applications provide many exercises so that learners can master grammar easily.

The reviewed applications are valuable examples of the use of graphics or diagrams to support learners in learning natural language grammar. Methodologies used in these applications are very suitable for teaching constructed languages. However, the aims of applications for teaching of natural languages are different from those of applications for teaching constructed languages. Therefore, those methodologies need to be adapted for teaching constructed languages. In addition, constructed languages normally embed specific query engines for searching information efficiently that are not required in the context of teaching natural language. For that reason, teaching applications for constructed languages must be able to teach users to understand this specific feature.

Even though there are many similarities among different constructed languages, each of them still has distinct features. This research now examines in detail an example of a constructed language to investigate what specific features of the language have to be learnt. The research uses FSCL for that purpose since the language contains all specific features of the "natural language like" category of constructed languages for describing and retrieving information as presented in Section 2.3. In the next chapter, this language will be presented in detail including the language structure and its retrieval mechanisms. Further, the suitability of the FSCL structure tree for teaching FSCL is investigated.

Chapter 3

FSCL, a language for description and retrieval

This chapter first reviews the FSCL language in detail. This includes the language's specific structures, a comparison between this language and natural language, and its query engines. The chapter then examines the possibility of a teaching application using diagrams to teach the language's structure.

3.1 FSCL, a "natural language like" language for studying behaviour

The Flexible Structure Coding Language has been developed at Massey University, New Zealand (Heinrich, 1999). The initial aim of FSCL's development was to create a "natural language like" language which is suitable to support the analysis of behaviour. The language was later modified and applied to a new area, the support of computer-based educational systems. FSCL has been incorporated into an information system to support the analysis of multimedia documents called PAC. As it is a "natural language like" language, FSCL contains two essential elements: vocabulary and grammar.

3.1.1 Vocabulary

FSCL's vocabulary is divided into eight categories. These categories are defined following the "word classes" in natural language:

- Noun;
- Verb;
- Article;
- Adjective;
- Adverb;
- Conjunction;
- Preposition;
- Auxiliary.

As will be recognised immediately, the word class "pronoun" which is the one of the categories in natural language, is not part of the FSCL categories. Words of the vocabulary can be stored into sets that are used for studying different topics. Figure 3.1 shows an example of the vocabulary window for a vocabulary set for studying behaviours of children in a classroom. Words that have relationships with other words are defined into hierarchies. For example, as "Jane" and "Lisa" are the names of two girls, users can define the two lower-level words "Jane" and "Lisa" below the noun word "girls".

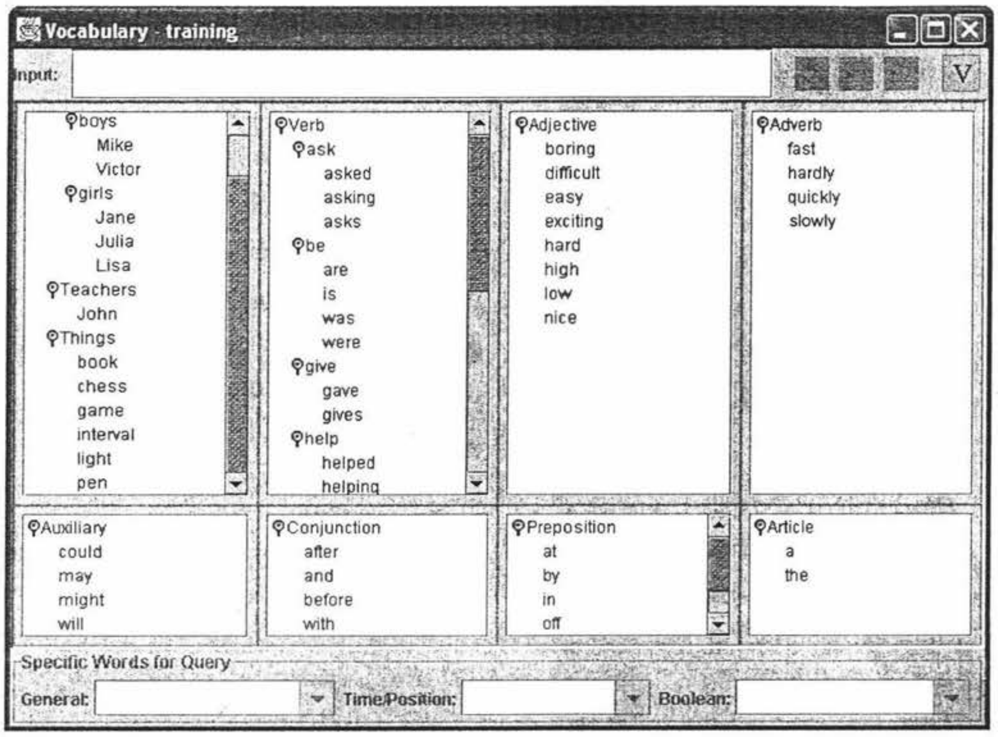


Figure 3.1 PAC's Vocabulary window

3.1.2 Grammar

Like in natural language, FSCL grammar determines how words from different categories can be combined into components to form meaningful sentences. These word groups form subjects, activities and objects in a sentence. A FSCL sentence follows the general structures of "subject-activity-object" or "subject-activity-object-to-verb-object". For example, the two sentences "Lisa reads a book" and "John helps Lisa to read the book" present these structures respectively. FSCL uses a LL-type grammar to identify the relationships of these word components to each other. The LL-type grammar is defined as parsing from left to right. This traces out a leftmost derivation for the input string and uses input symbols for look-ahead (see Louden, 1997 for more details on LL-type grammars). As the grammar allows any descriptive sentence to be separated into components, it supports computer applications in extracting the sentence meaning automatically (Heinrich, 1999, 2000).

3.1.3 Using FSCL to describe and retrieve multimedia information

FSCL is attached to a behaviours analysis system called PAC for information description and retrieval purposes. Users utilise FSCL descriptive sentences to describe multimedia documents. These descriptive sentences are stored into databases. Users then can search for desirable sentences associated with multimedia documents through the FSCL query engine. In detail, the process of using FSCL to describe and retrieve multimedia information includes steps like the following:

- *Collect data*: Multimedia data can be collected in the different forms of text, video or audio format from interviews, or participant observations.
- *Set up vocabulary*: Words need to be defined and stored in a database before they can be used for formulating FSCL sentences. Users can define any word to be used for information description. These words must be put into categories by using a vocabulary window interface. One may define different vocabularies for different topics of study. Users can modify words of the vocabulary in the system at any time.
- *Formulate descriptive sentences:* Users formulate descriptive sentences by selecting the words from a vocabulary. These sentences will be checked for grammatical correctness. Descriptive sentences, which are syntactically incorrect, are either discarded or corrected by users. Each descriptive sentence can be linked with a video segment that presents behaviour. The users will store descriptive sentences in the database inside the PAC system.
- *Retrieve desirable sentences:* After creating descriptive sentences, users can search for specific sentences using FSCL's retrieval engine. FSCL offers different types of queries such as keyword and sentence queries. In addition, FSCL offers some flexibility in queries through the suppression of details or the use of words having relationships. Details of this query mechanism are explained in Section 3.2.3. The query results can be used for different purposes such as for reviewing video segments or analysing activities/behaviours of subjects/actors in a video.

The steps described above can be adjusted depending on the requirements or the purpose of the study. The example below explains the steps of using FSCL to support a behaviour study called "Context of Support: Learning to Read in Small Groups in New Zealand" by Wilkinson and Townsend (1999) as reported in Heinrich (1999).

Studying purpose and data collection

The purpose of this study is to analyse the parameters which contribute to learning to read in New Zealand primary schools. This study included an analysis of the instructional procedures of the reading abilities in a number of selected students over a year. Data was collected in three types: the video recording of reading lessons, audio recordings of the interview with the teachers after these lessons and results of word knowledge tests given to the students before and after the reading lessons. There are three different types of descriptive information. The first type was "word learning opportunities" describing every instance where one of the target words featured in the lesson in any form. The second type was descriptions of "procedural engagement" that captured the attention status of each student. Descriptions for the last type were "substantive engagement" based on the transcript of the interviews with the teachers. The descriptions resulting from all of three descriptive types were combined and interpreted in the context of results from the word knowledge tests.

Building the study's vocabulary

The development of vocabulary was based on the specifications presented by the research members and the different types of descriptive information: word learning opportunities, procedural engagement and substantive engagement. The research members in the research team recognised that words needed for the procedural and substantive engagements were simple and obvious. Therefore, they agreed with a list of words that were needed to insert to the vocabulary. In the case of "word learning opportunities", the research members had to write down a set of sentences they were likely to use for the descriptive sentences. The members in the research team identified the categories of the words involved in the set. They then checked the compliance of these sentences with the FSCL's structures. Words from those sentences were selected and entered into the vocabulary in a form of hierarchies. Within the first few sentences of the descriptive process, a number of new words had to be added or modified. In some cases, the research members added some necessary vocabularies. After this initial phase of the descriptive process, only several words had to be added or modified. Some words were removed from the vocabulary, as they had not been used at all.

Generating FSCL descriptions

Following the study purpose, three types of behaviour descriptions were required including descriptions for procedural engagement, descriptions for word learning opportunities and descriptions for substantive engagement. The processes for generating the first two descriptions are presented as following:

- *Descriptions of procedural engagement*: The descriptions are generated for each target child participating in a lesson in a separate pass through the behaviour recording. Changes in the attention status of a child were described initially as point-of-time information and then converted into periods of time, covering the whole timeline of the lesson. The procedural engagement data were stored in one description set for all target children in a lesson. For example, the descriptive sentences for the procedural engagement would be:

  The name of the child;

  The attention status of the child.

- *Descriptions of word learning opportunities:* The descriptive sentences for this behaviour were generated based on point-of-time basis for each lesson. Words, which were thought to be a challenge to the students, were defined. The children would be tested on their understanding of these words. A descriptive set was commonly created for each lesson. Descriptive sentences for the word learning opportunities might have elements as following:

  The actor - children or the teacher;

  The activity word;

  The objects of the action - children or the teacher;

  The words in the book the word learning opportunity was about.

Some descriptive sentences for "word learning opportunities" of a challenge word "FIND" are:

> Lisa immediately calls out the words after the word FIND on page 3
> John tells Lisa to read again a sentence with the word FIND on page 4.
> John explains the meaning of words relating to the word FIND to Lisa.
> John directs attention to the error on words relating to the word FIND.

Querying of descriptive instances

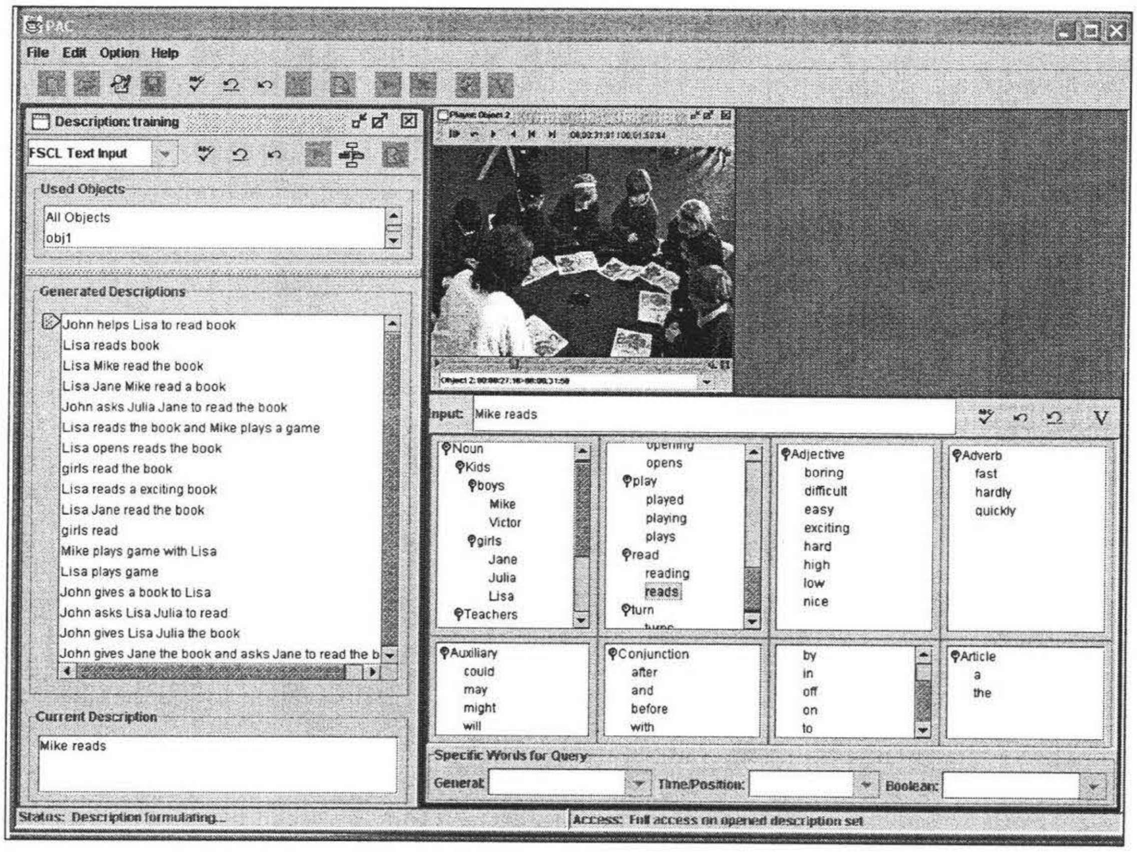
After these descriptive sentences were formulated, several questions could be generated for the studying of "word learning opportunities":

What was the attention status of a child when a word learning opportunity occurred?

Which word learning opportunities occurred for a particular word?

What sequence of child-teacher interaction occurred around a word learning opportunity?

Based on query outputs, the domain expert started to analyse and interpret the resulting data. Figure 3.2 displays the application window during the process of generating descriptions for a video document.



John helps Lisa to read book
Lisa reads book
Lisa Mike read the book
Lisa Jane Mike read a book
John asks Julia Jane to read the book
Lisa reads the book and Mike plays a game
Lisa opens reads the book
girls read the book
Lisa reads a exciting book
Lisa Jane read the book
girls read
Mike plays game with Lisa
Lisa plays game
John gives a book to Lisa
John asks Lisa Julia to read
John gives Lisa Julia the book
John gives Jane the book and asks Jane to read the b

Mike reads

Figure 3.2 GUI of PAC system while generating FSCL descriptions

3.2 FSCL versus natural language

This section describes the similarities between FSCL and natural language and then highlights the differences between these two languages. At the end, the section considers the possibility of using FSCL's structure trees called semantic trees in teaching FSCL's structures and its query engine.

3.2.1 FSCL, a natural language like language

In general, FSCL and natural language share many characteristics. FSCL has eight categories which are similar to those of natural language. The only missing category is "pronoun". However, in FSCL, users can easily replace words of the pronoun category with the words of the noun category (see Section 3.2.2 for more details). The similarities of FSCL's word categories and those of natural language help users, who know the underlying natural language, to learn FSCL quickly. In addition, FSCL's structure "subject-verb-object" is also similar to natural language so it is easy to understand for human readers (Section 3.2.3.2 supplies more details). The similarities of natural language and FSCL can be seen through the following analysis:

There are three essential components for forming FSCL sentences and each component is a combination of words from different categories, including:

Subject: [Article][Adjective] *Noun;*

Verb: [Adverb] *Verb;*

Object: [Preposition][Article] *Noun;*

Note: [] indicates that a word of this category is optional.

A FSCL sentence is created by a combination of those components and is mainly divided into three groups as following:

1. A simple FSCL sentence is in the form of "subject-verb-object".

   Example: John gives Lisa a book;
2. A complex FSCL sentence is formed by simple sentences connected by conjunctions.

   Example: "Lisa reads a book *and* Mike plays a game";
3. An infinitive FSCL sentence is in the form of "subject-verb-object-*"to"*-verb-object".

   Example: "John helps Lisa *to* read a book".

With the examples above, one can see that FSCL structure is very similar to natural language structure.

3.2.2 Differences in structures between natural language and FSCL

Despite the similarities of FSCL structures compared to natural language structures, there are important differences between them in the terms of vocabulary, grammar, and presentation of sentence structures. This section discusses in detail the differences between FSCL and natural language structures.

Compared to natural language, FSCL has some restricted features that *do not*:

- Have the "pronoun" category;
- Allow conjunctions between words within a word group;
- Use the word "not" in negative sentence;

- Support structures which use modal verbs;
- Support the inflection of words;
- Bother about morphology.

- FSCL does not support the use of pronouns such as "he", "she" or "him". FSCL overcomes this deficiency by replacing pronouns with words of the category "Noun". The reason for this omission is to keep the analysis of FSCL sentences simple. This way, a sentence can still be understood by a human reader as well as be "understood" by a computer application. The replacement of a "noun" for a "pronoun" in a FSCL sentence is illustrated by the two following sentences:

  English: *Lisa* reads a book then *she* plays a game.

  FSCL: *Lisa* reads a book then *Lisa* plays a game.
- FSCL does not use conjunctions in a word group. Instead, it provides a different approach described as follows:
  a. Where multiple subjects (multi-subject) perform exactly the same activities on the same objects, FSCL expects that that the subjects are not concatenated explicitly but there is an implicit "and" among them. In other words, FSCL omits connection words in a multi-subject.

     Example:

     English: *Lisa and Jane* read a book.

     FSCL: *Lisa Jane* read a book.
  b. If a subject performs multiple activities (multi-activity) on the same objects, the activities are not joined with a conjunction but an "and" is assumed. In other words, FSCL does not use connection words for multi-activity for the same subject.

     Example:

     English: Lisa *opens and reads* a book.

     FSCL: Lisa *opens reads* a book.
  c. If an activity relates to several objects (multi-object), FSCL assumes that those objects are not joined with a conjunction but there is an "and" between them.

     Example:

     English: Lisa reads *page 4 and page 5.*

FSCL: Lisa reads *page 4 page 5.*

- FSCL does not support negative sentences. However, users can use a different way to describe a negative sentence by using a "not" word which can be defined in the adverb category.

  Example:

  English: Lisa *does not* play game.

  FSCL: Lisa not play game.

- FSCL does not support sentences which use modal verbs or auxiliaries. Users can define modal verbs or auxiliaries in the category "Activity" and use them in combination with other verbs. However, FSCL will not recognise the special grammatical functions of these verbs. For example, with the sentence "Lisa is reading the book", FSCL assumed that "Lisa" does two activities "is" and "reading".
- FSCL does not support inflections of words. However, users have the ability to define words in their different inflections. For example, users can use a hierarchy to define the different forms of the word "play":

```
play
 ├── plays
 ├── playing
 └── played
```

  Then the different forms of this word can be used in FSCL sentences as:

  - Lisa plays.
  - Lisa playing.
  - Lisa played.

- FSCL does not follow the morphology of natural language. The language does not look at the morphological forms of the words and it does not use lexical information to verify the structure of a sentence. It regards a sentence like 'Lisa *play* the game' as a correct sentence. In other words, a 'correct' FSCL sentence may violate natural language rules.

3.2.3 Specific query mechanism in FSCL

FSCL supports users in posting different forms of queries against descriptions stored in a repository. Users can search for information by using the traditional keyword search method. Users can also use a specific query engine embedded in FSCL called "sentence query" technique. By using sentence queries, users can obtain "the correct and complete retrieval of descriptive sentences by basing the query evaluation on the grammar structure of descriptive and query sentences" (Heinrich, 1999, p114). The following section uses textual explanations to outline both FSCL query mechanisms, keyword and sentence queries.

3.2.3.1 Textual description of the FSCL query mechanism

Query mechanisms in FSCL are separated into keyword search and sentence search as following:

- *Keyword queries:* The simplest queries are keyword queries. Users select any word from the vocabulary as a query. The matched sentences are descriptive sentences that contain the keyword. For example, the sentence "Mikes plays with Lisa" matches the keyword "Lisa". In addition, by using the concept of equivalent words, users can find that sentence by using the keyword "girls" (see the following section).
- *Sentence queries:* A more powerful query technique implied in FSCL is sentence queries. In FSCL, a query is also a FSCL descriptive sentence. As FSCL sentences are separated into word groups of components: "Subject", "Activity" and "Object", sentence matching can be seen as the matching between "Subjects", "Activities" and "Objects" of descriptive and query sentences. In other words, a descriptive sentence is matched with a query sentence if and only if the two following conditions are satisfied:
    1. The subject of the descriptive sentence is *same/equivalent* to the subject of the query, and
    2. These subjects of the descriptive sentence and the query *do* the same/equivalent activities on the same/equivalent objects.

  For example, the descriptive sentence "Lisa goes to the library and reads a grammar book" and the query sentence "Lisa reads a book" satisfy the matching condition above. In the descriptive sentence, "Lisa" does two activities: "goes" with an object "to library", and "reads" on another object "a grammar book". In the query sentence, "Lisa" does an activity "reads" on an object "a book".

Following the conditions above, these two sentences match with each other because:

1. "Lisa" is the subject in both descriptive and query sentences – the same subjects;
2. "Lisa" does the same activity "reads";
3. "Lisa" reads the objects "a grammar book" in the descriptive sentence that is equivalent with the object "a book" in the query sentence.

This subsection defines the concept of equivalent words and word groups then explains the features of FSCL's query engine in detail.

Equivalent words: A descriptive word (DW) is equivalent to a query word (QW) if that word is the same word or is at a lower level of hierarchy compared to the word QW. For example the DW "Lisa" is equivalent to the QW "girls" as the word "Lisa" is at a lower level of hierarchy compared to the word "girls". By using this concept, the descriptive sentence "Mike plays with Lisa" will match the keyword query "girls" following the above keyword queries.

Equivalent word groups: A descriptive word group (DWG) is a word group of a descriptive sentence. A query word group (QWG) is a word group of a query sentence. A DWG is considered as an equivalent word group to a QWG only when the DWG has *all equivalent words* to those in the QWG but not versa.

Example:

1. Equivalent word groups for the same words:

   DWG: "Lisa Jane Mike"

   QWG: "Lisa Mike"

   In this example, the DWG has the same words "Lisa" and "Mike" comparing with the QWG. One can recognise that the DWG has one more word "Jane" (e.g. more details).

   DWG: "The red book"

   QWG: "book"

   In this example, the DWG has the word "book" that is the same word with the QWG. The DWG contains words from different categories.
2. Equivalent word groups for words that are at a lower level in hierarchy

DWG: "Lisa Mike"

QWG: "girls"

In this example, the DWG has the word "Lisa" that is at a lower level compared to the word "girls". The DWG also has one more word "Mike" that is a boy.

With this equivalence definition, it can be seen that the order of equivalent words is not important and a DWG can have more details compared to a QWG. Therefore, users can create suppression of details in query, and broaden or narrow search results.

Suppression of detail in query: As an equivalent word group of a descriptive sentence can have more details than a word group of a query sentence, users can decide the level of detail by creating appropriate word groups in query sentences. If a word group in a query has more details, more details are needed for the matching sentences.

Example:

Query sentence:	Lisa reads *a grammar book*
Matching sentence:	Lisa reads *a good grammar book*
	Lisa reads *a grammar book*
Not matched:	Lisa reads *a book*

The sentence "Lisa reads a good grammar book" is matched with the query and it contains more detail compared to the query (a *good* grammar book). The sentence "Lisa reads a book" does not match the query "Lisa reads a grammar book" since it does not have the detail "grammar".

Broadening and narrowing search by using hierarchy: Users can broaden or narrow search results by using words in a query sentence at different levels of hierarchy.

Example:

Query sentence:	*girls* read a book – broadening search
Matched sentence:	*girls* read a book.
	Lisa reads a book.

Lisa reads a grammar book

Query sentence: *Lisa* reads a book – narrowing search

Matched sentence: *Lisa* reads a book.

The reason why the descriptive sentence "*girls* read a book" does not match the query "Lisa reads a book" is that this query narrows the matching result as it only searches for a girl with the specific name "*Lisa*".

In the sentence query category, there is one type of query called the wildcard query. In this type of query, a word can be replaced with a wildcard of the appropriate category. As a wild card has a category, FSCL can check the syntax of the query using that wildcard. FSCL defined three wild card types:

ANY-NOUN;

ANY-VERB;

ANY-CONJUNCTION.

With this definition, a wild card query can be:

ANY-NOUN play

The matching results can be:

Lisa plays.

Lisa Mike play in the garden.

3.2.4 Learning FSCL

It can be seen that, to use FSCL efficiently, users must understand and remember differences of the structures between FSCL and natural language. Users also need to understand clearly the specification of FSCL query engine. Any confusion between FSCL and natural language may lead to the creation of incorrect FSCL sentences or to producing inappropriate queries that can lead to inaccurate returns. There are two common errors created by users. In the first case, users may enter incorrect FSCL descriptions that do not follow FSCL structures. In the second case, users may create FSCL descriptive sentences that are correct in following FSCL structures. However, due to the structures of these descriptive sentences, it might be not possible to take full advantage of these FSCL query options. The examples below illustrate this claim.

Case 1: Users enter descriptions with wrong FSCL structures. This type of errors occurs when users are confused between FSCL structure and natural language structure.

Example 1:

Lisa turns the light off.

FSCL does not allow users to enter sentences that have preposition words after a noun. The correct FSCL sentence is "Lisa turns off the light".

Example 2:

After watching television, John will go to bed.

FSCL does not allow users to enter a sentence that have connection words at the beginning of a sentence. The correct FSCL sentence is "John will go to bed after watching television".

FSCL is equipped with a syntax checking mechanism to eliminate errors belonging to Case 1. This checking mechanism prevents the storage of incorrect FSCL sentences.

Case 2: Users enter sentences with correct FSCL structures but that are unsuitable for retrieval.

Example 1:

Descriptive sentences:

1. Lisa plays guitar *and* chess.
2. Mike plays chess *with* Lisa.

Query sentence:

Lisa plays chess.

In the first sentence, the user utilises the connection word "*and*" for the objects "guitar" and "chess" for "Lisa" who does the activity "plays". This is correct following the FSCL structures. However, FSCL assumes that the word "chess" is a separated subject as it is after a connection word "and". In contrast, in the query sentence, the word "chess" is the object of the subject "Lisa" and the activity "plays". Therefore, this descriptive sentence does not match the query sentence.

The second descriptive sentence is also correct following the FSCL structure rules. However, FSCL assumes that the word group "with Lisa" in this sentence is an object group (with the same function as the word "chess"). In contrast, "Lisa" is the subject of the activity "plays" in the query sentence. Consequently, this descriptive sentence does not match the query sentence.

When the two descriptions above are rewritten as "*Lisa* plays the game chess" and "Mike *Lisa* play the game chess", they will match the query sentence "*Lisa* plays the game".

Example 2:

Descriptive sentences:

1. Lisa was hit and Lisa was crying.
2. Lisa was hit and was crying.

Query sentence:

Lisa was hit and was crying.

In both first and second descriptive sentences, "Lisa" does two activities: "was hit" and "was crying". However, FSCL structures of these sentences are different. The first sentence contains two *simple* sentences "Lisa was hit" and "Lisa was crying" with a connection word "and". Therefore, the activities "was hit" and "was crying" are in two different sentence parts. In contrast, in the second description, "Lisa" does both activities "was hit" and "was crying". Therefore, only the second descriptive sentence matches the query.

From the examples above, it is clear that users must understand how to create descriptions following FSCL structures. They also need to know the limitations of FSCL so they can generate descriptive sentences that are easily retrieved. In addition, users also need to understand how the FSCL matching engine works as discussed in Section 3.2.3.

3.2.5 The FSCL semantic tree

As discussed in the previous chapter, syntax diagrams of a language are used to present language structure or to examine the grammatical correctness of a sentence. Similarly, a simple tree called semantic tree is used to present FSCL structures. The tree can also present semantic roles of each element in a sentence. In addition, the tree can be used to teach users the FSCL matching engine. This section will discuss those possibilities in detail.

3.2.5.1 FSCL semantic tree, a simple syntax tree

As discussed in Section 2.3.2.2.2, Phrase Structure Grammar (PSG) uses Phrase Structure Tree to present the structure of natural language. Figure 3.3 presents the syntax tree of the sentence "John gives a book to Lisa".

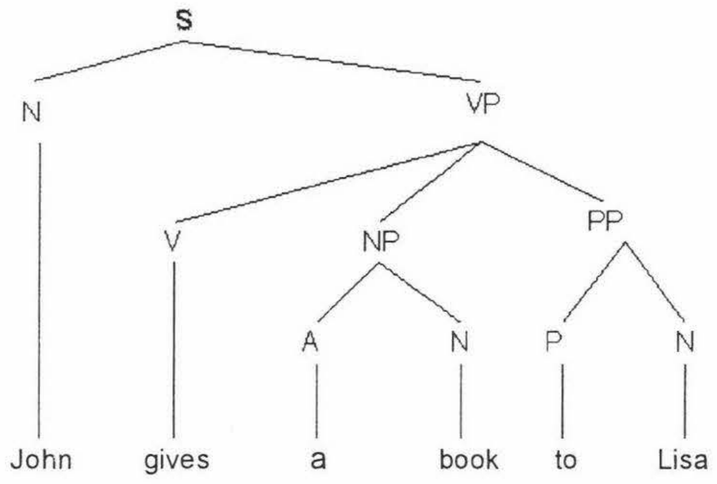


Figure 3.3 A syntax diagram (in PSG theory) for a simple sentence

This syntax diagram can be also presented in a simple form of an abstract syntax tree (Figure 3.4). In this abstract syntax tree, an isosceles triangle presents a word group's function. The article "a" and the noun "book" now present a noun phrase, and the preposition "to" and the noun "Lisa" are a preposition phrase. This tree presents less detail and is simpler than the full syntax diagram presented in Figure 3.3.

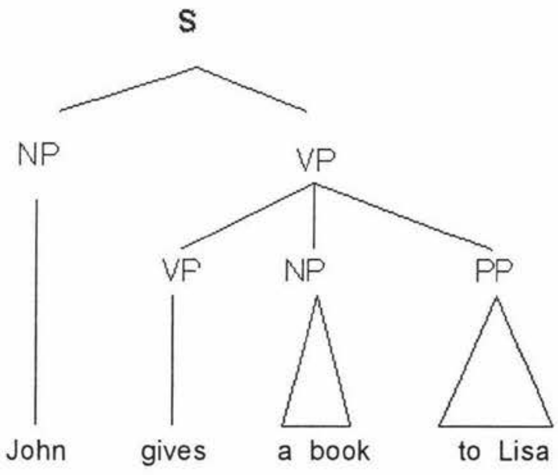


Figure 3.4 An abstract syntax diagram (in PSG theory) for a simple sentence

FSCL also uses trees called the semantic trees to present its structures (first mentioned Section 2.1.2.2). This kind of tree is as well displayed in top-down layout. The FSCL structure tree of the example sentence "John gives a book to Lisa" is shown in Figure 3.5. The highest level is the root S presenting the Sentence. The subject "John" and the verb "gives" are in the second and third level respectively. The two objects "a book" and "to Lisa" are displayed in the fourth level. A FSCL sentence is read by going through the semantic tree from top to bottom, left to right direction.

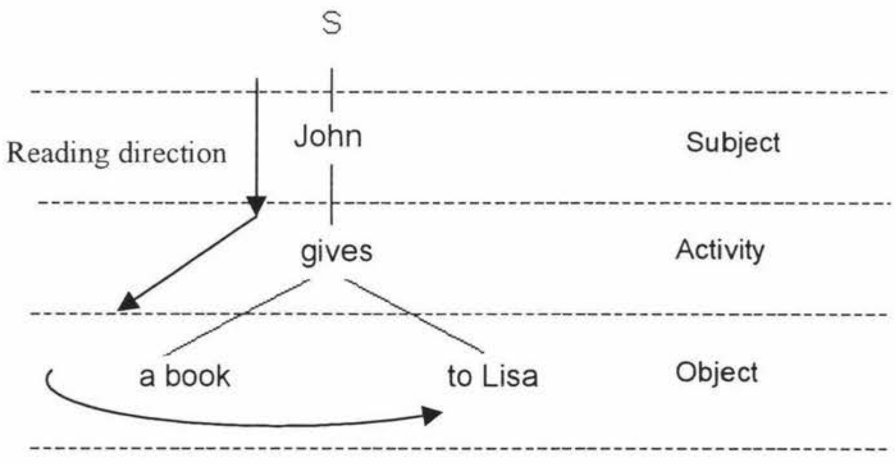


Figure 3.5 The FSCL semantic tree for a simple sentence

When this FSCL semantic tree is presented by a left-right format, one can say that the tree is identical with the abstract syntax trees in PSG theory (Figure 3.6). Therefore, the FSCL is considered as the abstract syntax tree.

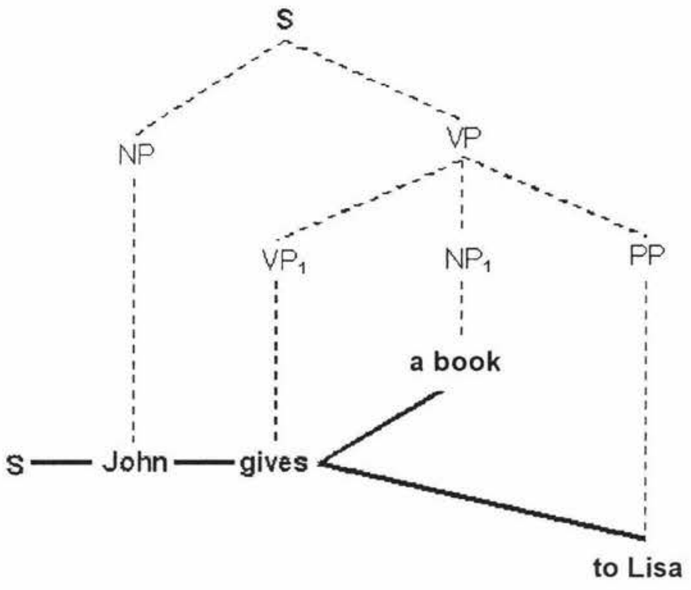


Figure 3.6 Comparison between PSG and FSCL trees

Examining the FSCL and PSG trees, one may notice that the big difference of FSCL trees comes from not having the grammatical information such as NP, VP or PP. Instead, FSCL trees display the roles of word groups in FSCL sentences. This feature is considered as an advantage of FSCL structure trees that helps users to understand FSCL structures for the following reasons.

Users, who will use FSCL for describing and retrieving information, are assumed to known natural language. Therefore, these users do not need to be concerned with grammatical details of FSCL sentences. To learn FSCL, users need a notation to help them to understand FSCL structures so that they can use the language for describing and retrieving information. Diagrams used for analysing language structures such as PSG tree are not suitable as they provide "more information" which is not necessary. In addition, many grammar theories such as the dependency grammar and word grammar do not recognize elements that create a sentence such as NP, VP or PP. Hudson (1984) suggested that there was no need for having PSG structure complements such as NP or

VP if roles of word groups are presented. Therefore, the presentation of syntactical functions, such as NP or VP, in FSCL structure diagram tree can be left out.

3.2.5.2 FSCL semantic tree, a representation of semantic roles of a sentence

A descriptive sentence is normally used to describe a situation or an event. For a given event, it would be possible that different people may use different words for description depending on how they perceive the event. However, no matter how different that given event is described by different people, it always contains an actor or subject that can be a person or a thing, its action or state of being and possible objects/recipient for the subject or its actions. That information represents sentence meaning in logical terms called semantic roles. Regardless to different language structures, semantic roles are presented in every language.

This concept reflects the meaning of descriptive sentences in natural language and reveals "what may be a universal aspect of human perception" (Thomas and Muriel, 1996 p.145) because human tends to see two aspects of events: things and actions. To describe events, the thing that is most prominently involved can be called "topic", "agent", "actor" or "subject" and then the event that contains it or what is said about it can be called "recipient" or "object" about the topic, "activity" of the actor or the "predicate" of the sentence. This perception was once considered as the basis for developing the universal grammar or universal language for humans (Härtl, 2003).

A FSCL semantic tree separates a sentence into components then places these into tree order. The tree is not only considered as an abstract syntax tree but also presents the sentence's semantic roles. This subsection explains how FSCL semantic trees can display that information.

- *Semantic tree for a simple sentence (Group 1 see Section 3.2.1):* The FSCL semantic tree of a simple sentence "John give a book to Lisa" is presented in Figure 3.7.

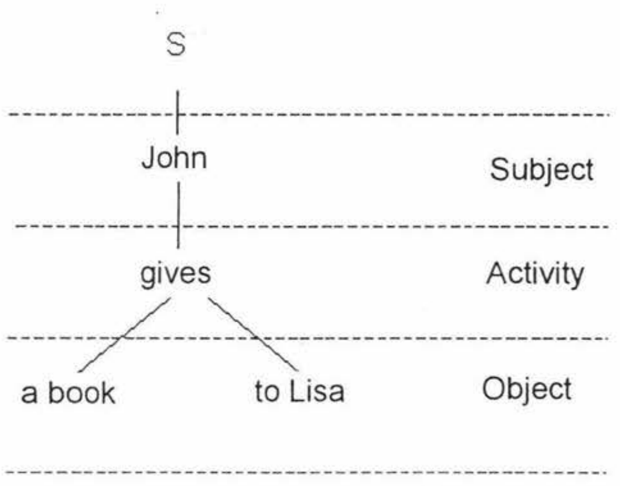


Figure 3.7 A tree diagram for a simple sentence

The diagram divides the sentence into four components with three different roles: a subject, an activity and two objects. The subject/actor answers the question "who gives a book to Lisa" and the activity "gives" answers the question "what John does". The objects "the book" and "to Lisa" answer the question "what John gives" and "to whom John gives". This diagram, therefore, is a presentation of what users need to describe an event. As word groups that have the same roles are placed in the same level in the tree, the number of replications of constituents is reduced. By using the links, the relationship among the subjects, activities and objects in the sentence is also clearer.

- *Semantic tree for a combination sentence (Group 2 see Section 3.2.1):* The FSCL semantic tree of a combination sentence can be seen through an example sentence "Lisa reads a book and Mike plays a game". The functions and relationships of word groups are shown in Figure 3.8.

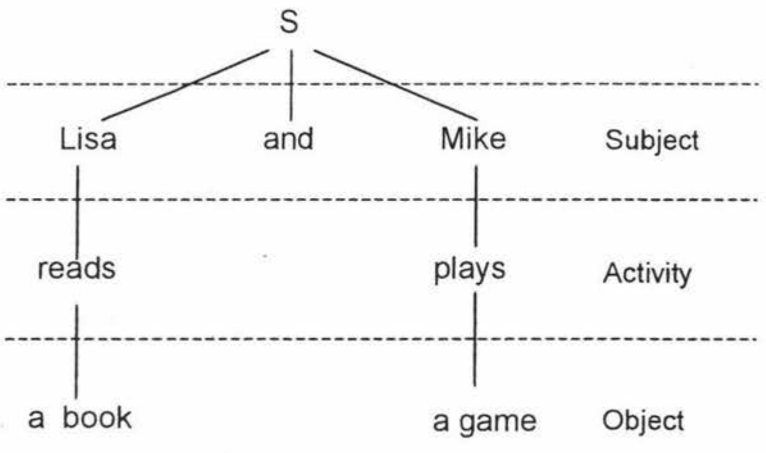


Figure 3.8 Semantic tree for a combination sentence

In this diagram, it is clearly that there are two simple sentences "Lisa reads a book" and "Mike plays a game". Two sentences are linked together by using a conjunction word "and".

- *Sentences with infinitive verbs (Group 3-see Section 3.2.1):* The sentence "John helps Lisa to read the book" is an example of an infinitive sentence. Figure 3.9 shows the semantic tree of this sentence.

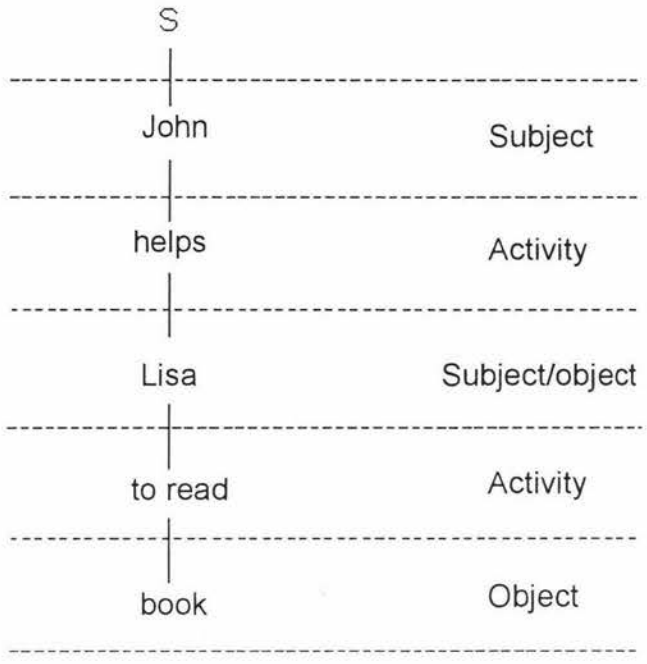


Figure 3.9 A FSCL diagram for an infinitive verb sentence

In this diagram, the subject "John" does an activity "helps". The object "Lisa" is a *direct* object to answer the question of whom "John helps". This object is also the subject for another clause "Lisa to read the book". The infinitive verb "to read" is the activity for the subject "Lisa". In this case, "John" does an action and "Lisa" needs to do another action. The word "the book" is the object for the activity "to read".

From the reviews above, one can conclude that using FSCL semantic trees can help users to understand the FSCL structures easily. In that tree, semantic roles of subjects, activities or objects/recipient of each sentence are obvious. Furthermore, the relationship "subject-activity-objects" is clearly identified when following the branches of the tree. Therefore, users can understand easily how to form a FSCL sentence from words.

3.2.5.3 FSCL semantic trees to present matching patterns

Using FSCL semantic trees not only helps users to understand FSCL structures but also assists them in understanding the language's query engine. As discussed in Section 3.2.3.1, the requirement for matching between a descriptive and a query sentence is that they must have an equivalent "subject–activity-object" relationship. As a FSCL semantic tree presents the "subject-activity-object" structure of a sentence, the equivalent of "subject-activity-object" relationships between descriptive and query sentences is easily recognised through finding matching tree branches between these two trees. Therefore, to compare a query sentence and a descriptive sentence, both of them will be presented as the semantic trees of "subject-activity-object" relationships. The nodes presenting "subject", "activity" and "object" between the two trees are then compared for possible matching.

Figure 3.10 in the following page displays an example to explain this concept more clearly. The two semantic trees (descriptive tree 1 and descriptive tree 2) present the structures of the two descriptive sentences "Lisa opens the book and reads the first page" and "Lisa plays the game and Mike reads a book". Now the sentence "Lisa reads" is used to query these two descriptive sentences. The semantic tree of this query is presented in the lower left half of the figure. Checking the three semantic trees, one can see easily that the descriptive tree 1 has the same pattern "Lisa reads" as the query tree. The second descriptive tree does not have the "Lisa reads" pattern, therefore it is not a "result" of the query sentence.

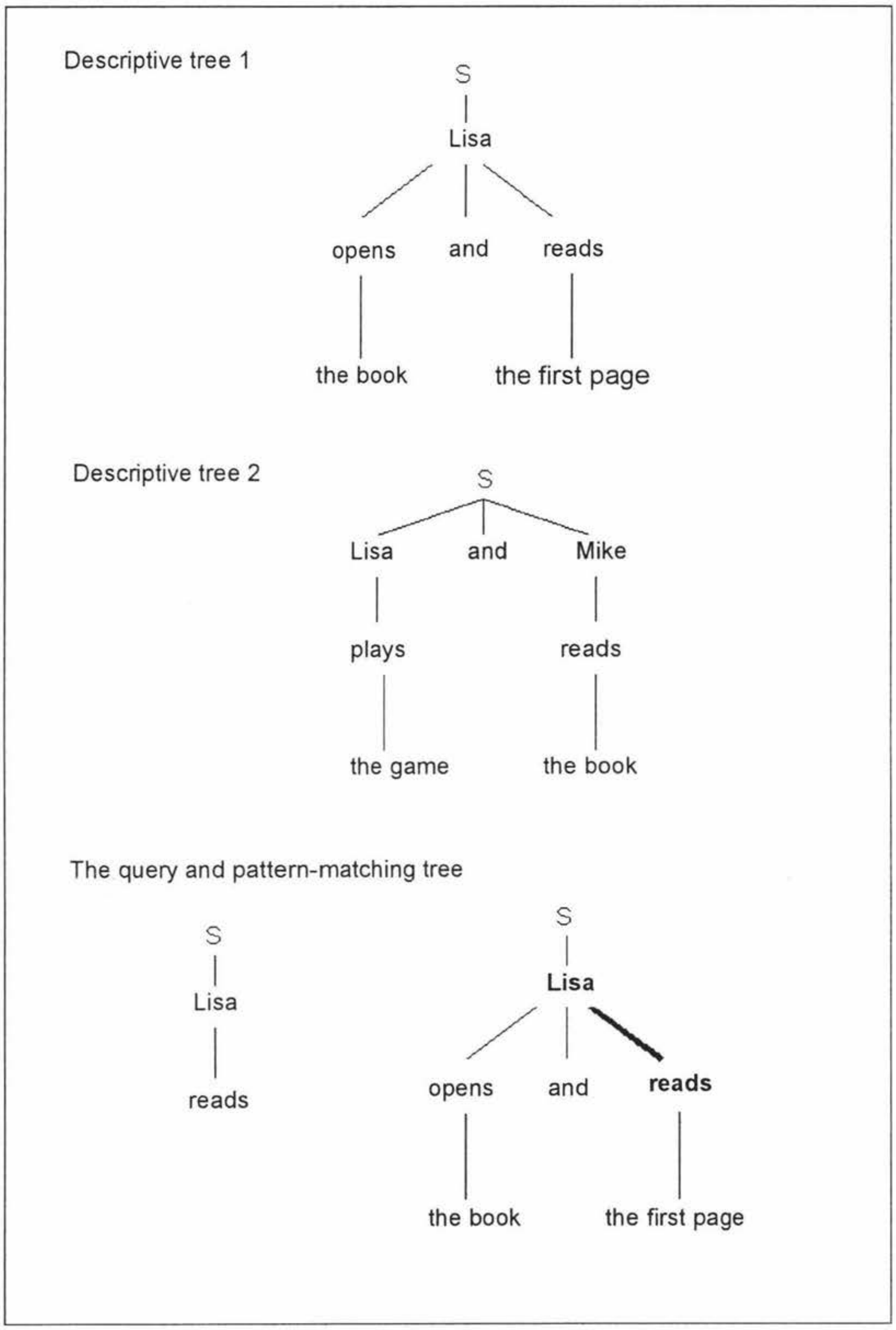


Figure 3.10 A matching pattern between query and descriptive trees

One important notion drawn from viewing the diagram above is that users can search the sentence "Lisa opens the book and reads the first page" by using any sentence that has a matching pattern with this descriptive sentence. For example, query sentences such as "Lisa opens" or "Lisa opens and reads" can be used for queries since they have the matching patterns with the descriptive sentence.

Given the example in Section 3.2.4 in textual representation, users may not understand why the descriptive sentence "John plays chess with Lisa" does not match the query sentence "Lisa plays chess". When those sentences are presented by semantic trees, users can see easily why these descriptive and query sentences do not match (Figure 3.11).

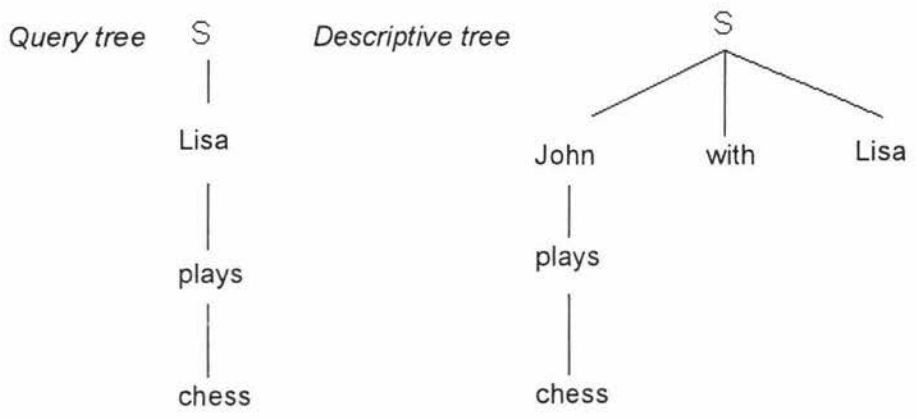


Figure 3.11: No matching pattern between descriptive and query trees

The illustrations above prove that the presentation of matching patterns between query trees and descriptive trees can help users to understand the FSCL query engine easily. It also facilitates to avoid users creating descriptive sentences that are correct following FSCL structures but difficult for retrieval.

3.3 Summary

FSCL bears many elements of natural language. However, there are major differences between the structures of these languages. In addition, FSCL embeds a specific query mechanism that helps users to search correct descriptive sentences that have the same structures as queries. Therefore, there is a requirement for a teaching method for teaching users both FSCL structures and its specific query engine.

Diagrams presenting FSCL semantic trees can be used to support users understanding of the distinct features of FSCL. The reason for this is that users can understand two important factors at the same time when studying semantic trees: the simplicity of an "abstract" syntax tree and semantic roles of elements in a sentence. Presented as an "abstract" syntax tree, the tree suppresses information that is not required in this context and helps users to focus on the similarities and differences between FSCL and natural language structures. The semantic roles of elements in a sentence allow users to create FSCL sentences following the nature of human in describing events. In addition, the visualisation of the matching patterns between query and descriptive sentences supports the users in understanding the FSCL query engine much more easily than explanatory text.

In the next chapter, teaching concepts to teach constructed languages based on the structure diagrams of constructed languages are developed. FSCL and its semantic tree structures discussed in this chapter will be used to illustrate the concepts' development process.

Chapter 4

Concept development

This chapter discusses the concept development for a visual teaching application called Visual Interactive Structure Learning (VISL) that assists users in learning constructed language structures and their query-matching engines. The chapter also explains how these concepts are applied to a specific constructed language, FSCL, which has been introduced in the previous chapter.

4.1 VISL, a unified approach to teach structured languages

The concepts for an application to teach constructed languages are based on the conclusions drawn from the literature review and an example language, FSCL. The following factors are considered:

- Advantages of diagrams in enhancing learner's understanding of language structures;
- Advantages of using computer applications in supporting the learning of language structures;
- Effectiveness of active learning in assisting learner's understanding and in their long term memory;
- Benefits of diagrams in presenting matching patterns between descriptions and queries in enhancing user understanding of the query engines of constructed languages.

The teaching concepts suggested for VISL include passive learning and active learning to help users to mastermind constructed language structures and their query engines:

- *Passive learning*: Explains language structures and their matching engines through lessons and illustrates these lessons through example descriptions and structure diagrams;
- *Active learning*: Provides an interactive learning environment in which users work with structure diagrams to enhance their understanding of both language structures and query engines.

The following section explains in detail the purpose of these passive and active learning approaches for helping users with learning constructed languages:

- *Understand structures of constructed languages for description by using passive and active learning:*
  - *Passive learning*: At first, users will not know the language structures. Therefore, the language structures will be presented by textual explanations accompanied with illustrative diagrams so that users can understand how descriptions are structured. Users will be familiarised with simple descriptions through viewing their tree structures. This passive learning approach is presented as "Visualized Structure Tree VST" mode.
  - *Active learning*: Users tend to forget what they have learnt by observation but they can remember and gain deeper understanding when they themselves have practiced with the material. In this active learning model, users are allowed to interact with structure trees by either building structure trees by both graphical and textural inputs from scratch or by modifying existing ones so that they can be actively involved in exploring possible language structures. This learning approach is called "Structure Tree Manipulation STM" mode.
- *Understand query-matching engine for retrieval by passive and active learning:*
  - *Passive learning*: Users work through textual explanations that are supported by diagrammatic illustrations. Conditions of the matching mechanism are presented by examples and illustrated by matching locations between query and descriptions. Through studying those explanations and examples, users can gradually understand how descriptions are satisfied with the requirements of a query. This passive mode is called "Visualized Query Matching VQM".
  - *Active learning*: Users can learn better when they can actively explore the ideas and formulate their own solutions. Therefore, in a further step of learning, users will practice what they have understood in the passive learning mode in an active learning environment. Users are allowed to build a new query and then learn the matching locations between query and descriptions. From these exercises, users will gain the necessary skills for creating suitable

queries to get the best returns. The project names this mode as "Visualized Query Building VQB".

The relationships among these modes are presented in Figure 4.1. Mode VST and Mode STM use the "Structure tree presentation" process to display the structure of descriptions. Mode VQM and VQB use the "Matching pattern display" process between descriptive and query trees to explain how these are matched to each other. The "Matching pattern display" process is derived from the "Structure tree presentation" process.

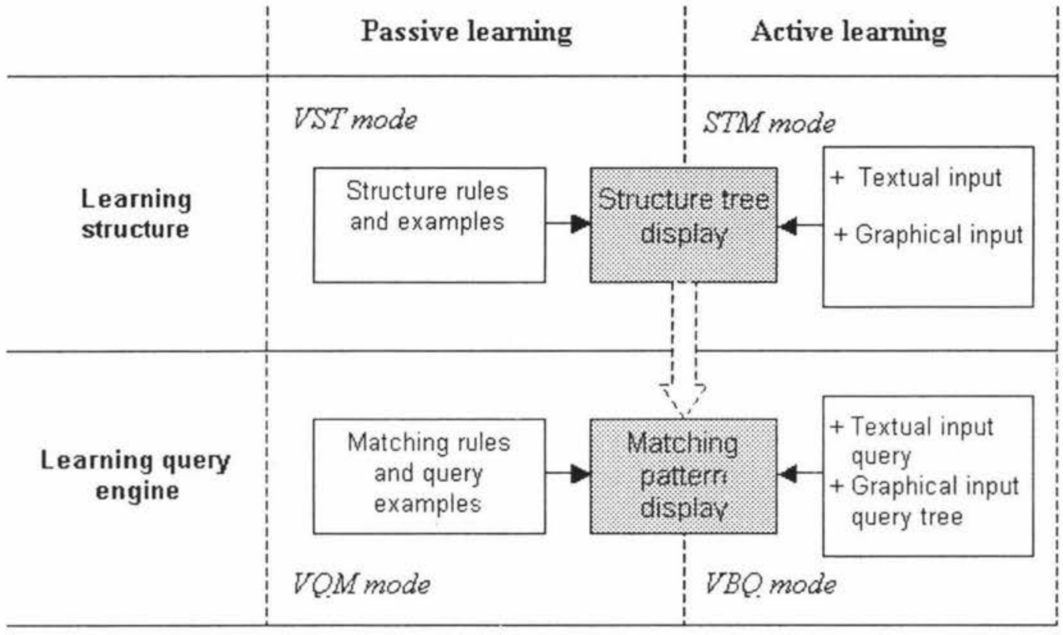


Figure 4.1 Relationship among modes of VISL

The skeleton of the VISL architecture based on the concepts above is presented in Figure 4.2. The four modes within VISL are seamlessly integrated to support users step-by-step in learning constructed language structures and their query engines.

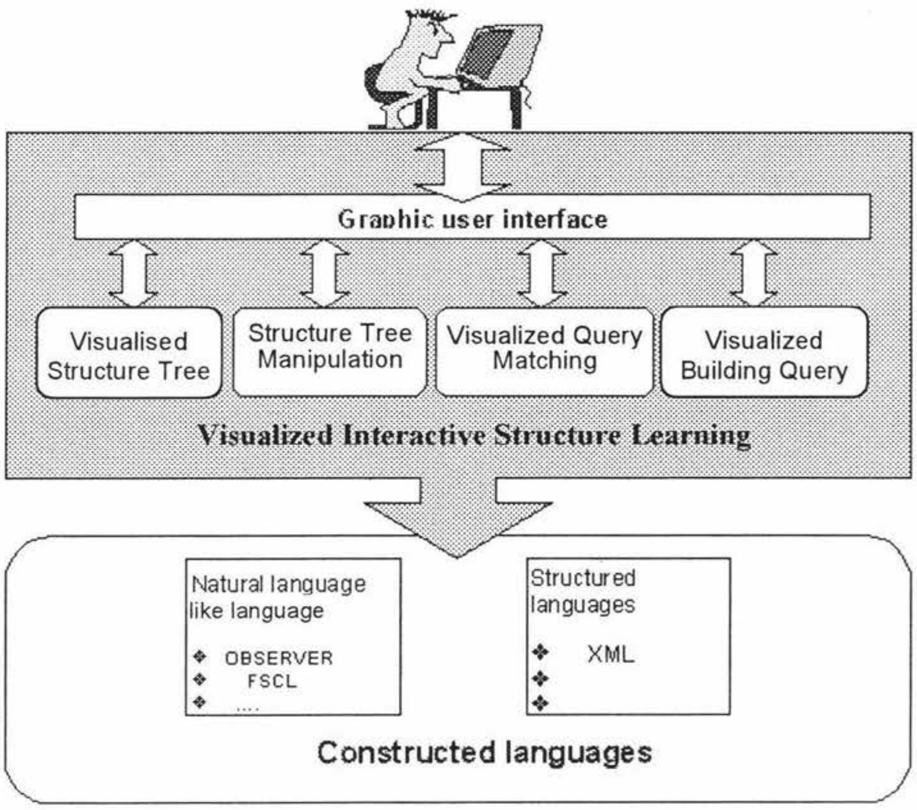

Graphic user interface
Visualised Structure Tree
Structure Tree Manipulation
Visualized Query Matching
Visualized Building Query
Visualized Interactive Structure Learning
Natural language like language
❖ OBSERVER
❖ FSCL
❖ ....
Structured languages
❖ XML
❖
❖
Constructed languages

Figure 4.2 Visualized Interactive Structure Learning (VISL) architecture

4.2 Details of VISL concepts to teach constructed languages

This section presents in detail how the learning concepts of the previous sections are developed into four modes. FSCL is used as the illustration to explain these concepts. The first two modes, VST and STM, have been implemented for FSCL as a training tool. The figures in the respective sections have been produced from screen shots of this implementation. Implementation details are described in Chapter 6. Figures used in the second two modes, VQM and VBQ, are conceptual diagrams. The discussion of each VISL mode is divided into three parts: the teaching concept, an illustrative example and specific processes inside that mode.

4.2.1 Visualized Structure Tree mode

This section includes three subsections: the concept of Visualized Structure Tree (VST), an example using FSCL and the processes happening inside VST.

❖ *Concept of VST*

When using a constructed language for the first time, users will not know its structures at all. With a "natural language like" language, a user may ask: Is the structure of the language different from or similar to natural language? How is a word group created or how is a sentence created from different word categories? What are specific features of the language that I need to know? The VST mode provides this information using the following approach:

- *Display language structure diagrams with predefined examples:* As users will not know the structures of the language to be learnt, the mode firstly provides predefined simple examples of language structures and allows users to select any example from a list to view its structure. This gives users an overview about the language structures.
- *Describe sentence structures with textual explanations:* The mode teaches users the language structures by introducing the structure rules of the language. Each structure rule will be presented by an explanatory text.

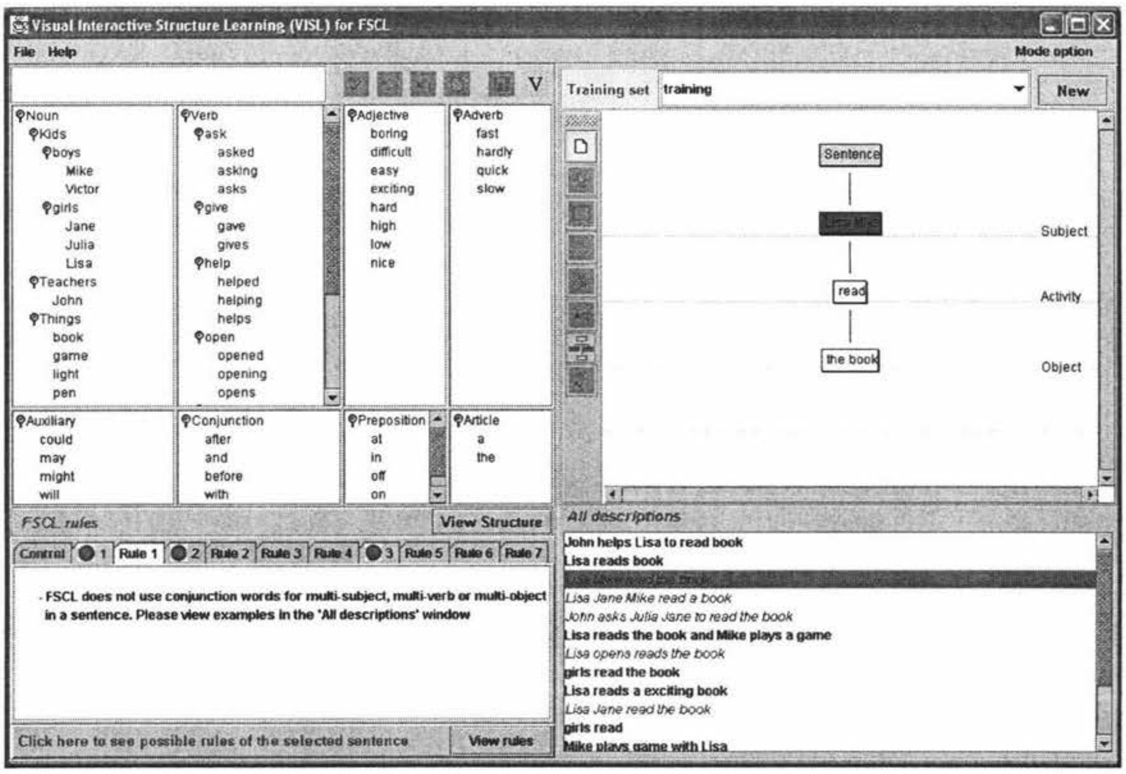
  To teach FSCL, rules like the following can be presented to users:

  - A subject of a FSCL sentence is created by a number of individual noun words without any connection word;
  - FSCL does not care about morphological forms of words.

  The mode only presents simple sentences, but it outlines as much as possible the differences between FSCL and natural language structures. Through these descriptive texts, users will know how components in FSCL are created and will learn the differences between natural language and FSCL.
- *Illustrate structures of the language to be learnt by presenting its structure diagrams*: The conclusion from Chapter 2 pointed out that diagrams are extremely useful to help users to understand language structures. That conclusion is further emphasised by Mayer et al. (1995): "when illustrations were placed alongside texts and contained annotated captions of the information from texts, students' recall and comprehension improved". Therefore, VST shows an illustrative diagram for each structure rule of a constructed language. By studying the explanations of rules and seeing in parallel the graphical display of sentence structures, users can understand clearly the meaning of these explanations and gain better cognition about the language structure. That is because diagrams are able to "bring out more detailed, knowledgeable, responsive, awareness to the object, situation or text being communicated" (Canning-Wilson, 2001, p1).

With FSCL, graphical representation of the language structures provides both overview and details of sentence structures with interrelationships and interdependencies among elements. Chapter 3 has explained that by viewing a FSCL semantic tree, users can immediately recognise the semantic roles such as the subject, verb and object of each sentence component. Users can easily recognize the similarities and differences between the subject, verb and object of a FSCL sentence and natural language. By seeing the visual information of FSCL descriptive sentences with the assistance of explanatory text, users will not only know how "word groups" are linked together to create a FSCL sentence, but also know how words from different categories can be combined into "word groups".

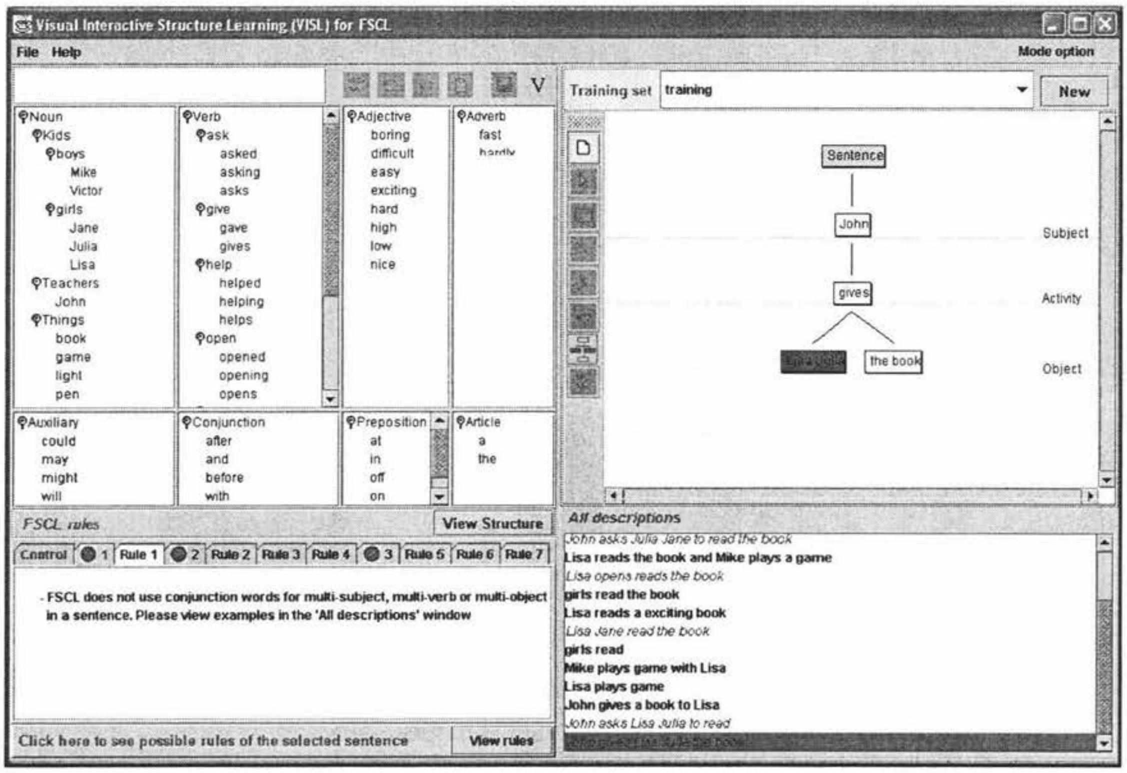
Figures 4.3 and 4.4 present examples of a combination of explanatory texts and FSCL tree structures for a FSCL structure rule: FSCL does not use connection words in a multi-subject, multi-verb or multi-object component of a sentence. In the sentence "Lisa Mike read the book" and "John gives Lisa Julia the book", the multi-subject "Lisa Mike" and the multi-object "Lisa Julia" are highlighted, respectively. From these explanations and examples, users will understand how the subject or activity in a FSCL can be created.



- FSCL does not use conjunction words for multi-subject, multi-verb or multi-object in a sentence. Please view examples in the 'All descriptions' window

John helps Lisa to read book
Lisa reads book
Lisa Mike read the book
Lisa Jane Mike read a book
John asks Julia Jane to read the book
Lisa reads the book and Mike plays a game
Lisa opens reads the book
girls read the book
Lisa reads a exciting book
Lisa Jane read the book
girls read
Mike plays game with Lisa

Figure 4.3 Structure rule and an illustrative diagram for a multi-subject sentence


Visual Interactive Structure Learning (VISL) for FSCL
File Help
Mode option
Training set training
New
Noun Kids boys Mike Victor girls Jane Julia Lisa Teachers John Things book game light pen
Verb ask asked asking asks give gave gives help helped helping helps open opened opening opens
Adjective boring difficult easy exciting hard high low nice
Adverb fast hardly
Auxiliary could may might will
Conjunction after and before with
Preposition at in off on
Article a the
Sentence
John
gives
the book
Subject
Activity
Object
FSCL rules
View Structure
Control 1 Rule 1 2 Rule 2 Rule 3 Rule 4 3 Rule 5 Rule 6 Rule 7
Click here to see possible rules of the selected sentence
View rules
All descriptions

- FSCL does not use conjunction words for multi-subject, multi-verb or multi-object in a sentence. Please view examples in the 'All descriptions' window

John asks Julia Jane to read the book
Lisa reads the book and Mike plays a game
Lisa opens reads the book
girls read the book
Lisa reads a exciting book
Lisa Jane read the book
girls read
Mike plays game with Lisa
Lisa plays game
John gives a book to Lisa
John asks Lisa Julia to read

Figure 4.4 Structure rule and an illustrative diagram for a multi-object sentence

❖ *Processes in the VST mode to display a language structure tree*

Figure 4.5 presents the processes in the VST mode. A predefined database of descriptions is set up in the repository as examples. These descriptions are carefully selected to cover all possible language structure rules. The descriptions are presented in a list. Users can select any structure rule of the language to view its details. Descriptions that follow the selected rule will be displayed in a different format to attract the user's attention. Users then can choose any description in this list to see its structure tree. When a description is selected, it is parsed resulting in a structure tree that is presented in the nearby working space. The nodes presenting the selected rule will be highlighted. There are some issues relating to the tree presentation such as how to draw balanced trees in a working space or how to determine which of the rules apply to a description. Those issues are discussed in detail in the next chapter.

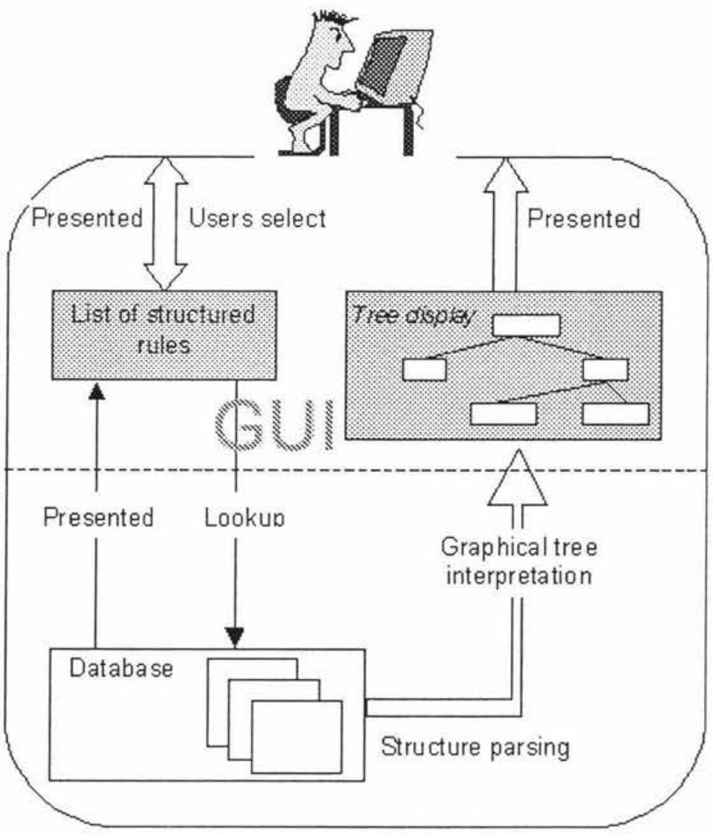


Figure 4.5 Processes of displaying structure trees in VST mode

4.2.2 Structure Tree Manipulation mode

In the previous mode, the users have studied structure rules and illustrative examples of constructed languages. The understanding users have gained by studying in the VST mode will be used and taken further in the Structure Tree Manipulation (STM) mode as they are allowed to build and manipulate constructed language structure trees from available individual words. In this section, the concept of STM is introduced, an example is provided and the processes to generate structure trees are also presented.

❖ *Concept of STM*

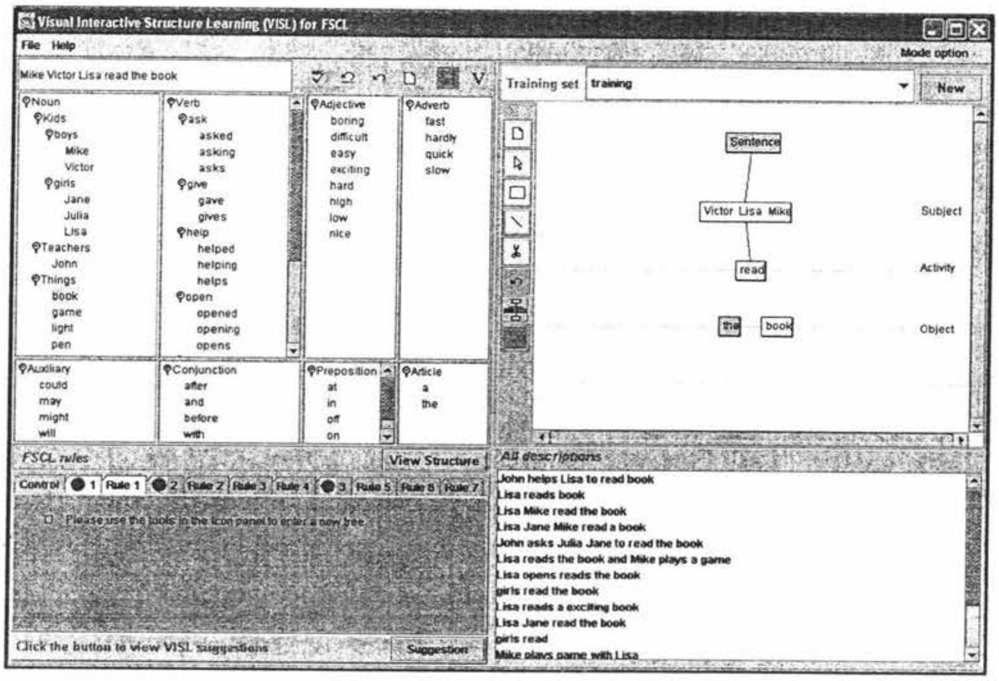
The mode stimulates users to learn constructed languages through interactions on constructed languages' structure diagrams in an enjoyable learning environment. This active learning approach helps users to strengthen the understanding gained from the previous mode. In this mode, users are allowed to create constructed descriptions by both graphical and textual input. The STM mode then checks the correctness of these inputs. In detail, the STM mode can help users to learn language structures by:

- *Identify users' misconceptions by allowing them to check language structures of descriptions that they have formulated:* As users can create their own constructed descriptions and check the validation of these descriptions following structure rules of constructed languages, they can identify any misunderstandings they might have developed from the previous mode. They also can apply the advantages of trial-and-error feedback in language learning as "More by trial and error than by the adequacy of definitions and descriptions, we gradually gained some facility in analysing language" (Eskew, 2001, p1). STM locates errors and points out violated rules. STM also offers solutions if the users are not able to correct these errors. By providing instantaneous and extensive feedback and making users understand clearly the errors, the application helps users to master the language easily.
- *Reinforce users' understanding by allowing them to create their own descriptions:* The STM mode provides a working space for users to do exercises on building structure trees from a set of vocabulary/elements of a description. Users can input textual descriptions and see the resulting structure trees. If users do not like the textual input method, they can build descriptions by the graphical approach. As users have to create descriptions by themselves, they have to concentrate their thoughts and have to be patient to create descriptions correctly. Therefore, users can reinforce the understanding gained from the previous mode. Users need to ponder over the words, examine the language structures and then enter correct word groups so that they imbibe little by little the language structure, since "listen and forget, see and remember, do and understand" (Hyerle, 1996, p13). The exercises also help users to convert what they have learnt from the previous mode into long-term memory. Furthermore, when users are motivated to use STM to build structure trees, they may enjoy the intellectual challenge of active visual construction of ideas.
- *Explore possible constructed language structures by creating and modifying their tree structures:* This mode not only allows users to create structure trees, but also permits users to manipulate those structure trees. When users want to

drive the discovery process and want to interact with their own graphical trees, they can modify these tree structures in different ways such as by adding, deleting and moving nodes or changing contents of any node in these graphical trees. Through these exercises, users will be able to explore variations of language structures. By using the drawing space, users can easily experiment to create new structure trees by simply moving and clicking the mouse then checking the validation of those structure trees. By doing so, the users will discover many possible language structures. They can also invent many new descriptions that go beyond the available descriptions stored in the database.

> In the case of FSCL, users can check the structure of the each node in the tree before checking the whole draft sentence. The correctness of a FSCL word group is checked based on two conditions: the correct words inside each node (i.e. words from different categories) and their proper function (i.e. subject, activity or object) in the structure tree. Users can decide on the function of a word group by placing it on the respective level in the tree. Following FSCL structures, a word group can be correct for one function but not for others. For example, the word group "to Lisa" is syntactical correct when it is an object but is not correct when it is a subject. After users check the correctness of the word groups in a tree, they continue to build the tree by adding links between the word groups and then check the tree as a whole.

The example in Figure 4.6 explains how the concept in this mode is applied to teach FSCL. In order to create an FSCL sentence that is equivalent to the simple natural sentence "Lisa, Mike and Jane read a book", users need to employ knowledge learnt from the previous mode. Users must firstly remember that they have to create a node of "Lisa Mike Jane" without the word "and" for the subject. They must then create a "read" node for the activity and a node of "a book" for the object. Users then link these nodes into a tree to create the FSCL structure tree. When users have finished building the tree, they can check its structure. If this sentence is correct following FSCL structures, the structure tree of this sentence is displayed. At this stage, the tree has the structure of "subject-activity-object" presenting the nodes for "Lisa Mike Jane", "read" and "a book". Users may want to delete the node "a book" then check the syntax for the tree again. Since this sentence is correct following FSCL structures, users can understand or be reminded that FSCL allows the structure of "subject-activity".



Mike Victor Lisa read the book

Please use the tools in the icon panel to enter a new tree

Click the button to view VISL suggestions

John helps Lisa to read book
Lisa reads book
Lisa Mike read the book
Lisa Jane Mike read a book
John asks Julia Jane to read the book
Lisa reads the book and Mike plays a game
Lisa opens reads the book
girls read the book
Lisa reads a exciting book
Lisa Jane read the book
girls read
Mike plays game with Lisa

Figure 4.6 STM during building a FSCL structure tree

When a user constructs the subject, "Lisa Mike Jane", he may enter a "natural language like" subject as "Lisa Mike and Jane". An error message is displayed to explain that the word "and" is not allowed in a multi-subject and the rule that the user has violated is highlighted. STM will provide a solution of how to fix this error if required by the user.

❖ *Processes for building a structure tree in STM*

Figure 4.7 presents the process for building a structure tree. This mode provides a working space in which users can draw trees with nodes of words and links. Trees drawn yet not checked for grammatical correctness are called "*draft trees*". Users can manipulate draft trees in the working space. When a draft tree is completed, users initiate syntax checking to test for the correctness of the draft tree. Users will be informed if the structure of the draft tree is correct. If the structure is not correct, an error interpretation routine is executed. This routine finds the exact location of the errors and highlights the errors in the draft tree. A node, which has structure errors, will be displayed in a different format from that of correct nodes in order to attract the user's attention. If users do not know how to fix the error, they can view a solution provided by STM. This will help users to recognise the structural errors and correct them. The processes are equivalent when users create descriptions by textual input.

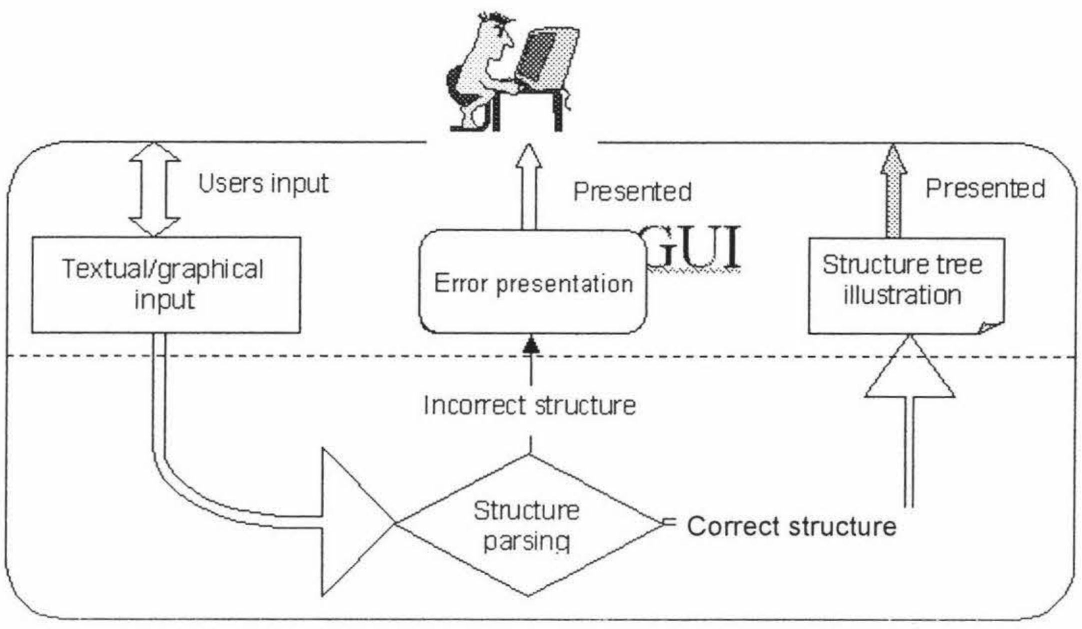


Figure 4.7 Processes of building a structure tree in the STM mode

4.2.3 Visualized Query Matching mode

Different constructed languages may have different query engines. However, these query engines can be explained and illustrated easily through diagrams. Therefore, the Visualized Query Matching mode (VQM) will assist users in understanding constructed languages' query mechanisms through the presentation of structure diagrams.

There are two subsections in this section. The first presents the concept of VQM. The second section describes the processes inside VQM to present matching locations presented in a description tree.

❖ *Concept of VQM*

To teach the query engines of constructed languages, VQM will explain how to create a query and how a description is a result of a query. Diagrams are used to assist the presentation of the issues.

- *Explaining how to create queries with illustrative diagrams:* Creating queries such as keyword or free text queries in natural language is simple. However, users usually do not know how to create queries in constructed languages. Therefore, the first task in this mode is to explain how to create those queries. Explanations, example queries and their structure diagrams are used to help users to understand how to create a correct query. In the case of FSCL, it is simple as a FSCL query is also a FSCL sentence.
- *Teaching users how the query engine works by explanation and illustrative examples:* Again, the approach of using explanatory notations and diagram illustrations is applied to assist users to understand the constructed language's rules that explain how descriptions satisfy the requirements of a query. Each requirement is presented by some concrete illustrative examples that assist users in understanding the explanatory text.

  For teaching the FSCL query engine, the mode explains how two nodes of word groups are equivalent and how two structure trees have matching patterns. By studying nodes that present two matching word groups and viewing the explanation, users are able to understand how this "word group" matches with that "word group" but not the others. For example, a node of the subject word group "Mike Jane Lisa" of a descriptive sentence "Mike Jane Lisa read the book" provides a match with the subject word group "Mike Jane" of a query sentence "Mike Jane read the book", but not vice versa.
- *Automatically presenting matching pattern:* VQM stores pre-defined queries and descriptions and allows users to study the matches between queries and descriptions. Users can see the visual structures of those queries and the location of the matching positions in the retrieved descriptions. By studying the match locations between the query and retrieved descriptions, users can understand why a description satisfies the query requirements. VQM lets the users select any query and adjusts the display of descriptions that match this selected query.

For teaching the specific FSCL query engine, this mode will explain why descriptive sentences do not match queries due to the limitations of FSCL structures. This is because users may have entered descriptive sentences that are correct following FSCL structures but, as discussed in the previous section, might not produce the matches expected by a FSCL novice:

Section 3.2.4 outlined that the descriptive sentence "John plays chess with Lisa" and the query "Lisa plays chess" will not match each other, as they have different sentence structures.

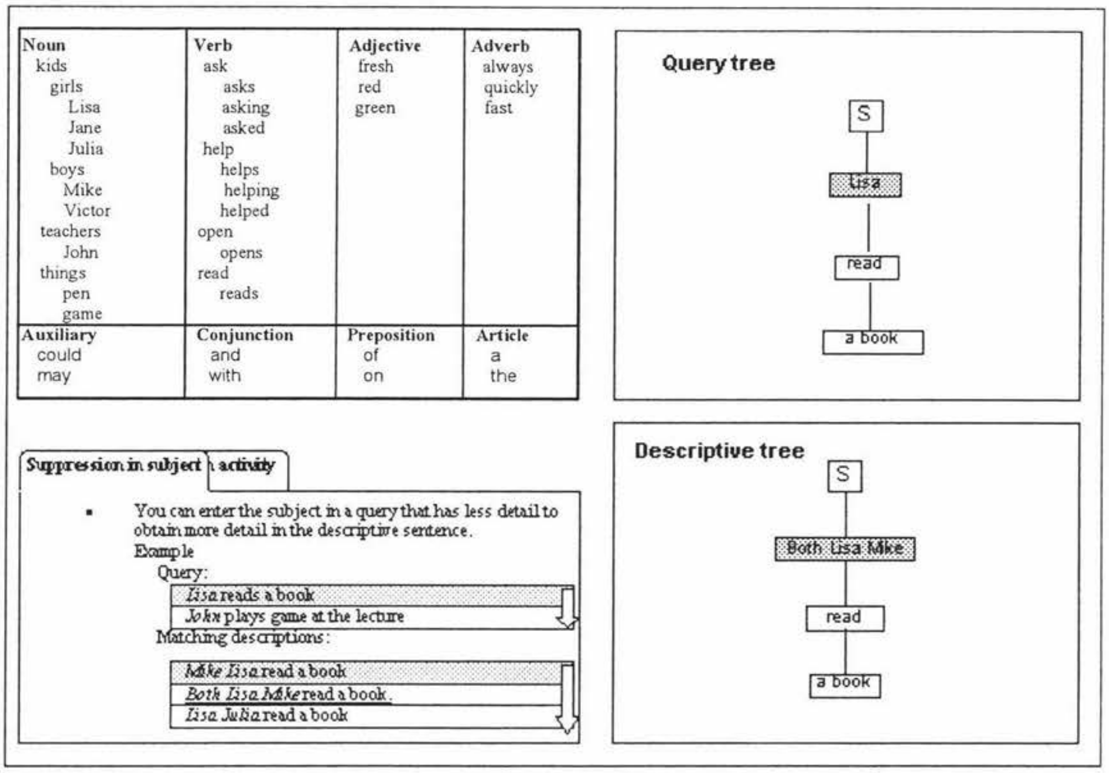
Section 3.2.5.3 proved that users would easily understand why a description would match a query by viewing their structure diagrams and the matching pattern. From that point of view, the mode used for teaching FSCL will help users to avoid the structures that can cause difficulty in the retrieval process by:

Providing a list of FSCL structures that are difficult for the retrieval process;

Providing sentence examples for both queries and descriptions;

Illustrating these examples by presenting structure trees of these sentences (queries and descriptions).

Figure 4.8 presents an example of the suppression of details of the subject in a query (see the issue of "suppression of details in a query" in Section 3.2.3.1) for a "matching lesson". In the "Example queries" block, a list of examples presenting that lesson of the query engine is presented. Users can choose any query in the list. In the "matching descriptions", a list of descriptions matching the selected query is presented. Users can choose any description in the list to view the matching pattern. In this example, a user chooses the query "Lisa reads a book" and selects the matching descriptive sentence "Both Lisa Mike read a book". It can be seen that the subject "Lisa" in the query tree matches with the subject "Both Lisa Mike" in the descriptive tree. To attract users' attention, VISL presents these matching nodes in a different colour compared to other nodes in the trees.



Noun	Verb	Adjective	Adverb
kids	ask	fresh	always
girls	asks	red	quickly
Lisa	asking	green	fast
Jane	asked		
Julia	help		
boys	helps		
Mike	helping		
Victor	helped		
teachers	open		
John	opens		
things	read		
pen	reads		
game			
Auxiliary	**Conjunction**	**Preposition**	**Article**
could	and	of	a
may	with	on	the

Suppression in subject | activity

- You can enter the subject in a query that has less detail to obtain more detail in the descriptive sentence.

Example

Query:

Lisa reads a book

John plays game at the lecture

Matching descriptions:

Mike Lisa read a book

Both Lisa Mike read a book.

Lisa Julia read a book

Figure 4.8 An example of matching subjects for query and descriptive trees

❖ *Process in presenting a matching pattern automatically:*

Figure 4.9 presents the process of presenting a matching pattern. A list of pre-defined queries stored in the database is presented to users. Once users select a query from the list, the VISL matching engine is called and descriptions that satisfy conditions of the selected query are displayed. When users choose a description in the list, an interpretation of the matching location is called to display the matching location between the query and the description. This interpretation includes an algorithm to compare the query and description. Details of an algorithm used for a specific language, FSCL are discussed in the next chapter.

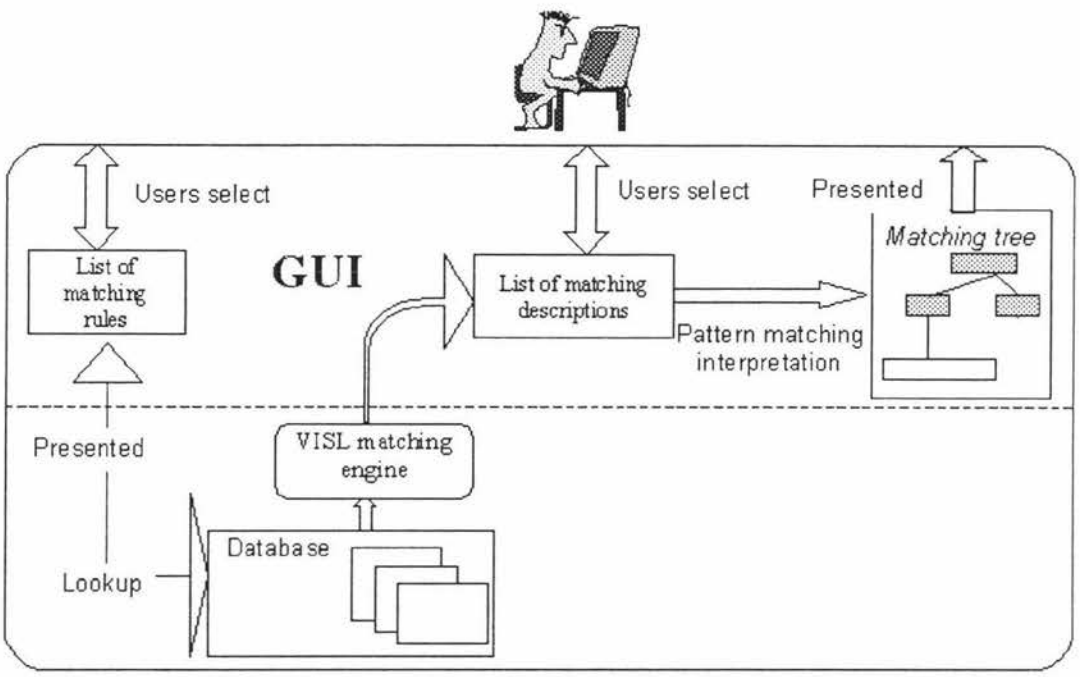


Figure 4.10 Processes of automatically presenting pattern matching in VQM mode

4.2.4 Visualized Building Query mode (VBQ)

This active learning mode assists users in strengthening what they have learnt from the previous mode. In this mode, users can create their own queries by drawing query trees or entering queries in textual format. Uses will use these queries to search for pre-defined descriptions or descriptions created by the STM mode that are stored in the repository and then study the matching results.

This section first lays out the concept of the VBQ mode and then demonstrates this concept by an example. At the end, the processes inside this mode are described.

❖ *Concept of VBQ*

After learning the matching conditions between descriptions and queries and viewing examples, users now can practice creating queries by themselves. By allowing users to build queries, search matching descriptions and view matching locations between queries and descriptions, the mode will help users to:

- *Clarify and reinforce users' understanding of how the query engine works:* The mode helps users to learn better, remember longer what they have learnt. In addition, they can gain better understanding of how to create sensible queries to get correct returns. As different constructed languages have different query engines, this section will explain how the VBQ mode can assist users in learning the FSCL query engine.
    - *Assist users in creating vocabulary with appropriate hierarchies by allowing them to change queries.* Creating a suitable vocabulary with appropriate hierarchies will help users to narrow or broaden matching descriptions. This is useful for both detailed and aggregate analysis. For example, in a research project focussing on children conducted with FSCL, researchers may want to identify girls or boys who want to read books or play games during the lunch break. They may also want to know whether a child named "Lisa" wants to read a book or play a game. Users can create queries that retrieve only the descriptive sentences using general word "girls" (aggregate) or descriptive sentences that refer to a specific girl name "Lisa" (detailed). In FSCL, the deeper a word is in a hierarchy, the more specific is the information it describes. The VBQ mode will assist users in creating queries to get narrow or broad results by allowing them to change queries and view matching results. This practice helps users to recognise the importance of making use of the hierarchy relationships in description and querying. Figure 4.10 presents the strategy applied by VISL to help users to understand that feature of FSCL.
    - *Encourage users to create a better vocabulary database and generate descriptive sentences that are easy to locate:* People may use several different words referring to the same thing or same activity. To facilitate searching, it is necessary to use a single word for those things or activities. Since FSCL can be used by a group of users, words that need to be added into the vocabulary should have a meaning everyone in the group can agree with. A well-designed vocabulary can help to connect the ideas of vocabulary creators and other users who use that vocabulary and avoid inputting ambiguous vocabulary. This helps users to choose the right interpretation of any description. The mode allows users to repeat changing each word or word groups in a query and see the results with visual matching patterns. This encourages users to select words to get exactly the expected results.

Figure 4.10 presents an example for the teaching concepts applied to help users to understand better the issues around word hierarchies in FSCL. When users move the mouse over a node of a query tree, a pop-up window will appear to present a list of words that are available for selecting. Users can select any word in that list to modify the content (words) of the node. With this feature, users can move up to a higher level of hierarchy of the current word of the query to get a general result or move down to a lower level of hierarchy of the current word to get more specific answers. The query results will change following the modifications in the query.

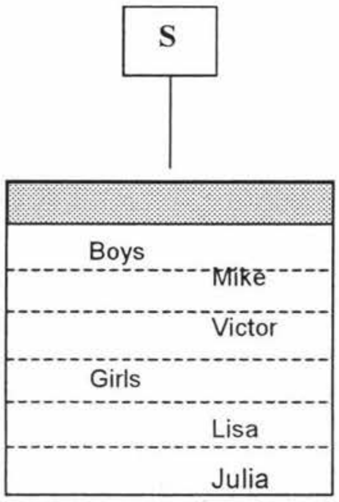


Figure 4.10 Users can choose any word in a hierarchy on a query tree

❖ *Processes in VBQ mode:*

Figure 4.11 presents the processes in the VBQ mode. Users can create textual input or build query trees from available words. These queries then will be checked for structural correctness. An error message will appear if users create a query with incorrect structure. If the query is correct, a list of descriptions that satisfy the query requirements will be presented. Users can select any of those descriptions to view the matching locations between the query and the selected description.

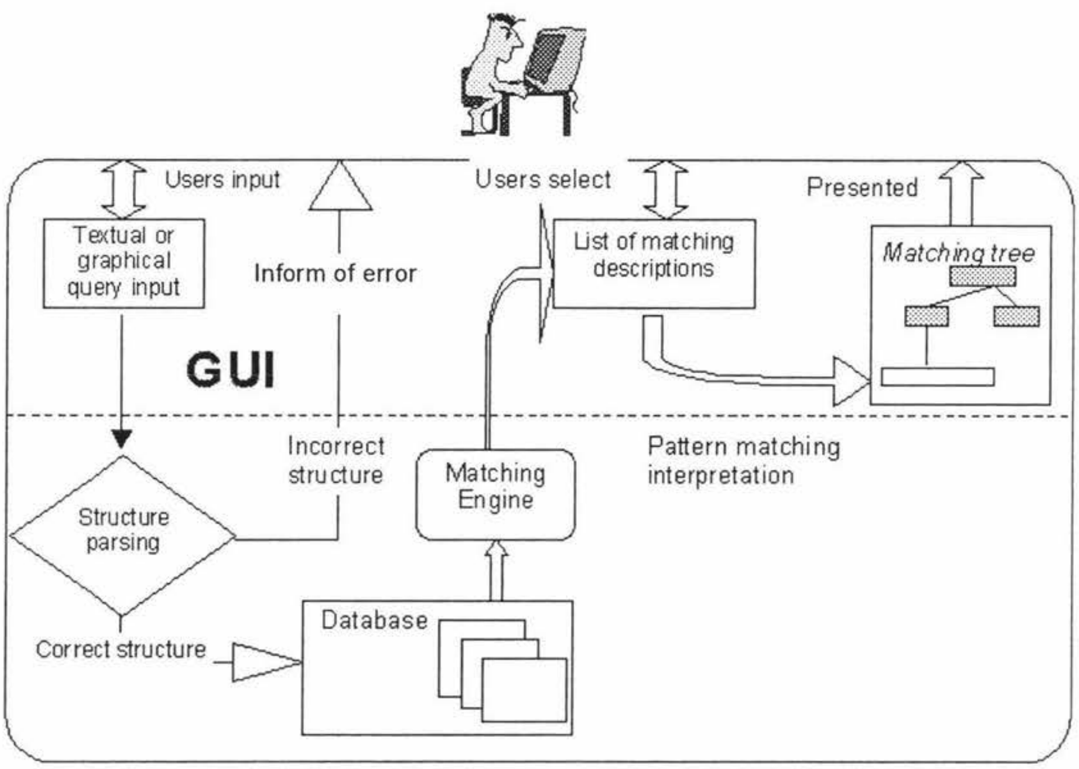


Figure 4.11 Processes in VBQ mode

4.3 Adaptation of VISL concepts to teach other constructed languages

The teaching concepts above use FSCL as an example for illustration. The concepts are transferable to teach other languages that have constructed structures. For "natural language like" languages such as the CABER and Observer languages, these concepts can be applied in a straightforward manner.

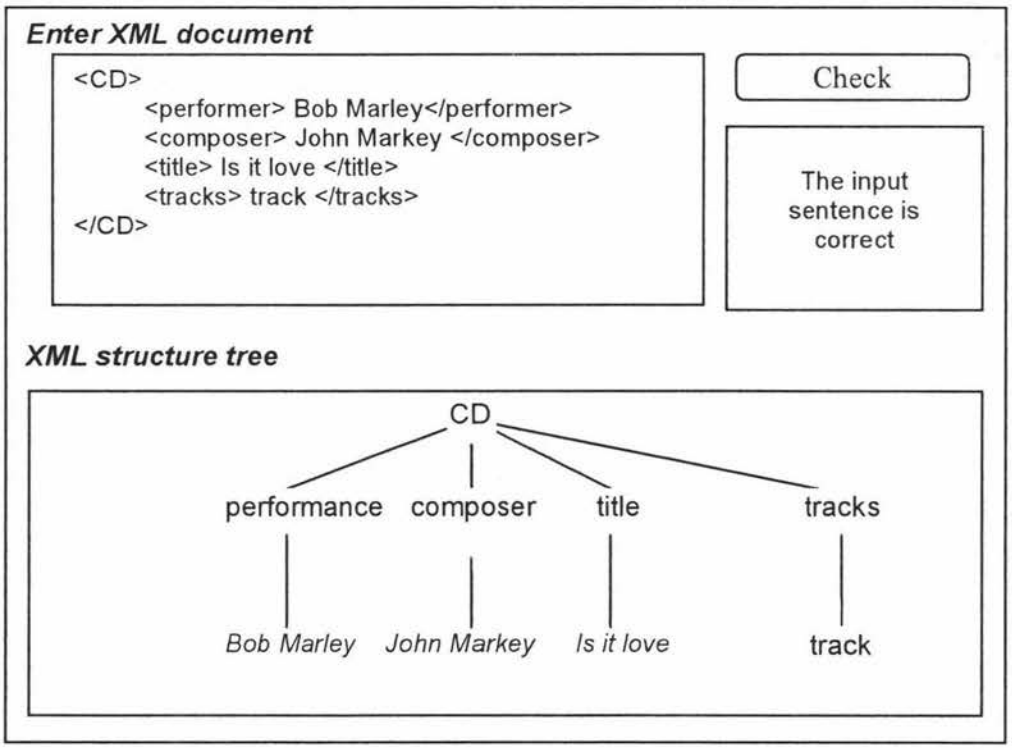
The concepts above also can be used to teach structured languages such as XML. For example, to teach XML *structure* and its associated *query engine*, XPath, the approach for a teaching tool can be used:

1) Passively present the structures of XML documents and the XML query engine by combining textual explanations and illustrative diagrams as following:
    a. Introduce the rules of XML structure and illustrate those rules with tree presentations.
    b. Explain how to generate correct queries compliant with XPath structures and show how a part of a XML document is the result of a

query by using rule explanations and illustrations of structure diagrams.

2) Enhance users' understanding by providing an active learning environment that helps users to mastermind the language and the query mechanism:
    a. Allow users to create XML documents by textual input or graphical input and the application will check structural correctness of these inputs. The tool also allows users to interact with those diagrams so they can discover possible structures of XML documents.
    b. Allow users to create queries and review matching results. Users can view locations of a XML document that is a result of the input queries and understand clearly how the query engine works.

Figure 4.12 shows an exercise illustrating concept 2.a for learning XML structures. Before doing this exercise, users will have familiarised themselves with the underlying XML DTD or Schema and will be able to test their understanding of this. The exercise requires users to input a XML document following the pre-defined DTD. As the input document is correct, the structure tree of this XML document is displayed with a congratulation message.



```
<CD>
        <performer> Bob Marley</performer>
        <composer> John Markey </composer>
        <title> Is it love </title>
        <tracks> track </tracks>
</CD>
```

Figure 4.12 Presenting XML structure by using diagram

Figure 4.13 presents an example illustrating the concept 1.b for learning a XML query mechanism, XPath. Query examples are pre-defined and presented in the top left hand side. The structure tree of the XML catalogue document is presented in the right hand side of the figure. In the figure, a user selects the first XPath query "catalogue/CD/tracks/" in the list and presses "Enter". The meaning of this query is presented in the lower left hand side. The sections in the XML structure tree that are result of the query is displayed and presented in the right hand side. The matching location is shown in a different format compared to other nodes in the structure tree.

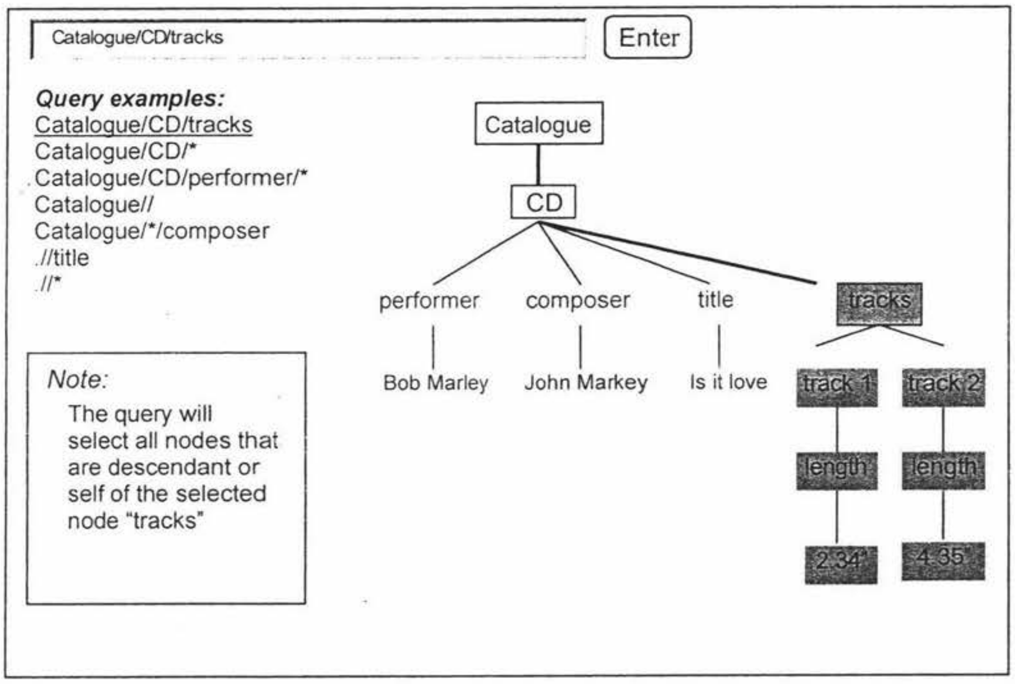


Figure 4.13 An example using diagrams to view query results in a XML document

4.4 Summary

This chapter discussed in detail the concepts for teaching the constructed language through four modes. The first two modes are used for teaching the language's structures. The other two modes are used for teaching the language's query mechanisms. For learning language structures, users will first go through the passive learning mode to understand the language structures. They then go through the active learning mode that consolidates what they have leant from the previous mode by using an interactive graphical approach. For learning the language's query mechanisms, users will understand why a description is satisfied with requirements of a query through the passive learning mode. Users then practice building queries and viewing result descriptions through the active learning mode. As different constructed languages have different structures, the research explains how the developed concepts were applied for a specific constructed language, FSCL. Learning through those modes, users will develop full understanding of how to construct correct descriptions and create effective queries. All of these benefits are obtained from using visual presentations extensively as they facilitate "constructing and remembering, communicating and negotiating meaning, and assessing and reforming the shifting terrain of interrelated knowledge" (Hyerle, 1996, p34).

Designed with these four modes, VISL encourages users to learn and mastermind both constructed language structures and their query engines. In the next chapter, the details of how those concepts are implemented for the constructed language, FSCL, will be discussed.

Chapter 5

Key issues toward an implementation of VISL

In the previous chapter, the concepts of teaching constructed languages with four modes have been discussed. The example of a constructed language, FSCL, has been used to illustrate the teaching concepts. This chapter will explain some key issues that need to be considered before developing an application using these concepts to teach FSCL. These issues are separated into two categories: application development issues and technical issues.

1. Application development issues: The issues regarding to the development of the application including:

- How VISL is developed as an efficient CALL application (Section 5.1.1). The chapter discusses issues regarding to how VISL is designed.
- Should VISL be separated from or combined with the current PAC, the computer application that includes FSCL. The chapter explains why the approach of a combination between PAC and VISL was chosen (Section 5.1.2).
- How to create an effective user-interface (UI). The project reviews how a graphical user interface (GUI) for training systems is designed in order to create a suitable learning environment for users to interact with (Section 5.1.3).

2. Technical issues: The issues regarding to the development of VISL's functions including:

- How the FSCL structure rules and matching rules are organised in lessons (Section 5.2.1) – this issue is required for the VST mode and the VQM mode;
- How to draw a balanced tree to display FSCL structure trees – required for all modes in VISL. The chapter reviews some possible algorithms to draw balanced trees and then selects a suitable algorithm to display FSCL semantic trees (Section 5.2.2).
- How VISL can deal with errors caused by novice users when they create their own FSCL sentences (Section 5.2.3) – required for the STM mode. This

section discusses the way applied in VISL to present errors to users. It also explains how VISL can help users to fix the errors.

- How VISL identifies the rules of FSCL structures included in FSCL sentences (Section 5.2.4) - required for the VST mode and the STM mode. This section also presents an approach used in VISL to help users to modify the content of a node while building structure trees.
- How descriptive trees match with a query tree – required for the VQM mode and the VBQ mode. This section presents an algorithm used for searching descriptive sentences that match a query and display matching patterns (Section 5.2.5).

5.1 VISL's development issues

This section discusses general issues regarding the development of the VISL application including its features, its relationship with the computer application that implements FSCL, database, and user interface design.

5.1.1 VISL, a CALL application

This part presents obstructions that developers may face when they develop CALL applications. It then presents solutions to overcome these problems.

5.1.1.1 Issues of concern in developing CALL applications

Constructed languages are commonly integrated with computer applications for the purpose of describing and retrieving information. Therefore, learning constructed languages is only the first step of many topics that users need to learn before using these computer applications. For example, in PAC, the application used for analysing behaviour, users not only need to know how to generate correct FSCL sentences or search for specific information but also need to know how to segment video files and how to interpret or analyse the search results obtained. For these reasons, a teaching tool for constructed languages should be simple to learn and interesting enough so that users can learn the language quickly before they start to use the application.

In addition, Levy (1997) gives some constrains that developers may face when developing a CALL application:

- People will always find a way to use things that the designers did not anticipate.
- People will always want or need features that they did not know they wanted or needed until they begin to see and use the software.
- Preferences and priorities among users and between developers and users will certainly differ.
- Important aspects of the way people use the system depend on certain software features and structures.

5.1.1.2 Design of VISL

Taking constrains and requirements above into account, the project develops a CALL application called VISL to teach FSCL structures and its query engine. Built based on the learning concepts developed in Chapter 4, the application provides an interactive learning environment with specific features: flexibility, interactivity, step-by-step learning, tutoring and student centeredness.

- *Flexibility:* In many language-teaching applications, users can only learn through small sets of predefined examples. In VISL, besides storing many predefined examples, VISL allows users themselves to input sentences in the form of both textual input and graphical tree input. Furthermore, users can use the pre-defined vocabulary or freely build their own vocabulary.
- *Interactivity:* VISL allows users to interact with structure diagrams by practicing to construct structure trees or build graphical query trees. Users are able to manipulate structure diagrams, such as delete, add or modify any available trees, to explore possible structures of the learning languages.
- *Step-by-step learning:* VISL is implemented to support users to learn FSCL step by step. VISL presents only one structure rule of the language at a time by displaying each rule with some concrete illustrative diagrams as examples. This will help users to reduce memory overload and will encourage users' long-term memory.
- *Tutoring:* FSCL structure rules and FSCL matching conditions are divided into categories and "lessons". Users can select any lesson to learn. VISL provides effective feedback that assists users in knowing what types of errors they have made and how to fix these errors.

- *Student centeredness:* As a constructed language may be used for different groups of users and each user has individual needs, VISL provides a platform that engages independent learning for many types of users. Users can choose any mode to learn and create any domain to practice with. Since users can add their own vocabularies or create their own descriptive sentences that are suitable for their own purposes, the application provides a self-motivated learning environment that can encourage slow learners.

5.1.2 VISL, a combined module with PAC

Two approaches were considered when the project was started:

- VISL is separated from PAC to be an independent teaching application. Any change inside PAC does not affect the VISL application.
- VISL is integrated with PAC as a component. VISL can use some components already available in PAC.

Building a training application that is separated from PAC as a stand-alone application can be a simple approach as it is easy to start with. Developers would have much more freedom to develop components for the VISL application. For example, they can develop separate syntax checking to verify FSCL structures (for the STM mode) or create their own database to store pre-defined FSCL sentences (for the VST mode). Developers can also use any programming language to develop the application. However, developing a separate checking mechanism might be a complicated process and might require considerable time.

On the other hand, if VISL is integrated with PAC, the developers must fully understand the PAC structure, its components as well as its code. Developers also have to use the programming language that has been used for the development of PAC. However, there are many advantages if there is a combination between VISL and PAC. The VISL application can inherit some functions already existing in PAC. For example, it can use PAC's current parser and can use the databases already set up by PAC. Moreover, as PAC is developed with an Object-Oriented principle and the popular Java programming language, an add-in module to PAC should be straightforward. In addition, from the user's point of view, the user can easily access the application for training when VISL is integrated with PAC. Therefore, the approach of integration between VISL and PAC was chosen for the current project. Using this approach, the VISL application is considered as an additional module inside the PAC system (Figure 5.1). This would not necessary be the case for other applications used for teaching other constructed languages.

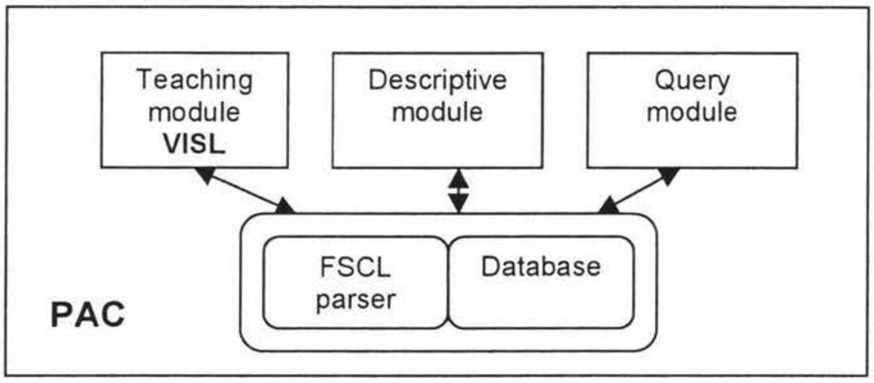


Figure 5.1 VISL, a module integrated into PAC

5.1.3 VISL's User Interface design

As it uses diagrams extensively to teach language structure, the VISL application can be seen as a visual tool to teach users FSCL structures. Therefore, good user interface (UI) design is one of the vital factors for the success of the application in enhancing users' understanding. Due to the importance of the user interface, the project has investigated how to develop an effective UI for the VISL application.

This section is divided into two parts. The first part presents some principles on developing an effective UI for teaching applications and the second part discusses how those principles can be applied for the UI of VISL.

5.1.3.1 Principles in designing UI for learning systems

The user interface design is commonly a central issue for the usability of a software product. "Usability is measured by the extent to which the intended goals of use of the overall system are achieved (effectiveness); the resources that have to be expended to achieve the intended goals (efficiency); and the extent to which the user finds the overall system acceptable (satisfaction)" (Oppermann 2002, p2). Poor usage of design principles can increase instructional time and reduce completion rate and persistence. Szabo and Kanuka (1998) found that people who used the lessons of a computer-based tutorial with good design principles, completed the lessons in less time (21%) and had a higher completion rate (74% vs. 45%) than those who used the lessons with poor design principles, even there was no difference in achievement scores between the two groups. Oppermann (2002) pointed out that the user interface of training applications should be self-descriptive and suitable for learning to a much higher degree than for many other applications because the system is only used by a user for a limited time until he or she has learnt the content of the system. He also suggested that the designer should focus on the appeal of the diagrams (e.g. is the diagram really attractive?). He explained that, in the modern days, many people are overloaded with information. Therefore, if diagrams do not look nice, people may not want to use them. In addition, Oppermann recommended that graphics must immediately and automatically present the most important points. He also suggested that diagrams should organise information into clear visual hierarchies. A diagram should be designed with clear recognition (size, colour, brightness, position etc.). The following section explains how those principles are applied in VISL.

5.1.3.2 UI principles applied in VISL

As discussed in Section 4.3, VISL allows users to do actions like the following:

- View FSCL structure rules, matching rules and graphical trees of example sentences;
- Freely select words in a vocabulary window to create new graphical or textual descriptive sentences or query sentences;
- Interact with diagram tree structures: add, delete or modify the content of structure trees.

With those functions, VISL's UI should have a vocabulary window in which users can freely select words. It should also present example sentences so that users can easily choose to view their structures. In addition, VISL's user interface needs to have a working space for users to view graphical trees or to build new structure trees. In addition, the UI needs the graphical tools necessary for building structure trees. From these requirements, the UI is designed with the following features:

Firstly, since users use VISL before they use PAC, the consistency between the VISL interface and PAC's GUI is important. This helps the users to reduce the time required to learn the functions already existing in PAC. Therefore, some parts of the VISL interface are designed similar to those of PAC's interface such as the vocabulary window or the icon panel for textual input of FSCL sentences. Like the UI of PAC, VISL uses many icons to speed up searching, for immediate recognition, for better recall, and to save window space in the display area.

Secondly, VISL's GUI divides the screen into a limited number of clearly recognisable rectangular areas to group information or functions following suggestions by Koning et al., (2002):

- The vocabulary window and the working space window are used for all modes and they are frequently used. Therefore, they are located at the top of the user interface as "the higher the more important" (Koning et al., 2002, p.1).
- The users can select any word in the vocabulary window to draw new FSCL trees in the working space window. Therefore, the vocabulary window is located on the left and the working space window used to draw FSCL structures is displayed on the right. This design follows the guideline "All inputs are on the left side and all outputs are on the right side" (Koning et al., 2002, p1).
- The list of the structure rules or matching rules is placed on the left. When users click on any rule in the list, descriptive sentences following that rule and their structure trees will be shown on the right side. This design is implemented following the instruction "The object of the right is active after the object on the left" (Koning et al., 2002, p.1).

The diagram in Figure 5.2 illustrates the design of VISL's GUI based on the design principles presented above.

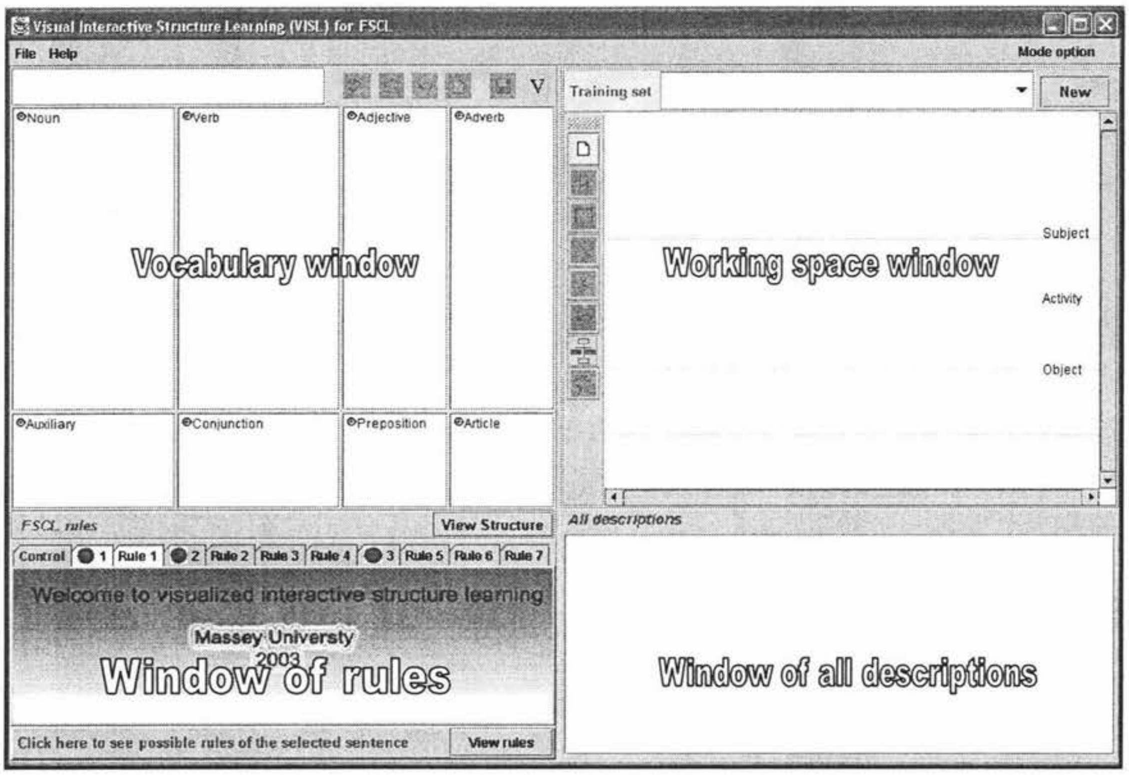


Figure 5.2: The design of VISL's user interface

5.2 Technical issues

This section present technical issues relating to the development of functions in the application following the teaching concepts developed in Chapter 4.

5.2.1 Presentation of FSCL structure and query matching conditions

As discussed in the previous chapter, in the VST mode and the mode VQM, users will learn constructed language structures and their query engines though explanations and illustrative examples. This section presents how the structure of FSCL is organised into "lessons" that will be displayed to users. It also presents how the matching conditions of descriptive and query sentences are arranged.

5.2.1.1 Lesson of FSCL structures

FSCL has both similar and different properties compared with natural language (Section 3.2). This section presents some lessons that describe different structures between FSCL and natural language.

- FSCL does not use the pronoun category as natural language. Users can use nouns instead of pronouns.

Lisa sees game movies before *Lisa* goes to sleep.

John gives the book to Lisa then *John* asks the class to read.

- FSCL does not follow the morphology rule as natural language does. Therefore, a correct FSCL sentence may violate natural language structure.

  Lisa *read*.

  The teacher *ask* some students to stop the game

  Students do *many exercise* before students can play.

- FSCL does not use connection words at the beginning of a sentence. See example below:

  Wrong sentences:

  After finishing the lecture, Lisa will go to see movie.

  During the interval, Mike Lisa play chess.

  Correct sentences:

  Lisa will go to see movie after *finishing* the lecture.

  Mike Lisa play chess *during* the interval.

5.2.1.2 Lessons of the FSCL query engine

FSCL allows users to create two types of queries. The first one is keyword search and the second is sentence search. In sentence search, users can suppress details in a query or broaden or narrow search results. This section presents several examples to explain how VISL organises the sentence-searching engine for *suppression of detail* queries into "lessons".

- Users can enter the subject in a query that has less detail and obtain the descriptive sentences with more detail.

  Example

  Query:

  Lisa read a book

  Matching descriptions:

  Mike Lisa read a book.

  Both Lisa Mike read a book.

- You can create a query that has less detail in the object. The object of matching descriptive sentences may contain more details.

Example

Query:

Lisa reads *a grammar book.*

Matching descriptions:

Lisa reads *a German grammar book.*

Lisa reads *a good grammar book.*

Lisa reads *a grammar book.*

5.2.2 VISL's algorithm to display FSCL semantic tree

There are two types of graphical trees that are commonly seen: normal trees and binary trees. In a normal tree, a parent may have one or many children. Trees in this type are a natural way to present information of organizational charts, design spaces or directory structures. Binary trees are trees, in which, each parent may have only one or two children. The binary tree is a fundamental data structure used in computer science as it can be used for rapidly storing sorted data or rapidly retrieving stored data (Rosé, 2003). Drawing binary trees is simple because almost every node in a tree has two children and therefore, node positioning is simple. With a normal tree (or general tree), a parent may have one or many children so it is more difficult to position the children for producing visually pleasing tree pictures. Various algorithms have been proposed for this difficult problem of drawing general trees but implementations only exist in special purpose software or are designed for special environments (Bruggemann-Klein and Wood, 1996).

As discussed in Section 3.2.3, FSCL semantic trees can be seen as general trees. This section reviews some typical algorithms for the drawing of general trees and then selects a suitable algorithm to draw FSCL trees.

5.2.2.1 Typical algorithms to draw a general tree

Following HCI principles discussed in Section 5.1.3, to enhance users' understanding, trees presenting FSCL structures must be aesthetic, pretty and easy to read. Several rules have been introduced to define a well-shaped drawing of a tree. Luo (1993, p6) introduced some rules to draw aesthetic trees, including:

- Rule 1: Trees impose a distance on the nodes; no node should be closer to the root than any of its ancestors.

- Rule 2: Nodes at the same level of the tree should lie along a straight line and the straight lies corresponding to the levels should be parallel.
- Rule 3: The relative order of the nodes on any level should be the same as in the level order traversal of the tree.
- Rule 4: A parent should be centred over its children.
- Rule 5: A sub-tree of a given tree should be drawn by the same way regardless of where it occurs in the tree.

To draw a tree that satisfies the conditions above, the basic task is to assign a pair of coordinates (x and y) to each of nodes in a tree. It is easy to determine and change the value of y that corresponds to the level of the nodes in a tree. The more difficult task is to find out the value of x that corresponds to the order of the nodes in the tree. The following sections review some typical tree drawing algorithms to display general trees.

- *RT algorithm:* One of the first algorithms to draw general trees was the RT algorithm. This algorithm was introduced by Reingold and Tilford (1981) and is considered as a divide-and-conquer algorithm for determining the positions of nodes. This algorithm lays out the sub-trees of a node independently and then places them close together. It can be imagined that the sub-trees have been drawn on paper and cut along their contours. Combining the sub-trees can be done by a traversal of the parts of the contours that are among the sub-trees until the bottom of one contour is reached. When the sub-trees are put together, they will form a new contour of the combined sub trees if the sub-trees are of different height. This algorithm does not allow any level of sub-trees to move closer. They can only move apart from each other. Also, once a sub-tree is laid out, its shape is fixed. More details of this algorithm were presented by Reingold and Tilford (1981).
- *Moen's algorithm:* In Moen's algorithm, to determine the nodes' positions, the algorithm first traverses the tree to forms a contour for each node, which it stores in the node's contour field. If a node is a leaf, a function is used to calculate the contour directly. If a node is a branch, the function to form a contour includes three steps:

- Form a contour for each sub-tree;
- Place the children's contours as close as possible;
- Compute the offset between parent and children and complete the parent's contour.

After a contour for each sub-tree is formed, it is joined with other siblings of the same parent in the tree. After the contours of all siblings are formed, the algorithm computes the parent's position by putting the parent at the centre of its children's contours. More details on this algorithm can be found in Moen, (1990).

- *Walker's algorithm:* In Walker's algorithm (Walker, 1990), two tree traversals are used to produce the final x-value of a node. The first traversal assigns the initial x-value and modifier fields for each node and the second traversal computes the final x-value of each node by summing the node's initial x-value with the modifier fields of all of its ancestors. This algorithm produces evenly distributed, proportional spacing among sub-trees. Two traversals are discussed in detail as following:

The first tree traversal is a post–order traversal that positions the smallest sub-trees (the leaves) first and recursively processes from left to right to build up the position of larger and larger sub-trees. Sub-trees of a node are formed independently and placed as close as possible. When the tree moves from the leaves to the top, it combines smaller sub-trees and their roots to form a larger sub-tree. For a given node, its sub-trees are positioned one-by one and moving from left to right.

The second tree traversal is a pre-order traversal that determines the final x-value for each node. It starts at the top node and computes the sum of each node's x-value. As a drawing space can be used to display trees with different shapes, the second traversal sets the top node at a position that respects both the dimension of the drawing space and the shape of the drawn tree.

5.2.2.2 Drawing FSCL trees in VISL

Even both RT algorithm and Moen's algorithm satisfy the six rules of drawing well-shape trees above, trees drawn by these algorithm are not easily recognised (Bruggemann-Klein and Wood, 1996). This research selects Moen's algorithm as the trees drawn by this algorithm are aesthetically pleasing and have fairly evenly sized sub-trees. To draw FSCL structure trees, Moen's algorithm needs to have some adjustments. For example, Moen's algorithm requires that a child must have a parent. In contrast, FSCL structure trees may allow for a grand parent – grandchild relationships without a parent node in between.

This section discusses the modification of Moen's algorithm applied in FSCL and then illustrates the algorithm through an example.

The algorithm follows these steps:

- Set distances for each level in the tree (=y);
- Follow post-order traversal for a general tree, do:
    1. Calculate the width of the left most node of the tree;
    2. If this node has sibling, recursively call the step 1 for its sibling;
    3. Place the siblings as close as possible without overlap;
    4. Place the parent in the centre of its children. The distance between this parent to its children (following y axis) is = y.
- Redistribute all positions of all nodes depending on the diameter of the working space (e.g. the place to draw the tree) and location of the root.

The width of each node is the length of that node's words plus its bounds. After positions of all nodes are determined, the locations of the centre top and the centre bottom of each node are calculated to draw links between parent nodes and their children. Then the algorithm draws the tree from top to bottom with links.

The example below illustrates how this algorithm works. Figure 5.3 displays a tree that includes 9 nodes with different widths. Those nodes in the tree are marked from 1 to 9 following the post-order traversal in a general tree. The following steps are applied to draw the tree:

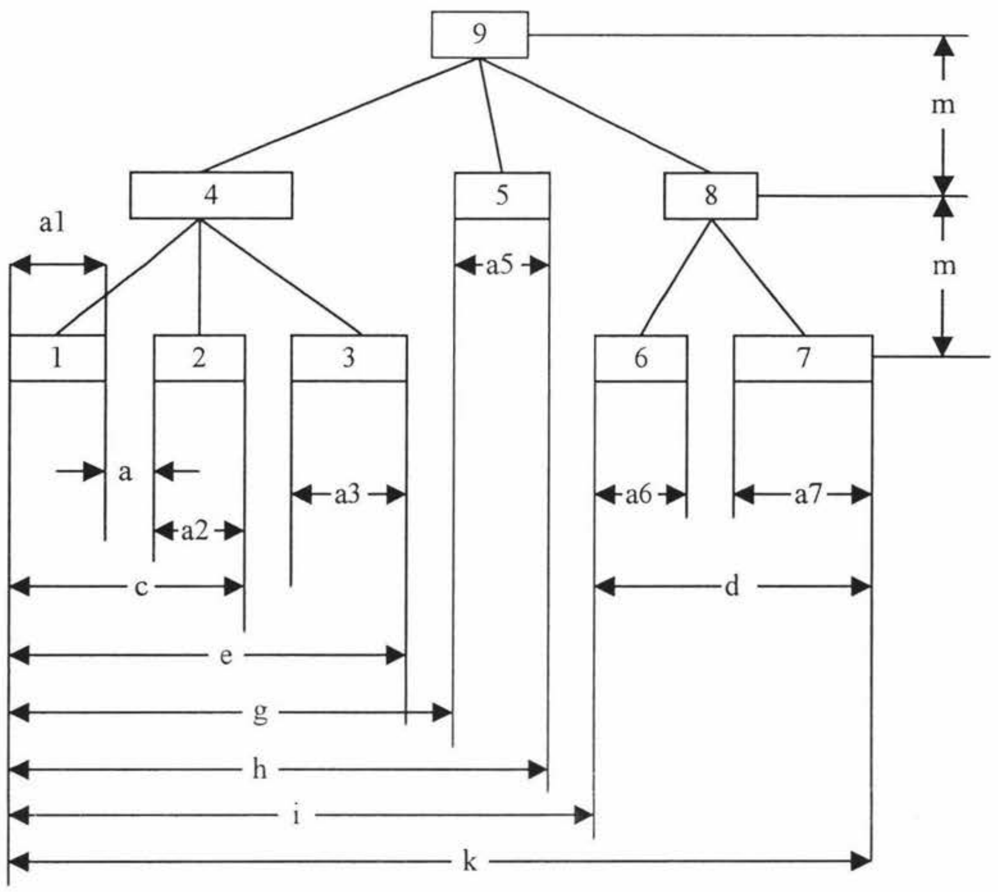


Figure 5.3 Locate nodes for a general tree

- Set distance between levels of nodes (subject/activity/object) = *m*;
- Set distance among siblings for a parent = *a*;

1. Calculate the width of the most left node 1; get the value a1;
2. Travel following post-order traversal, to node 2.
3. Node 2 does not have any children, calculate width of this node and get the value a2;
4. Calculate total a1 and a2 plus a; get c;
5. Travel to node 3;
6. Node 3 does not have any children, calculate the width of this node and get the value a3;
7. Calculate total c and a3 plus a distance a;
8. Travel following post-order traversal; no more sibling; calculate total e;

9. Travel to the parent node 4. Calculate the width of this node; get value a4;
10. Check value of a4<e, use e for total width (temporary) of the tree;
11. Travel following post-order traversal; get node 5. Node 5 does not have children;
12. Calculate the width of node 5 and get a5;
13. Add a5 with e plus the distance a and get h;
14. Travel to the sibling node 8. This node 8 has two children, node 6 and node 7; go to node 6; node 6 has no children; calculate the width of node 6 and get a6;
15. Travel to node 7. Node 7 has no children. Calculate the width of node 7 and get a7;
16. Node 7 has no children, calculate total a6 and a7 plus distance a and get d;
17. Travel to node 8. No more siblings. Calculate total h and d plus distance a, get k;
18. Locate of the root (node 9) based on k.

Following the algorithm above, if a node has a grandparent node, it must have a parent node. Explained in another way, if a grandparent node (A) has a grandchild node(C), it must have a child node (B) that is the parent node of the grandparent's grandchild node (C). In a FSCL structure tree, a node can have grandchild node but may not have a child node. The algorithm overcomes this problem by using an empty node for the "missing" node in the case a node does not have a child node, but has grandchildren nodes.

5.2.3 Strategy to verify correctness of users' input

In the active modes (the STM and VBQ modes), the users are allowed to graphically draw structure trees or textually input FSCL sentences from individual words. A vital task of VISL is to verify whether those inputs are syntactically correct or not, locate structure errors if any, and then explain why and which structure rules are violated. This section will firstly point out what are common errors made by users and provide an approach to deal with those errors in both graphical and textual inputs.

5.2.3.1 Common errors

The errors users may create when formulating FSCL sentences commonly fall into two categories:

- *Type-1.* Users may enter sentences that are correct following the structures of natural language but wrong following FSCL structures. These kinds of errors may occur when users are confused between natural language structures and FSCL structures. This type of errors can occur with both graphical and textual inputs. For example, a user may textually input a sentence "John *and* Julia help Lisa to read" or he may create a graphical tree of a sentence with four words groups of "John *and* Julia", "help", "Lisa" and "to read" for a subject, an activity and two objects respectively. In this research, VISL will focus on how to deal with those kinds of errors as they indicate the areas where user learning needs to occur.
- *Type-2.* There are obvious errors that would be wrong in both English and FSCL structures. These types of error are generated when users, by some unknown reasons, enter obviously wrong sentences. As it is assumed that the users have a basic understanding of natural language, those errors can be seen as clear mistakes. For example, users may input a wrong sentence by mistakenly entering the word "and" at the end of the sentence "Mike reads book *and"*. It can be seen that, users do not likely make these errors, as those errors are very easy to be recognized. As this kind of error is obvious, VISL simply points out the location of the errors that violate FSCL structures.

5.2.3.2 Error checking for textual input

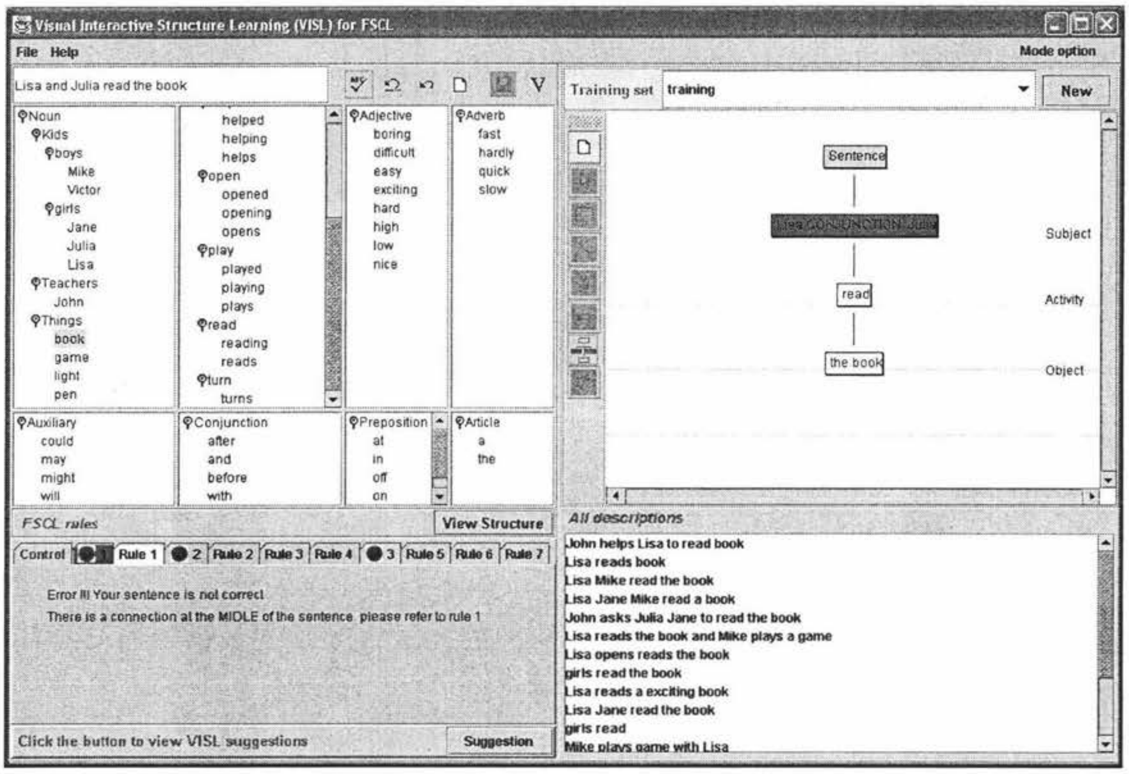
Chapter 3 pointed out that FSCL structure has restrictions compared to natural language. Therefore, users may generate sentences that violate these restrictions. The project selects all restrictions of FSCL and then builds algorithms to verify any violation of an input sentence. A sentence has a Type-1 error if that sentence violates any of these restrictions. VISL assumes that other errors as Type-2 errors. VISL verify Type-2 error by using the current FSCL parser. When VISL discovers a Type-2 error, it only displays the location of these errors and lets users to fix incorrect sentences themselves. This section discusses the errors belonging to Type-1 that may be created by novice users. Several examples below present users' common errors and explain how VISL deals with these errors.

- Users use connection words for multi-subject, multi-activity or multi-object in a sentence. For example, a user may enter sentences as the followings:
    1. *Lisa and Julia* read the book.
    2. John asks *Mike and Lisa* to stop talking.
    3. John hits *Jim and Lisa.*

  As FSCL does not allow users to enter connection words for multi-subject, VISL will detect the errors, locate the errors in the graphical trees and display error messages for these incorrect sentences. In addition, VISL will point out the rules that users have violated to help users to understand which errors they have made. Moreover, VISL presents a solution to users if they are not able to fix errors or if they want to verify their answer. Error sentences belongs to this type if in these sentences:
    - There is connect word "and" or "or" between noun words, and
    - The nouns after the connection word are not verbs or the sentence is ended after these nouns.

  Following the rules above, one can recognise that the sentence "John plays a game and Lisa reads a book" is correct.

To link this type of error to "multi-subject" rule, the sentence with the error will be marked as it *has* "multi-subject" structure. Therefore, after a user is informed that there is an error, he can view the "multi-subject" rule that is highlighted. Figure 5.4 presents the UI of VISL when a user enters an incorrect sentence "Lisa and Julia read the book". In the structure diagram of this sentence, the error message is displayed and the error node is highlighted. The user then can go to view the "multi-subject" structure rule.



Lisa and Julia read the book

training

Error !!! Your sentence is not correct

There is a connection at the MIDLE of the sentence please refer to rule 1

Click the button to view VISL suggestions

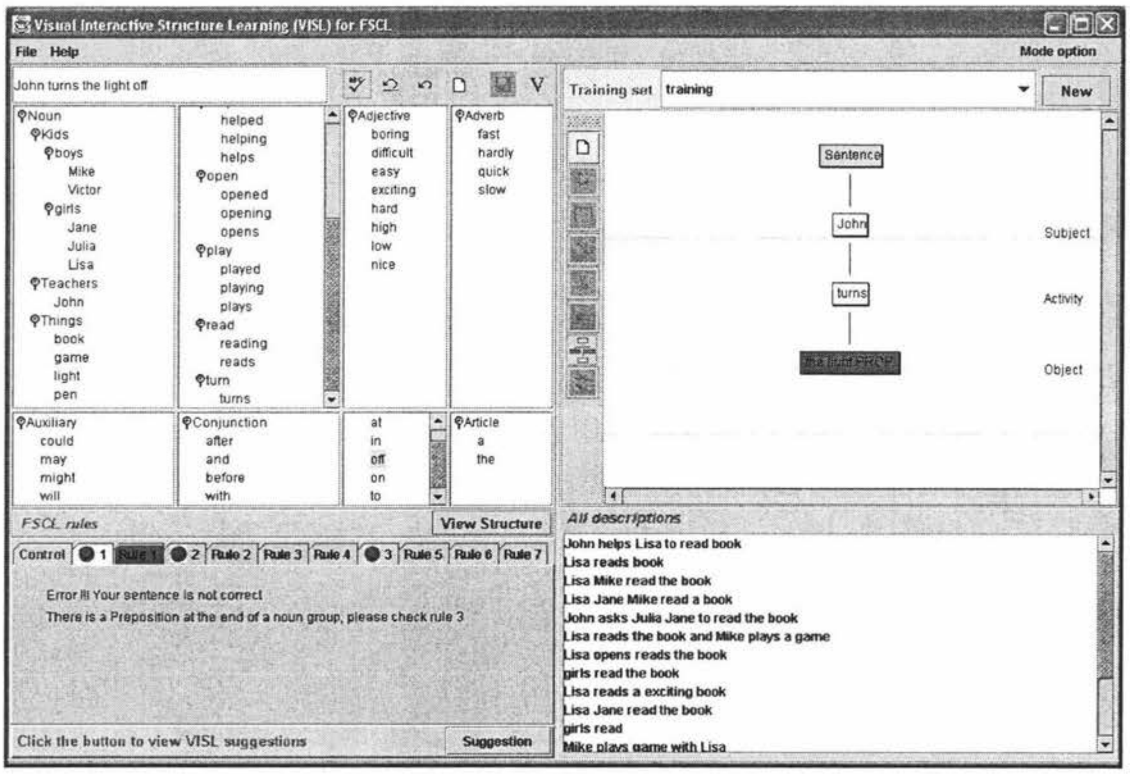
John helps Lisa to read book
Lisa reads book
Lisa Mike read the book
Lisa Jane Mike read a book
John asks Julia Jane to read the book
Lisa reads the book and Mike plays a game
Lisa opens reads the book
girls read the book
Lisa reads a exciting book
Lisa Jane read the book
girls read
Mike plays game with Lisa

Figure 5.4 UI of the VISL application for a violating rule 1

There may be several errors in one input sentence. VISL store a list of errors and then inform to users.

- Users enter sentences with the structure of "verb + noun + preposition". For example, a user may enter incorrect sentences as following:
    1. John *turns* the light *off*
    2. Lisa *lifts* the chair *off*.

FSCL structure does not allow users enter sentences with the structure of "verb + noun + preposition". In a sentence, if after verbs, there are nouns and after these nouns, there is a preposition, the sentence has this type of error. Figure 5.5 presents the UI of the VISL application when a user enters a wrong FSCL sentence: "John *turns* the light *off*".



John turns the light off

Error !!! Your sentence is not correct
There is a Preposition at the end of a noun group, please check rule 3

John helps Lisa to read book
Lisa reads book
Lisa Mike read the book
Lisa Jane Mike read a book
John asks Julia Jane to read the book
Lisa reads the book and Mike plays a game
Lisa opens reads the book
girls read the book
Lisa reads a exciting book
Lisa Jane read the book
girls read
Mike plays game with Lisa

Figure 5.5 UI of the VISL application for an error sentence

5.2.3.3 Structure checking for graphical input

When users want to build a structure tree for a FSCL sentence, they first create individual nodes from individual words. They then may want to combine these individual nodes into a "larger" node to present a word group. For example, users can drag two individual nodes of two words "the" and "book" to create a single node "the book". VISL will check these word groups of nodes to see whether they are correct or not following FSCL structure rules (Section 4.2.2). This word group-checking step assists users to understand how word groups are created to build components in a FSCL structure tree such as subjects, objects and activities. Therefore, VISL offers a mechanism to discover errors in any word group and explains why that word group violates FSCL structures. After a tree is completely built, its structure is checked as a whole. The word group-checking process in VISL is presented as the following:

- Store words of word groups of a tree that has been built in a temporary place.
- Update the current words of these word groups when users add or delete any individual word in a word group.
- Validate of these word groups following FSCL structure rules and inform users about possible errors. The correctness of a word group is based on the

categories of words in that word group and its function in the tree (subject, activity or object). Some examples below explain this:

- A multi-subject/activity/object node must not have connection word. However, a connection word can stay alone. For example, in the sentence "John reads and Mikes plays", the word "and" can stay as a subject word.
- In an object node, the preposition word must stay in the front of noun words (e.g. off the light, not the light off).

One can recognise that a word group (or node) is correct or incorrect depending on its function in a structure tree. For example, a word group "to John" is incorrect when it is drawn in the subject level in a tree. However, this word group is correct if it is drawn in the object level.

5.2.4 Some other issues

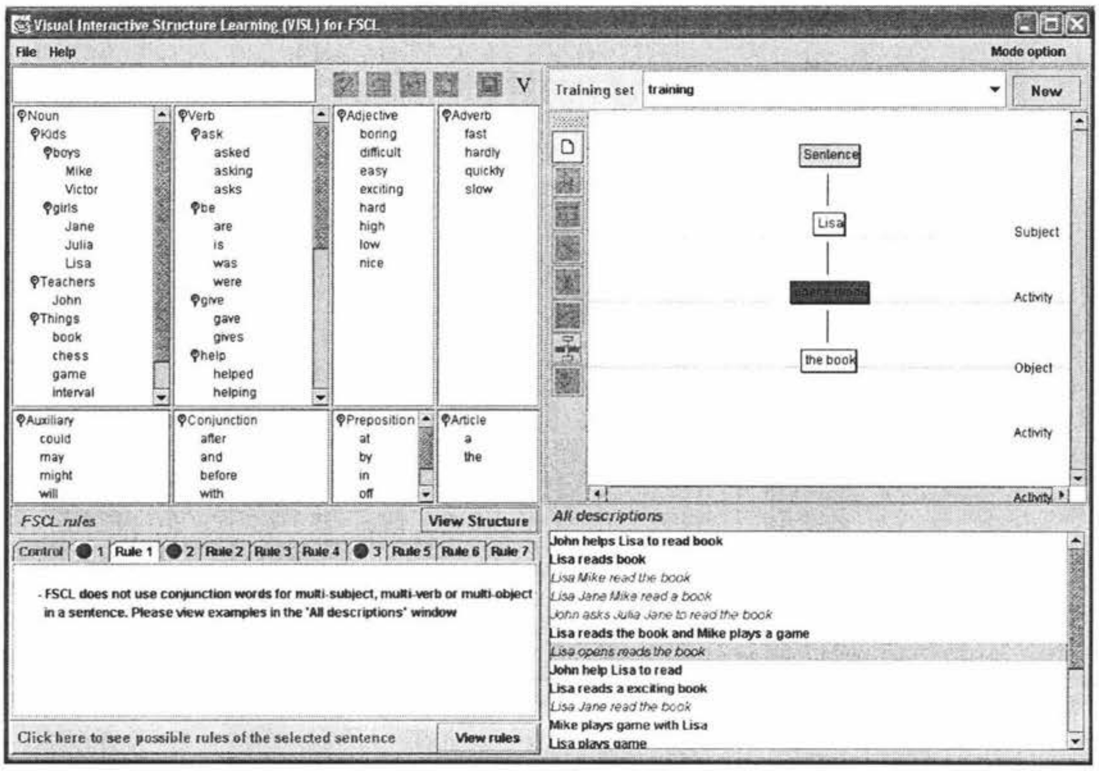
As discussed in Section 4.2.1, the VST mode allows users to view possible structures or restriction inside a FSCL sentence. This section clarifies how VISL verifies the structures for a FSCL sentence and how users can modify a word group when constructing a structure tree.

5.2.4.1 An approach to verify rules of a FSCL sentence

To find out the structure rules of a selected sentence, VISL obtains the word groups in this sentence and detects the rules relevant to each group of that FSCL sentence and presents those rules to users. Here are some examples of how VISL can detect rules (or restrictions) of a FSCL sentence:

- In the FSCL structures, there is rule claiming that FSCL sentences do not allow connection words in a multi-subject, multi-activity or multi-object. Therefore, if in the subject group or object group, the number of noun words is more than one, the sentence complies this structure rule. In addition, if in an activity word group, the number of the verbs is more than one, the sentence also follows this structure rule.

Figure 5.6 presents an example of the structure diagram for the sentence "Lisa opens reads the book". In this sentence, the activity word group containing two verbs "opens reads" is highlighted.



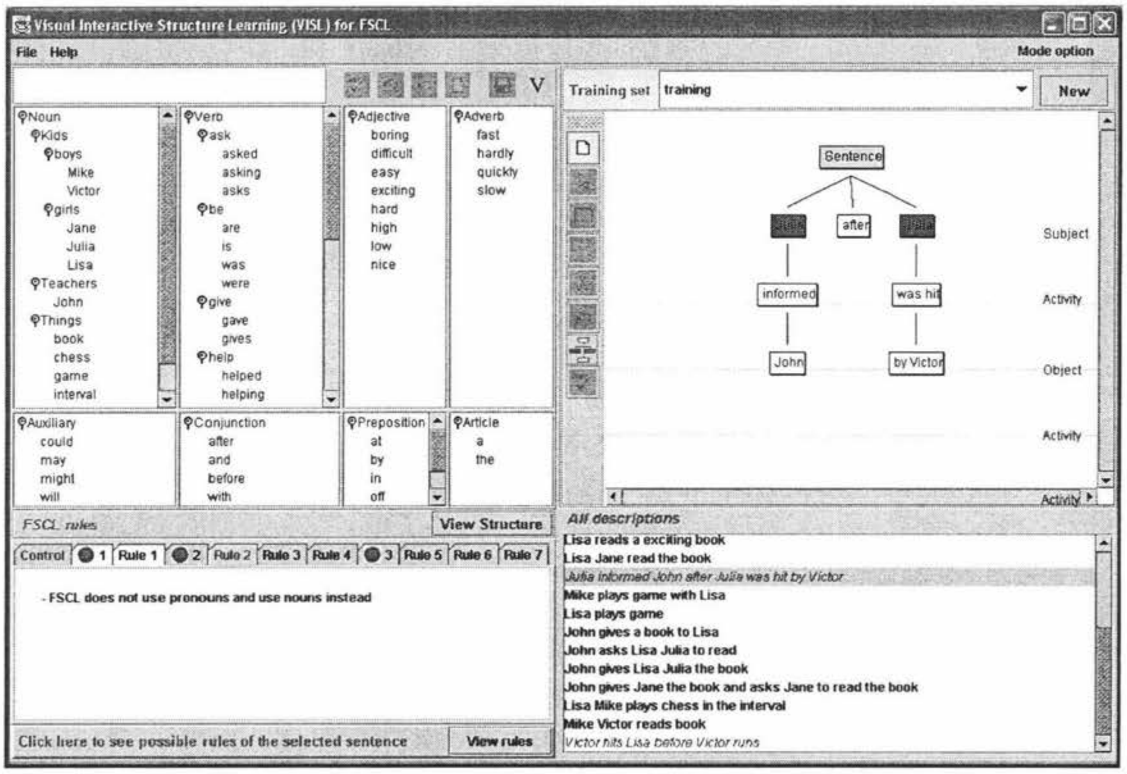
- FSCL does not use conjunction words for multi-subject, multi-verb or multi-object in a sentence. Please view examples in the 'All descriptions' window

John helps Lisa to read book
Lisa reads book
Lisa Mike read the book
Lisa Jane Mike read a book
John asks Julia Jane to read the book
Lisa reads the book and Mike plays a game
Lisa opens reads the book
John help Lisa to read
Lisa reads a exciting book
Lisa Jane read the book
Mike plays game with Lisa
Lisa plays game

Figure 5.6 An example of a sentence with multi-subject

- One of the FSCL structure rules claims that FSCL does not use pronouns but uses nouns instead. Therefore, if any noun in the subject group is replicated in other groups, the sentence complies this rule.

Figure 5.7 presents the structure of the sentence "*Julia* informed John after *Julia* was hit by Victor" with the highlighted nodes "Julia". One can recognise that, using natural language, a user can write "Julia informed John after she was hit by Victor".

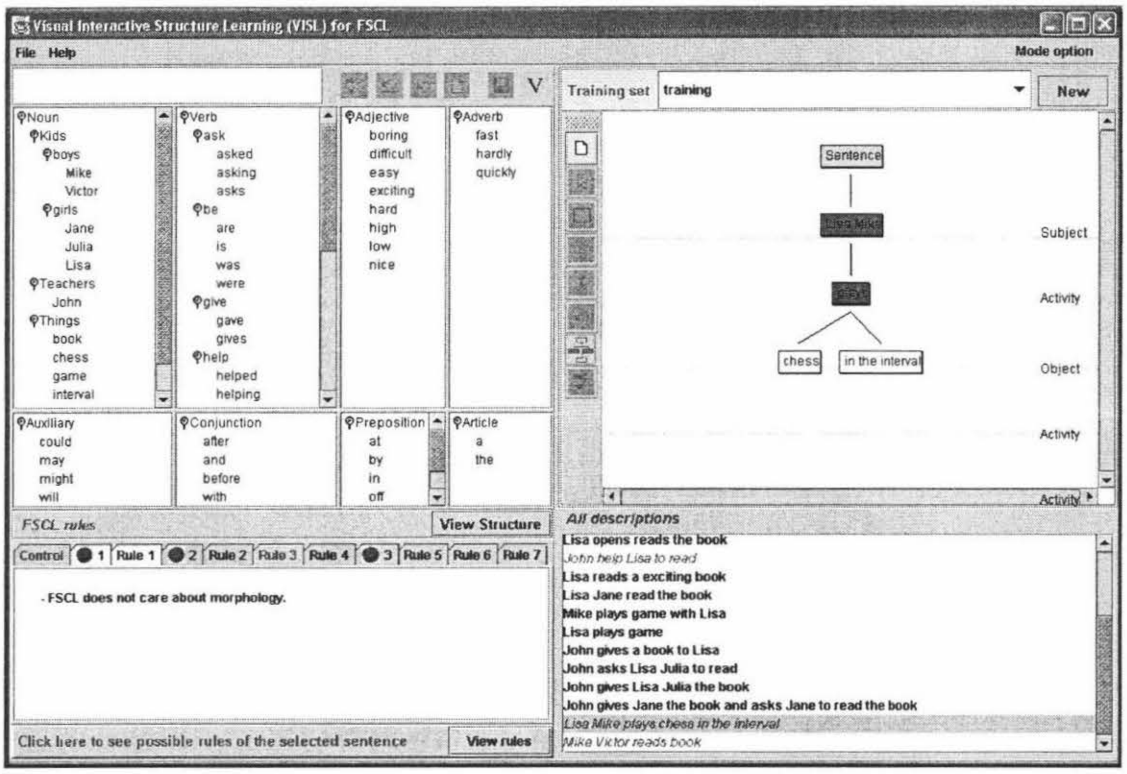


- FSCL does not use pronouns and use nouns instead

Lisa reads a exciting book
Lisa Jane read the book
Julia informed John after Julia was hit by Victor
Mike plays game with Lisa
Lisa plays game
John gives a book to Lisa
John asks Lisa Julia to read
John gives Lisa Julia the book
John gives Jane the book and asks Jane to read the book
Lisa Mike plays chess in the interval
Mike Victor reads book
Victor hits Lisa before Victor runs

Figure 5.7 An example of noun instead of pronoun sentence

- There is one FSCL structure rule stating that FSCL sentences do not follow morphology as natural language does. As WordNet can determine whether a verb is in plural or single form (see WordNet, 2004 for more details), VISL uses WordNet to verify if a FSCL sentence follows this structure rule or not. For example, if in a sentence having *multi-subject* or *plural subject* and a verb word in the active word group is in the *single* form, this sentence complies the rule of "FSCL does not care about morphology". Figure 5.8 presents a structure diagram of the sentence following this structure rule: "Lisa Mike *plays* chess in the interval". In this diagram, the nodes "Lisa Mike" and "plays" are highlighted.



- FSCL does not care about morphology.

Lisa opens reads the book
John help Lisa to read
Lisa reads a exciting book
Lisa Jane read the book
Mike plays game with Lisa
Lisa plays game
John gives a book to Lisa
John asks Lisa Julia to read
John gives Lisa Julia the book
John gives Jane the book and asks Jane to read the book
Lisa Mike plays chess in the interval
Mike Victor reads book

Figure 5.8 An example of FSCL sentence disregarding about morphology

In some cases, a sentence may comply with more than one structure rule. VISL will present a list of rules contained in that sentence. The headings of these rules in the FSCL rules panel also are highlighted to help users to find the rules that sentence is related to.

5.2.4.2 Tree structure modification

When building a FSCL tree, users may want to modify a word group in the sentence such as add or delete a word in this word group. To assist users to do so, the application provides a dialog box to help users to modify the selected word group in a tree easily. Figure 5.9 presents a dialog window that allows a user to change the content of the node "the teacher" by adding and removing words.

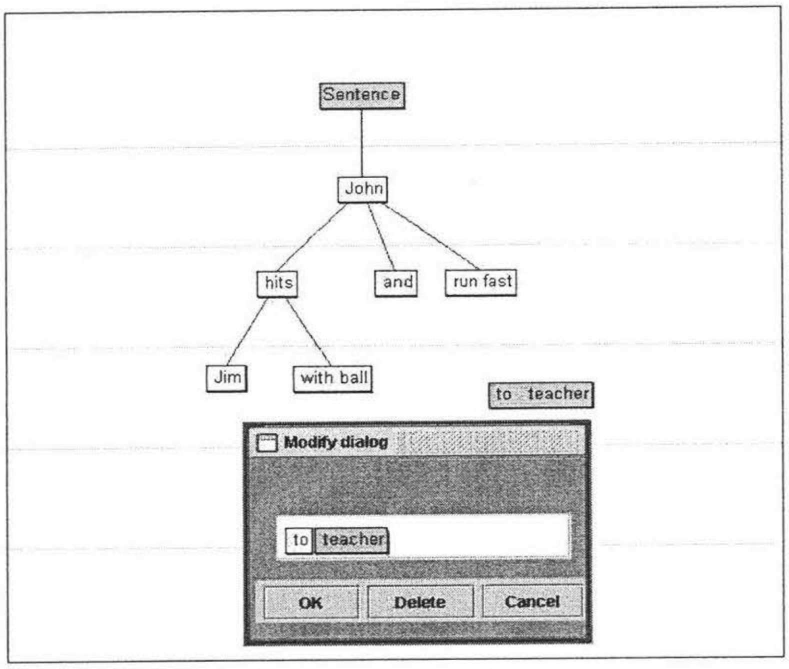


Figure 5.9 Modifying a word group

5.2.5 Algorithm for query matching

As discussed in Section 3.2.3.3, there is a requirement for presenting matching patterns between query and descriptive sentences. In addition, when a descriptive tree matches a query tree, the matching pattern between the two trees is presented in a different format to get users' attention. In this research, VISL displays the matching pattern in the descriptive tree with a different colour from other nodes.

With these requirements above, a query-matching algorithm is required to solve three tasks: determine whether a descriptive sentence matches a query sentence, find out the exact location of these matching nodes, and change colour for the matching pattern. Heinrich (1999) presented an algorithm to find matches between descriptive sentences and query sentences. However, this method is time consuming as it uses a recursive algorithm. In addition, the algorithm is not suitable for changing colour of matching nodes. Therefore, VISL develops its own algorithm that can handle the tasks above. This section first presents the algorithm in detail then provides an example to explain how the algorithm works. At the end, the section explains why the static visual display is chosen to present matching patterns between query and descriptive trees.

5.2.5.1 VISL query matching algorithm

The algorithm includes steps below:

- Convert the descriptive sentence into an array of nodes. Each node stores information about the word group of that node and its level. This array is called the descriptive array.
- Convert the query tree into an array as in the procedure above. This array is called the query array.
- Start from element 0 of descriptive array and query array, do:

  While not end of descriptive array do:

  Compare element ***i*** of descriptive array and element ***j*** of query array

  - If there is a match,

    - Set position ***j*** to the next element of the query array;

    - If ***j*** = the query array size, return true- two sentences are matched;

    - Change the background colour of the node at position ***i*** of the description array;
  - If there is no match,

    - Search backward from *(j-1)* to 0 of the query array. Find the first element of ***(j-1)*** to 0 of the query array that is matched with the current element ***i*** of the descriptive array.

    - If found,

      Set ***j*** to the position of this element of the query array;

      Search back in the descriptive array from ***i*** to the position of the element, which is matched with the element ***j*** in the query array.

      If there are changed colour nodes, change them back to original colour;
    - If not found,

      Keep current ***j***;

      Increase one element in the descriptive array;
- Return false - the two sentences are not matched.

5.2.5.2 An example illustrating the matching algorithm

The example uses the descriptive sentence "John gives Jane the book and asks Jane to read" and the query sentence "Jim asks Jane to read" to illustrate the matching algorithm and change the colour for matching nodes. After converted into arrays, the two sentences are presented as follow.

Descriptive Array (DV) for the description tree:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1-John,	2-gives,	3-Jane,	3-the book,	2-and,	2-asks,	3-Jane,	4-to read,

Query Array (QV) for query tree:

[0]	[1]	[2]	[4]
1-John,	2-asks,	3-Jane	4-to read

In this presentation, the first line displays the order of word group in the array. In the second line, the first item of each element is node's level and the second item is the node value. The structure trees of the two sentences are shown on Figure 5.10.

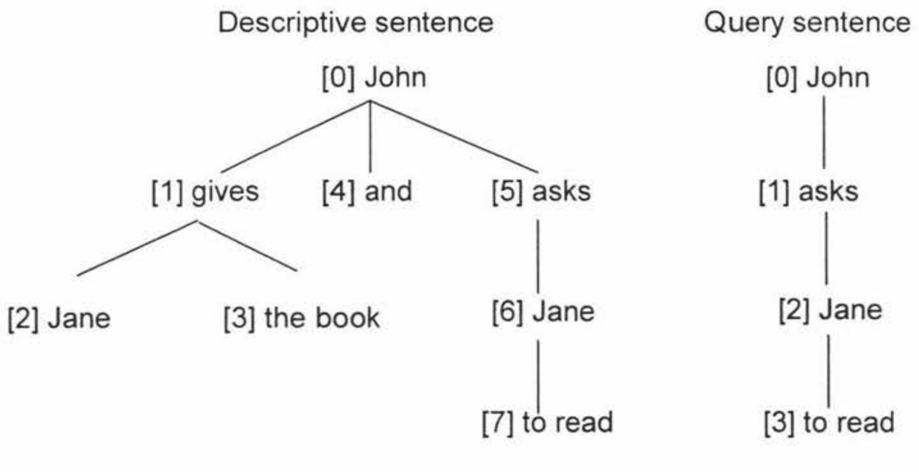


Figure 5.10 Structure trees of descriptive and query sentences

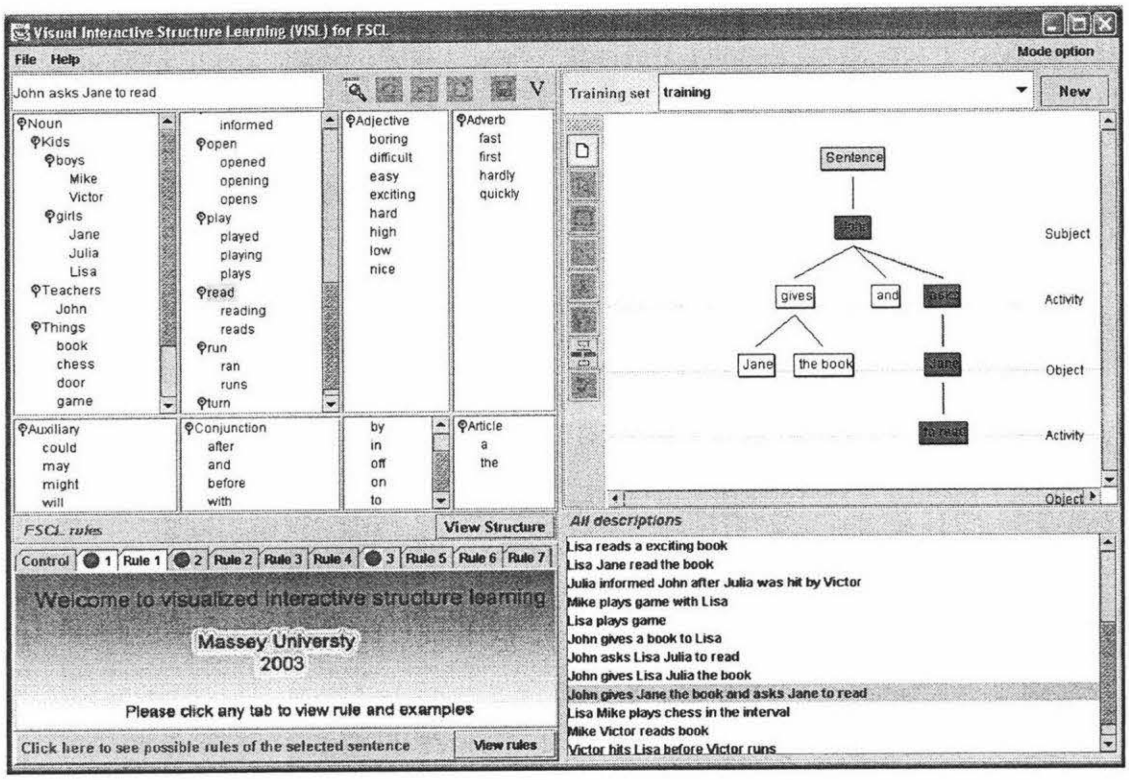
Starting from the first element in the Descriptive Array (DA) and Query Array (QA),

1. Item [0-John] of DA matches item [0-John] in QA.

2. Change colour of the node [0] in DA (0-John);
3. Increase DA and QA to items [1];
4. Item [1-gives] of DA does not match item [1-asks] in QA. Search backward items from [1] to [0] in QA. None of them matches item [1] of DA. Keep item [1] in QA;
5. Increase DA to item [2-Jane]. No match with [1-asks] of QA. Search backward items from [1] to [0] in QA. No math;
6. Increase DA to item [3-the book]. No match with [1-asks] of QA. Search backward items from [1] to [0] in QA. No math;
7. Increase DA to item [4-and]. No match with [1-asks] of QA. Search backward items from [1] to [0] of QA. No match;
8. Increase DA to item [5-asks]. Match with [1-asks] of QA;
9. Change colour for item [5-asks] in DA;
10. Increase items in DA to [6-Jane] and in QA to [2-Jane];
11. Item [6-Jane] in DA matches item [2-Jane] in QA;
12. Change colour for item [6-Jane] in DA;
13. Increase items in DA to [7-to read] and in QA to [3-to read];
14. Item [7-to read] in DA matches item [3-to read] in QA;
15. Change colour for item [7-to read] in DA;

All elements in query array are matched, return true.

Figure 5.11 presents a matching pattern between a query sentence and a descriptive sentence to illustrate the result of the algorithm above. The user enters a query "John asks Jane to read" then selects the description sentence "John gives Jane the book and asks Jane to read". The matching pattern is presented in a red colour.



John asks Jane to read

Lisa reads a exciting book
Lisa Jane read the book
Julia informed John after Julia was hit by Victor
Mike plays game with Lisa
Lisa plays game
John gives a book to Lisa
John asks Lisa Julia to read
John gives Lisa Julia the book
John gives Jane the book and asks Jane to read
Lisa Mike plays chess in the interval
Mike Victor reads book
Victor hits Lisa before Victor runs

Figure 5.11 Matching pattern for a query and a descriptive sentence

5.2.5.3 Static visual display versus dynamic visual display

To present the matching pattern between a descriptive tree and a query tree using the algorithm above, there are two possible approaches. One is the static visual approach in which the matching pattern is presented only after the algorithm is finished. Another approach is the dynamical visual display in which each step of the algorithm will be visually presented when the algorithm runs.

Many studies found that the dynamic visual displays (DVDs) might improve cognitive processes. They suggest that DVDs could help learners understand more computer science theories in presenting data structures (Stern and Naish, 2002; Hamilton-Taylor and Kraemer, 2002). For example, the process of abstracting data, operations, and semantics of computer applications can be presented by animated graphical views. This helps students understand how the algorithms work. In the case of the query-matching algorithm in VISL, users could see how each steps of the algorithm is followed and the colours of these nodes are changed when the algorithm is running. However, Hopkins and Park (1993) found that there are no superior effects of DVDs compared with a static visual display. Moreover, this project only focuses on why a descriptive tree matches a query tree and where the matching branches are in these descriptive and query trees. The focus of this research is not on teaching the user how the matching algorithm works. Therefore, the static visual display was chosen in the VISL application to present matching patterns.

5.3 Summary

This chapter discusses a range of key issues regarding how the concepts of VISL analysed in the previous chapter are implemented in a CALL application to teach FSCL's structures and its query engine. The chapter was separated into two parts and these are summarised as follows.

The first part discussed VISL's development issues. It first discussed features applied in VISL to provide a friendly learning environment for users to learn FSCL. The part then explains why a combination of PAC and VISL was chosen in this project. At the end, this part reviewed some concepts for designing efficient GUIs for training systems and discusses how those concepts were applied in VISL.

The second part covered technical issues regarding to FSCL structures and its query engine. This part first explained how VISL divides the unique structures and matching conditions of FSCL into simple rules and presents them in easy lessons. Each structure or matching rule is presented by a textual explanation and illustrated by graphical trees to help users to learn effortlessly the FSCL structures and its query engine. The part also discussed how to draw balanced trees to present FSCL structures. At the end, this part presented a matching algorithm to display the matching pattern between descriptive and query trees that is the key task of the VQM and VBQ modes.

The following chapter, Chapter 6, focuses on the development of a CALL application that implements the concepts discussed in the previous chapter, Chapter 4, and that were further clarified in this chapter. The Chapter 6 will explain in detail how software engineering concepts are used to construct the VISL application with the use of Object Oriented Modelling (UML), and the Java programming language.

Chapter 6

Development of VISL

In Chapter 4, the concepts for teaching constructed languages were established. In Chapter 5, key issues towards an application for teaching FSCL were highlighted. This Chapter 6 presents the VISL development process based on these concepts and key issues from a software engineering perspective. The chapter is divided into two sections. The first section reviews object-oriented tools used for developing the VISL application including the Unified Modelling Language (UML) for analysis and design, and the Java programming language for implementation. The second section discusses how the VISL application is developed using these object-oriented tools.

6.1 Object-Oriented tools for VISL development

Object-Oriented (OO) technology has many advantages compared to traditional development methods such as structured analysis and structured design as it helps to "reduce development time, reduce the time and resources required to maintain existing applications, increase code reuse, and provide a competitive advantage to organizations that use it" (Shah et al. 1997, p1). Therefore, in this research, OO technology was applied for the development of the VISL application. The project used the Unified Modelling Language (UML) for modelling the VISL application, a model drawing tool called Rational Rose and the Java programming language for the implementation. This section briefly reviews some features of these tools before discussing how these tools were applied to develop the VISL application.

6.1.1 UML, a standard modelling technique

UML was developed by Grady Booch, Jim Rumbaugh and Ivar Jacobson and accepted as a standard OO methodology in 1997 (OMG news, 1997). It has been developed based on a combination of all the best features from various OO system analysis and design methods. In addition, UML adds concepts and notations that have proven to be useful but are missing from those earlier methods. With advantage features, UML is able to support modelling complex systems (Douglass, 2000). In general, UML provides comprehensive views of a system including:

- *Functional view*: describes the system's functional requirements. UML uses use case diagrams and activity diagrams to present the static functional view and the dynamic functional view separately.
- *Dynamic structure view*: shows the transition behaviours of the system using sequence diagrams and collaboration diagrams. In addition, UML uses state transition diagrams to show the status changes of an object.
- *Static structure view*: presents overall structure of the system. Class diagrams are used to present the relationship of the classes and objects in the system. An object diagram presents a particular configuration in which it shows a set of objects and links at specific moments in time during the execution of the system.

In the Section 6.2, the key UML diagrams for developing the VISL application are discussed in detail.

6.1.2 Rational Rose

Rational Rose is an object-oriented Unified Modelling Language (UML) software design tool intended for visual modelling and component construction of enterprise-level software applications (TechTarget, 2004). The tool provides software developers with a complete set of visual modelling tools to develop robust and efficient solutions to satisfy business needs. It also helps developers to model the system before writing any code to ensure that the system is architecturally reasonable from the beginning. In addition, Rational Rose diagrams describe in great detail what the system will include and how it will work so that developers can use the diagrams as model for the system to be built. In this project, the tool is used to assists developers to create all UML diagrams including use case, activity sequence, collaboration, and class diagrams.

6.1.3 Java programming language

Java is a pure object-oriented programming language developed by Sun Microsystems. As it strongly supports multimedia technology using Java Media

Framework (Sun Microsystems 1, 2004), the language was chosen for developing the PAC application. The language also provides powerful components for developing graphical applications. Therefore, the language is suitable for implementing the VISL application. This section reviews a Java library of graphical components, called Abstract Window Toolkit (AWT) that will be used for the VISL implementation to create the user interface, and to draw and manipulate structure trees.

AWT includes platform independent components that are used to build graphical components. AWT provides basic user interface components such as buttons, lists, menus and text fields, etc. AWT also provides event handling mechanisms and image manipulation. An extension of AWT, Java2D, provides enhanced two-dimensional graphics, text and imaging capabilities for Java programs (JavaTech, 2004). Therefore, AWT components and Java2D, are used to develop the VISL application. The section provides more details about Java graphical components used in drawing and manipulating graphical elements in the VISL application including panels, canvas, graphics, and event handlers.

- *Panel and Canvas:* In Java, a panel is a rectangular area in which users can place user interface components such as buttons, menus or text fields. To support drawing 2D dimensional graphics such as Rectangle, Lines and String, Java has a specific component called Canvas. A canvas is simply a rectangular area in which users can draw diagrams by using "paint" methods. In the VISL application, canvases are used for drawing and modifying structure trees as well as displaying matching patterns in a tree.
- *Graphics:* In Java, every component comes with its own Java graphics object. All drawing must go through this graphics object. This object contains a collection of functions to draw images and texts. For example, to draw a rectangle in a canvas, the code below might be used:

    g.drawRect (*x*, *y*, *l*, *w*);

    in that:

        g is an instance of the graphic object of the canvas;

        x, *y* are positions of the top left position of the Rectangle;

        l, *w* are the length and width of the Rectangle.

Displaying text is considered as a special kind of drawing. For example, the syntax below presents a method that defines font and size for a string "The book" and then draws it at the location X and Y:

```
g.setFont (Font font);
g.setSize (int size);
g.drawString ("The book", int X, int Y).
```

A semantic tree can be seen as a combination of nodes and lines. A node is an integration of text and a rectangle. To draw a semantic tree in a canvas, the canvas's graphics object has to locate the positions of nodes in the tree before drawing those texts, rectangles and lines (the algorithm to locate those nodes was already presented in Section 5.4).

- *Event handling in Java:* One of the most important aspects of most non-trivial applications is the ability to respond to events that are generated by the various components of the application in response to user interactions. In Java, to handle an event, a listener is registered with a component (e.g. a button, menus or a node in a graphical tree). When a component fires an event, the event will be transmitted from that component to the listener (called event listener). What this event listener will do next depends on methods defined in it. In VISL, mouse events are commonly used for drawing nodes or lines to build graphical trees, and moving, deleting or modifying any highlighted node. The following example presents the event handling for drawing a straight line in a canvas.

  At first, the canvas registers with it an event listener called MouseListener to "hear" events. When a user clicks a mouse button inside the canvas, an event is created and it actives the MouseListener. A mousePress method defined inside this MouseListener will be called. In the mousePress method, the two functions getX and getY obtain values X_1 and Y_1 of the mouse's position. Those functions then inform the graphics object (the graphics object that is associated with the current canvas) of the start location (X_1, Y_1) of the line. When the user drags the mouse, a method mouseDrag in the MouseListener is called to inform the graphic object to draw a line from the start location (X_1, Y_1) to the current location of the mouse pointer and to update that line while the mouse is moving. When the user finishes drawing the line, a function mouseExit will be called and the end point of the line is located. The line connecting the first point and the second point in the canvas is then completed.

6.2 Object-oriented Analysis and Design of VISL

UML has been applied for the analysis and design processes in this project. Various kinds of UML diagrams are used such as use case diagrams, activity diagrams, interaction diagrams and class diagrams.

6.2.1 Use case analysis

Use cases describe an overall picture of what is planned to happen in the system. In other words, use cases illustrate the interactions between users and the system to perform functions inside the system. This section introduces the use cases for the VISL application and then presents one use case in detail.

6.2.1.1 Determine Use Cases

As discussed in Chapter 4, there are four modes in the VISL application. In the first mode, users can view FSCL structure rules or view rules of a sentence. In the second mode, users can graphically or textually input FSCL sentences. Users then can check the correctness of those sentences. In the third mode, users can view rules that explain how a descriptive sentence can match a query sentence. In the last mode, users are allowed to build their own queries, and then search for descriptive sentences from a database using those queries. From those interactions between users and the application, the use cases inside VISL application are created and displayed in Figure 6.1.

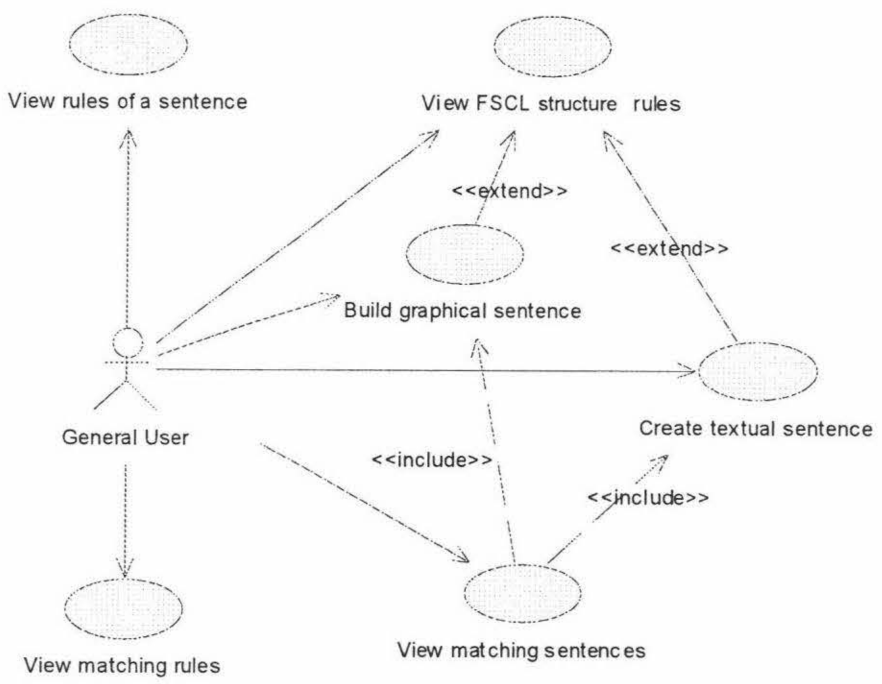


Figure 6.1 Use case diagram of the VISL application

6.2.1.2 Details of a specific use case

There are six use cases in the system (Figure 6.1). This section describes a use case "View FSCL structure rules" in detail.

Use Case Name:	View FSCL structure rules
Actor:	General User
Description:	The actor selects a FSCL structure rule to view its details or view the structure diagram of any sentence illustrating that rule.
Precondition:	The application presents a list of descriptive sentences with a list of FSCL structure lessons.
Main flow:	

The system presents a list of FSCL structure rules or restrictions. When the user selects a rule, the content of the rule is displayed. The sentences illustrating the selected rule will be highlighted inside the descriptive list. The semantic tree of the first sentence of the highlighted sentences will be presented. The word groups complying with the selected rule in that sentence will be highlighted in the semantic tree. The users can select another highlighted sentence in the descriptive list to view its structure and the nodes that follow the selected rule. The use case ends here.

Post-condition: FSCL sentences complying with the selected rule are highlighted. The semantic tree of the first sentence is displayed. Nodes of word groups complying with the selected rule in this sentence are highlighted.

6.2.2 Activity diagrams

Activity diagrams are used to describe workflows of a use case graphically. They consist of activities and states, and transitions between these. With those abilities, an activity diagram can show all of the possible events for a use case in one place. The activity diagram of the "View FSCL structure rules" use case is presented in Figure 6.2.

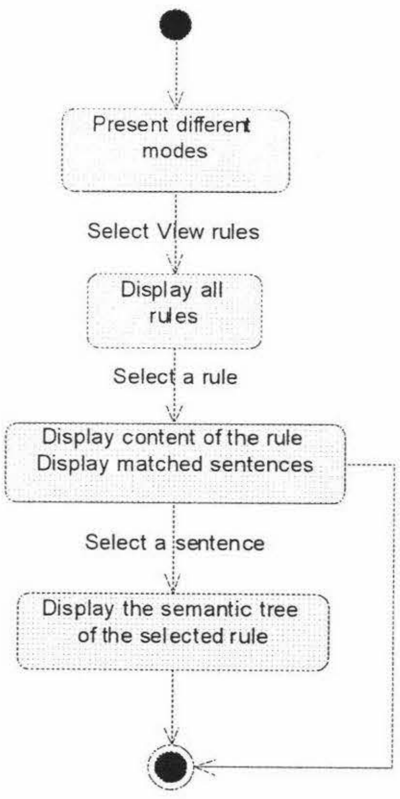

Present different modes
Select View rules
Display all rules
Select a rule
Display content of the rule
Display matched sentences
Select a sentence
Display the semantic tree of the selected rule

Figure 6.2 Activity diagram for "View FSCL structure rules" Use Case

6.2.3 Interaction sequence diagrams

In object-oriented systems, tasks are performed by objects interacting with each other through passing messages. An interaction is a specification of the way in which messages are sent between objects or classes in order to perform a task. Interactive diagrams are used to model these interactions then present these interactions of objects over time to fulfil a particular scenario of a use case. In this part, the interactive sequences between objects, which cooperate to fulfil the "View FSCL structure rules" use case, are presented in Figure 6.3 and described in a sequence of actions as follows:

> When an actor starts to study FSCL structure rules, he selects a FSCL rule from an object interface called FSCLStructureRuleUI. The FSCLStructureRuleUI object displays the details of the rule. It then sends a message with the rule's index to a control object called RulesControl. This RulesControl object scans all FSCL sentences in the list of FSCL current descriptions to locate the matching sentences for the selected rule. The indexes of matching sentences are sent to the FSCLDescriptionSet object that is inside an interface named FSCLDescriptiveSetUI. Those matching sentences will be highlighted in this interface. The first matching sentence will be converted to a DescTree object (DescTree is the presentation of a FSCL semantic tree). This DescTree object is converted to a graphical data structure through a converter. Before the tree is drawn, the RulesControl will inform a GraphicalTreeUI object to highlight the nodes that comply with the selected rule. At the end, a structure tree with highlighted nodes will be presented in the GraphicalTreeUI.

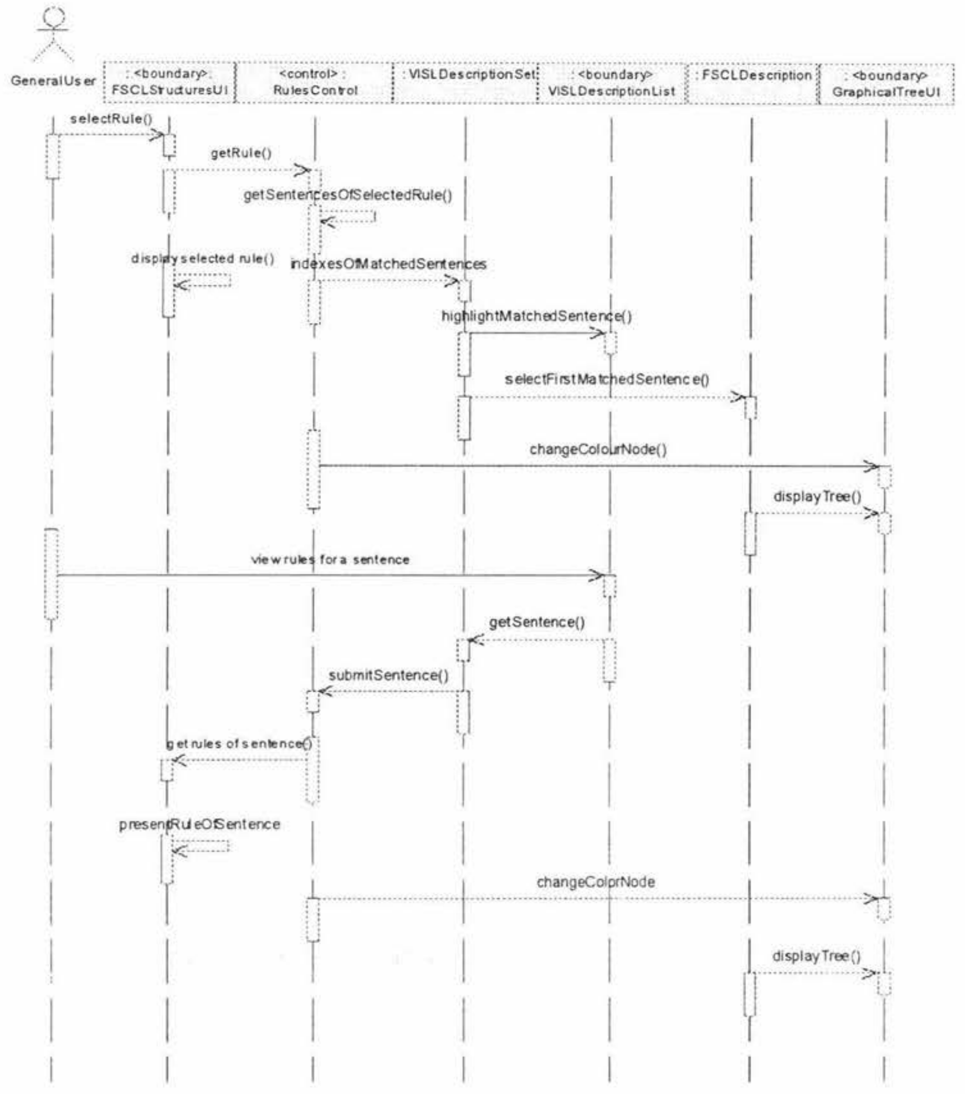


Figure 6.3 Sequence diagram for "View FSCL structure rules" Use Case

6.2.4 Collaboration diagrams

In an object-oriented system, each individual object provides a small element of the functionality of the overall system. When working together, these objects can produce a higher level of the functionality. In other words, a functionality required from the system is produced by many objects working together. Collaboration diagrams are used to show these objects with relationships that are identified by the sequence diagrams when they are working together in a real time system. Figure 6.4 presents a collaboration diagram for the "View FSCL structure rules" use case.

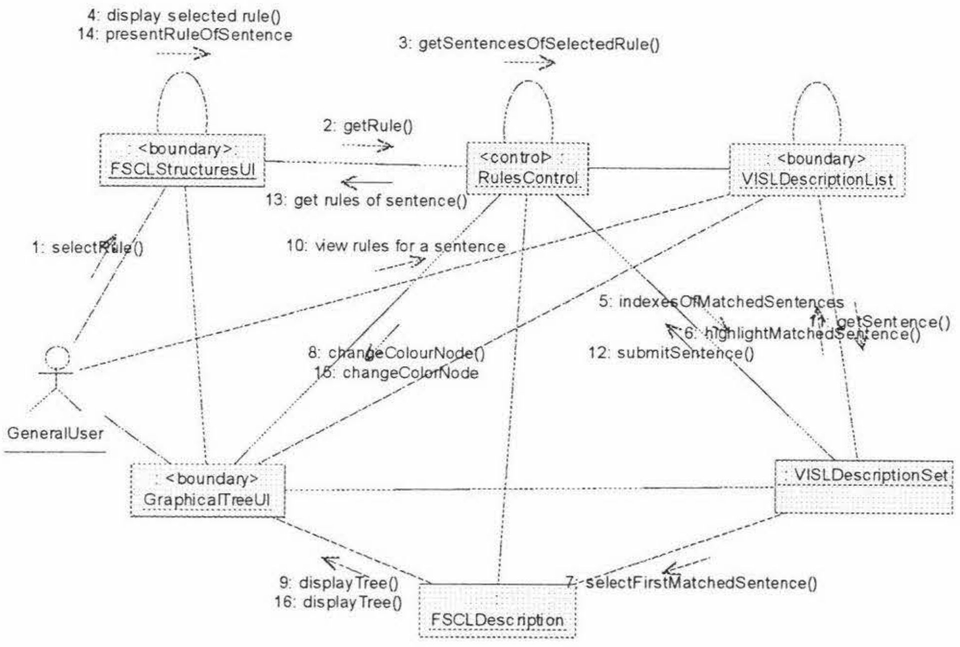

4: display selected rule()
14: presentRuleOfSentence
3: getSentencesOfSelectedRule()
2: getRule()
: <boundary> : FSCLStructuresUI
<control> : RulesControl
: <boundary> VISLDescriptionList
13: get rules of sentence()
1: selectRule()
10: view rules for a sentence
5: indexesOfMatchedSentences
6: highlightMatchedSentence()
11: getSentence()
12: submitSentence()
8: changeColourNode()
15: changeColorNode
GeneralUser
: <boundary> GraphicalTreeUI
: VISLDescriptionSet
9: displayTree()
16: displayTree()
7: selectFirstMatchedSentence()
: FSCLDescription

Figure 6.4 Collaboration diagrams for "View FSCL structure rules" use case

6.2.5 Class diagrams

In the previous sections, the interactions between objects were described through sequence and collaboration diagrams. These interactions require diagrams methods carried out by classes and relationships among classes. Diagrams that present methods of classes and the relationships between these classes are called class diagrams (Dennis et. al., 2002).

A class diagram includes attributes, methods (or operations) and constraints. From the objects presented in the sequence and collaboration diagrams (Section 6.2.3 and 6.2.4), a class diagram for the "View FSCL structure rules" use case is created and displayed in Figure 6.5.

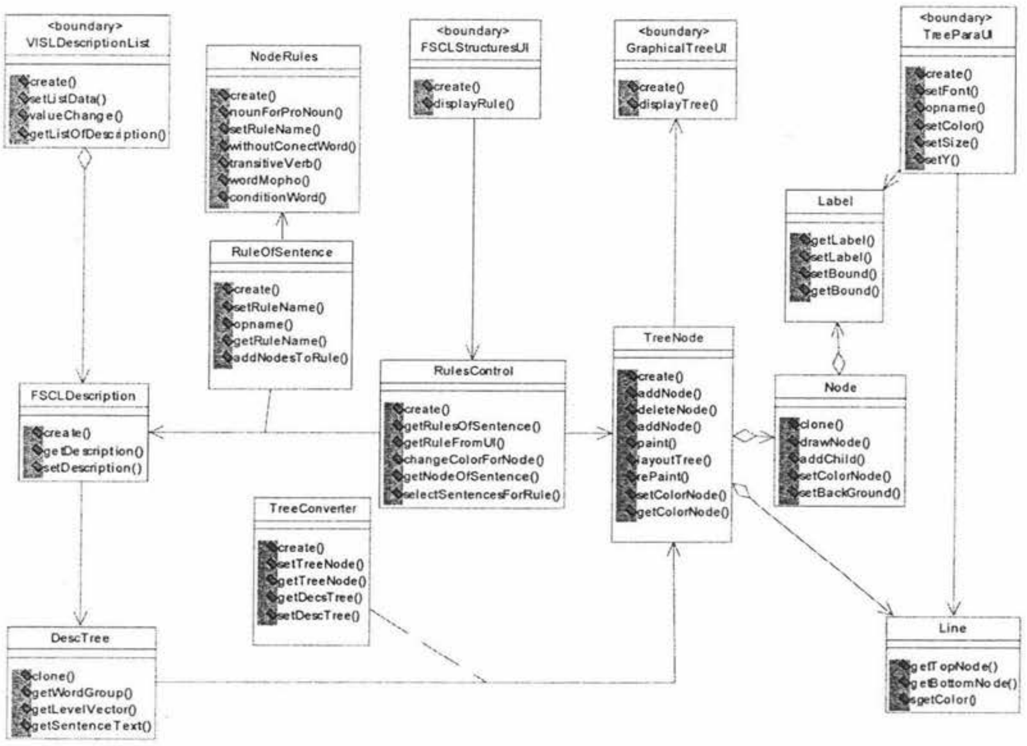


Figure 6.5 Class diagram for learning FSCL structure

Many classes have already been introduced in this class diagram through the interactions among objects presented in the sequence diagram in Figure 6.3. Five more classes are added. The four classes: Label, Node, Line and the boundary class "TreeParaUI" are considered as supporting classes used to change Font, Colour and Size for a TreeNode object. The class RuleAndNodes is used to determine what nodes in a sentence follow the selected rule.

6.3 Database in VISL

To teach FSCL structure, VISL also requires pre-defined FSCL vocabularies and FSCL descriptive and query sentences. For that reason, VISL needs a database system to store that information. To simplify the VISL development process, VISL uses the PAC database system to store all vocabularies and FSCL sentences required for the training module.

A new user, who accesses PAC for the first time, will be asked whether he wants to learn FSCL structures or not. If this user needs to learn FSCL structures, a vocabulary and pre-defined FSCL sentences used for training will be installed only for that user. If learning is not necessary, as the user may know the FSCL structures already, the user can immediately use the other modules in PAC. In this case, the pre-vocabulary and pre-defined FSCL sentences will not be installed for that user.

6.4 VISL implementation

It is straightforward from VISL's analysis and design phases to its implementation phase. All coding of classes and their attributes, operations and constraints defined in the design phase are generated using the Java programming language. All classes developed in the VISL application are combined into a package and stored as part of the PAC source code.

For example, in the VISL class diagram (Figure 6.5), there is the class called RuleControl. This class is used to find structure rules inside a FSCL descriptive sentence. It is also used to search FSCL descriptive sentences following a selected rule inside a FSCL description list. Here are some of the operations available inside this class:

- getRulesOfDescriptionSet(): get all rules of a FSCL descriptive sentence set.
- getRuleOfCurrentDescription(): get all rules of a FSCL descriptive sentence
- selectIndexOfMatchedSentence(): return indexes of all FSCL descriptive sentences that follow a selected rule.
- nodesHighlighted(): set nodes that need to be painted for a treeNode object.

The coding of the first part of the function getRuleOfCurrentDescription() inside this class can be written in Java language as the following:

```
public void getRuleOfCurrentDescription(FSCLDescription description) {
        DescTree  descTree = description.getDescriptionTree();
        Vector  wordGroups =descTree.getWordGroups();
         for (int j=0; j<wordGroups.size(); j++){      //for rule 1
               WordGroup wordGroup = (WordGroup)wordGroups.elementAt(j);
               if (isWordWithoutConnect(wordGroup)){
                      RuleAndNodes ruleAndNodes = new RuleAndNodes("rule1");
                      ruleAndNodes.addNode(j);
                      ((Vector)ruleOfSentences.elementAt(i+1)).add(ruleAndNodes);
                      }
              }
        ...

        }
```

6.5 Status of the VISL implementation

Due to time limitations, the implementation of the VISL application was not fully completed. At the current stage, the work done in this application is:

- Mode 1: Visualized Structure Tree: *completed;*
- Mode 2: Structure Tree Manipulation: *completed;*
- Mode 3: Visualized Query Matching: *completed the task of automatically presenting matching pattern;*
- Mode 4: Visualized Building Query: *completed the task of presenting matching pattern between new textual query and descriptive trees;*

6.6 Summary

This chapter discussed the development process for the VISL application following software engineering principles. The development was based on the concepts presented in Chapter 4 and the technical issues discussed in Chapter 5. At first, the chapter introduced object-oriented tools used for developing the VISL application including the modelling language-UML, the designing tool Rational Rose and the Java programming language. It then explained how these tools are used for the analysis and design phases and illustrated those phases through an example for a sub-function of the VISL application. The use of a database in VISL was also discussed. At the end, the chapter outlined the VISL implementation illustrated by a piece of simple Java code.

The next chapter, the final chapter of this thesis, will summary the development of the research. The chapter then recommends future work arising from this study.

Chapter 7

Conclusion and Future work

Three sections are presented in this chapter. The first section reviews the process of the research including the purpose of the research and how the research has addressed this purpose. The second section highlights the contributions of the research. The last section presents some recommendations and suggestions for further work.

7.1 Research review

This section first identifies the research tasks set for this research that were based on requirements of a method to teach constructed languages. The section then discusses how the research developed the visual learning concepts to accomplish those tasks and applied those concepts for teaching a specific constructed language, the Flexible Structure Coding Language, FSCL.

Research tasks

Many constructed languages have been developed in order for computer systems to overcome ambiguities and complexities existing in natural language in information description and retrieval. With the specific structures, these constructed languages help users to create unambiguous sentences or documents with correct structures. Therefore, computer applications can extract correct meanings or contents from a sentence or document without difficulty.

Advantages of constructed languages in information description and retrieval are only achieved when users apply these languages correctly. Incorrect usage of these languages can lead to creating vocabularies that are not consistent, producing descriptions that are incorrect following language structures, and generating inappropriate queries that lead to incorrect/incomplete returns. Therefore, appropriate applications designed for teaching constructed languages are required. These applications must satisfy following conditions:

- *Teach users constructed languages:* The teaching application must cover the specific features of constructed languages for creating correct descriptions and their query engines for searching for information.

- *Be simple and easy to understand:* Learning grammar/structure is one of the most boring parts of learning a new language. Therefore, the application should be simple and easy to understand. In addition it should be interesting and enjoyable enough to encourage users to learn.

Research development

The research development process was separated into four stages. Each stage is discussed in detail in each chapter (from Chapter 2 to Chapter 5). In brief, these stages can be summarised as the following:

- Diagrams are extremely useful in assisting users in understanding concepts and ideas. Therefore, in the first stage, the research reviewed how these advantages are used for enhancing users' understanding of language structures. A wide range of language grammar theories and their associated diagrams were reviewed. Computer applications using diagrams for teaching languages were also investigated. From studying of this research, it can be concluded that diagrams are very effective tools to help learners to learn language structures. This seems especially true for beginners or students who are studying linguistic at universities.
- There are several kinds of constructed languages used for many purposes in many computer systems. They share common features of having restricted vocabulary and a formulated grammar. However, each constructed language has its own specific features. Therefore, the second stage investigated in detail a typical constructed language, FSCL, for a case study. The research examined possibilities of using the language's structure diagrams to teach its structures and query engine.
- In the third stage, the concepts for an application to teach constructed languages based on their structure diagrams were developed. These concepts explored the potentials of diagrams in language learning, as they are easily understood and invariably remembered better than words on tasks of recall and recognition. These concepts offer a multifaceted interactive approach that allows users to mastermind both constructed language structures and their query engines. It provides a step-by-step learning strategy to reduce memory load and enhance users' long-term memory. It also offers an active drawing

environment that allows users to practice with learning materials. The research also explained how these learning concepts developed in this research could be adaptable to teach different constructed languages of both "natural language like" language and structured language categories.

- In the fourth stage, a detailed specification for these concepts was developed to allow the implementation of an application called VISL to teach a constructed language, FSCL, as a case study. The application was integrated with a computer application, PAC, used to study behaviour, as a learning module. Four modes were designed and implemented in this application to provide a comprehensive learning environment for teaching FSCL structures and its query engine.

7.2 Contribution

The most significant contribution of this research was the development of conceptualisations for teaching constructed languages. These conceptualisations were partly applied in a CALL application called VISL.

Conceptualisation

This research has developed the concepts for teaching constructed languages. In these concepts, the advantages of using computer systems and the richness of diagrams in presenting information are intimately combined to develop an exciting CALL application. The concepts provide a multifaceted learning approach by using passive and active learning strategies. Four modes were built for teaching constructed languages including:

- *Passive learning of language structure*: this mode exploits the benefits of diagrams in enhancing users' understanding of language structures. The important contribution of this mode is the automatic presentation of the structure diagrams of descriptions to users. This concept uses a combination of clear explanations, illustration of simple examples, and structure diagrams to teach users constructed language structures.
- *Active learning of language structures:* The mode explores the advantages of an active learning environment in enhancing users' understanding. In this active learning mode, users are allowed to interact with structure trees in

different ways. Users can freely explore possible structures of constructed languages by doing exercises such as building new trees or modifying any nodes while studying existing trees. Alternatively, users can textually input descriptions to view how their own inputs are interpreted as structure trees. In addition, the mode provides comprehensive feedback to help users to understand and correct structure errors when they are doing exercises.

- *Passive learning of query matching engine:* This mode, again, takes the benefits of diagrams in improving users' understanding of concepts and ideas. The mode provides many "matching lessons" to cover possible matching conditions of constructed languages. Explanations and illustrative diagrams are closely incorporated to teach users how to create a query and how query and descriptions match with each other. The most significant contribution of this mode is the presentation of matching patterns between queries and descriptions. By studying the matching patterns, users can easily understand how query and description are matched.
- *Active learning of query matching:* This mode explores the advantages of an active learning environment for clarifying and reinforcing user understanding. Through the repeatedly formulating new queries or modifying current queries and studying the matching results, users gain deep understanding about the search mechanisms in constructed languages. For the "natural language like" language, this assists users not only in building suitable queries to obtain better results but also in constructing appropriate vocabulary databases and descriptions to generate correctly retrievable descriptive information.

Development of the application Visualisation Interactive Structured Language

The application, Visualisation Interactive Structured Language (VISL), was developed based on the concepts specified above for a specific constructed language, FSCL. The application was built based on an object-oriented technology using the object oriented development tool UML and the Java language. The application was seamlessly integrated as an extension module in the existing computer application PAC that uses FSCL for studying behaviour. The development of VISL was divided into three stages:

- The integration of the concepts of teaching constructed languages into a CALL application: Several technical issues were solved such as how to draw a balanced FSCL semantic tree and how to present matching patterns between descriptive and query trees.
- The examination of the PAC application to see how VISL can "plug" into this system: The project examined how VISL can use some available functions in PAC (such as the parser for checking FSCL syntax).
- The employment of software engineering principles for developing the application: The project used UML techniques for modelling use cases, class diagrams, and activity diagrams for analysis and design. Knowledge of the Java graphics library was acquired for implementing the application.

Finally, the VISL application with the following functions was built to teach FSCL structures and its query engine:

1. *Teach users FSCL structures:* VISL identifies FSCL's structure rules and their restrictions compared to natural language. These rules and restrictions are organised into teaching lessons. VISL also stores a pre-defined vocabulary and a number of FSCL sentences that clearly illustrate FSCL's structure rules. VISL presents the rules of FSCL structure, descriptive sentences belonging to these rules and semantic trees interpreted from those sentences for teaching users the structures of FSCL. Furthermore, users are able to select any sentence to view its structure rules.
2. *Allow users to do different types of exercises to learn the language structures*: Users will practice what they have learnt by actively creating FSCL sentences using textual input and seeing how these sentences are interpreted into tree structures. The application provides necessary graphical tools that allow users to generate FSCL semantic trees, add, delete or modify any element in those structure trees.
3. *Provide extensive feedback:* When doing exercises, users may generate descriptive sentences with structure errors. The application will help users to understand errors by using both textual and graphical displays that inform users exactly where, why and how to fix errors when users generate FSCL sentences.

4. *Teach users the FSCL query engine:* The application organises matching conditions between query and descriptive sentences into matching lessons. It also sets up pre-defined query sentences to illustrate matching conditions. These conditions are separated into group word matching and sentence matching. The application teaches users matching conditions by presenting matching lessons using extensive help from graphical presentations that display matching patterns between query and descriptive trees.
5. *Allow users to do different exercises to understand the language's query engine:* The application allows users to generate/modify different FSCL queries (by graphical and textual inputs) and to view the returned results. The application then presents descriptive sentences that match the input queries and highlights the matching patterns.

The VISL application is divided into four modes: the passive and active modes for teaching FSCL's structure, and the passive and active modes for teaching FSCL's query mechanisms. After going through four modes, users will have obtained the three correlated goals: create a good vocabulary, produce findable sentences, and build suitable queries to obtain correct returns.

7.3 Future work

In this research, teaching concepts have been developed and have been partly applied in an application for teaching a constructed language. At the current stage, the implementation in the VISL system is not completed. In addition, the concepts of this research can be further expanded so they can be used for other purposes. This section discusses the possible improvements in the VISL application, the necessity for user testing and the possibility of further development of the research concepts.

Further development of VISL

VISL offers sufficient features for teaching FSCL structures and the query engine. However, further work is needed to provide more functions and to improve usability of the system:

- The application includes four modes: the first and the second are for teaching FSCL structure for information description and the third and fourth are for query engine for information retrieval. In the current stage, only the first and the second modes are fully implemented. In the third and the four modes, matching patterns between query and descriptive trees are implemented but the links between explanations and diagram illustrations are still required.
- In the current development, users have to enter individual words and then drag them together to create a word group when they want to build a structure tree. This process can be slow. To let users build structure trees faster, the application should allow users to enter word groups in the form of textual input. These word groups could then be placed in the drawing space with a simple action. This would simplify the drawing process of structure trees.
- There are two possible types for displaying FSCL structure tree. One is a presentation from left-to-right and the other is a top-down layout. No study was found that claims which layout is better. In this application, only the top-down layout was implemented. A future application should be able to present sentence structures in both left-to-right and top-down layouts. Users then had the freedom to select the most suitable layout for themselves.

User testing

The VISL application is considered a graphical CALL application for teaching FSCL structures. Therefore, user testing is necessary to verify the usability of the application in general and its graphical user interface (GUI) in particular. Questions like the following could guide the user testing:

Testing for usability of the application:

- Is the application easy to learn?
- Are the FSCL's rules easy to understand?
- Are FSCL matching conditions easy to understand?
- How long it will take users to master the language by using VISL?

Testing usability of GUI:

- Is the interface easy to use?
- Is VISL's GUI consistent with the interface of PAC?

- Are the error messages easy to interpret?
- Can users easily change from one mode to another mode?

Possible extensions to the concepts underlying VISL

The learning concepts developed in this research are primarily used for teaching constructed languages. These concepts also lay foundations for possible applications used for describing and retrieving information as the following:

- In library systems that store information of books or magazines in schools or universities, data can be grouped in a hierarch structures. At the highest level, books are separated into subjects or fields. Each subject or field is then divided into publishers and books' authors. Each author may have written several books. In addition, there may be many books or magazines for the same subject. New students might not know how to use these systems to find the required books at the first time. A librarian can help them to learn how to search. However, using a simple learning tool based on the learning concepts in this research, students could more easily understand how to search for required books.
- There are a growing number of computer systems that manage, organize and retrieve multimedia data. In these systems, content-based queries are used to retrieve multimedia data. Those queries are called image queries that based on both *textual descriptions* and *image characters* such as colour (e.g. green, magenta or purple), shape content (e.g. cross, oval or half-circle), motion (e.g. upward or left-right), spatial relationships (above, left of, or row) and volume (e.g. cube, ball or toroid-for 3D objects) of images (Bouet and Djeraba, 2003 and Wielbut, 2004). The matching images are images that have "similar" characteristics as the image queries. It would be interesting to investigate how the concepts developed in this research could be applied not only to teach users the matching engine using natural words (as queries in FSCL) for *textual descriptions* but also to teach users the *image-character* queries. Users could learn this matching engine by viewing textual explanations illustrated by diagrams and by practicing to create content-based queries to learn how to select good image queries to get best matching images from multimedia databases.

- Structured Query Language (SQL) is an ANSI (American National Standards Institute) standard computer language for creating, accessing and manipulating database systems. In this language, SQL statements are used to create, retrieve and update data in a database (w3school, 2004). To use successfully the language for creating databases and retrieving information, users must clearly understand how to create databases with good structures and create proper SQL statements to achieve correct information. There are several applications that use diagrams to support users to *understand* database structure (e.g. Skilled software, 2003). In addition, several graphical applications are developed to help users to *understand* SQL statements. For example, Oracle Corporation, (1999) uses structure diagrams to verify the validation of users' SQL statements. There is still room available for computer applications that explore advantages of the combination between diagrams and active learning for teaching users to design databases with *good structure* and to generate *accurate* SQL statements.

References

Akker, J., Branch, R. M., Gustafson, K., Nieveen, N. and Plomp, T. (1999). *Design approaches and tools in education and training.* Kluwer Academic publishers.

Altova (2004). *XMLSpy* [Online] http://www.altova.com/products_ide.html. Accessed 19/03/2004

Bernstein, C. (1992). *A Consumer Guide to Charles Bernstein* [Online] http://www.poeticinhalation.com/bernstein.html Accessed 07/05/2004.

Blackwell, A. F. (1997). *Diagrams about thoughts about thoughts about diagrams.* In M. Anderson, (Eds.). Reasoning with Diagrammatic Representations II: Papers from the AAAI 1997 Fall Symposium. Menlo Park, California: AAAI Press, 77-84pp.

Boggs, W. and Boggs M. (1999). *Mastering UML with Rational Rose.* Published San Francisco, Calif. London: Financial Times Management.

Bonwell, C. C. and Eison, J. A. (1991). *Active Learning: Creating Excitement in the Classroom.* Washington, DC: George Washington University, 1991. [Online] *http://www.ntlf.com/html/lib/bib/91-9dig.htm* Accessed 13/06/2004.

Borjars, K. and Burridge, K. (2001). *Introducing English grammar.* A member of the Hodder Headdline Group London and co-published in the United States of America by Oxford University Press Inc., New York.

Bouet, M. and Djeraba, C. (1998). *Visual Content Based Retrieval in an Image Database with Relevant Feedback* [Online] http://www.cs.wayne.edu/~grosky/CSC8710/Papers/Bouet.pdf Accessed 12/03/2004.

Bruggemann-Klein, A. and Wood, D. (1996). *Drawing trees nicely with Tex* http://www.csit.fsu.edu/~mimi/tex/doc/plain/treetex/tree_doc.pdf. Accessed 12/12/2003.

BYU Faculty Center (2000). *Active Learning in College Classrooms* [Online] http://www.byu.edu/fc/pages/tchlrnpages/focusnewsletters/Focus_F00.pdf. Accessed 08/02/2005.

CALICO (1999. *Scholarly activities in computer-assisted language learning: development, pedagogical innovations and research.* Joint policy statements of CALICO, EUROCALL and IALLT Arising from a research Seminar at the University of Essen. [Online] http://calico.org/CALL_document.html. Accessed 09/10/2003.

Cameron, K. (Ed.). (1999). *CALL and Learning Community.* Exeter: Elm Bank Publications.

Canning-Wilson, C. (2001). *Visuals and Language Learning: Is There A Connection?* ELT Newsletter. [Online] http://www.eltnewsletter.com/back/Feb2001/art482001.htm. Accessed 21/09/2003.

Capital Community College Foundation, (2003). *Guide to grammar and writing* [Online]. http://webster.commnet.edu/grammar/index.htm Accessed 12/10/2003.

Chamberlin, D., Florescu, D., Robie, J., Simon, J. and Stefanescu, M. (2001). *XQuery: A query language for xml*. In W3C Working Draft, 15 February 2001. [Online]. http://www.w3.org/TR/xquery/. Accessed 17/03/2004.

Chang, S., Ichikawa, T., and Ligomenides, A. (1989). *Visual Languages.* Plenum Press

Chapelle, C. (1994). *Theoretical bases for human-computer interaction research in CALL.* CALICO '94 Conference Proceedings in Duke University. 53-57pp.

Chen, C., Meng, H. J., Sundaram, H. and Zhong, D. (1998).*A Fully Automated Content-Based Video Search Engine Supporting Spatiotemporal Queries*. IEEE Transactions on circuits and systems for video technology, Vol. 8, No. 5, 602-615pp.

Cheng, P. C-H (1996). *Functional Roles for the cognitive analysis of diagrams in problem solving.* In Proceeding of the 18th Annual conference of the cognitive Science society. 207-212pp.

Christel, M. G., Winkler, D. B., and Taylor, C. R (1997). *Improving Access to a Digital Video Library* [Online] http://www.informedia.cs.cmu.edu/documents/INTERACT97tagged.pdf. Accessed 23/04/2004

Collins, P. and Hollo, C. (2000). *English grammar an introduction.* Macmillan Press LTD.

Crowley, T., Lynch, J., Piau, J. and Siegel, J. (1995). *The design of language: an introduction to descriptive linguistics.* Auckland Longman.

Darling, C. (2003). *Diagramming Sentences.* [Online] http://webster.commnet.edu/grammar/diagrams/diagrams.stm. Accessed 26/12/2003.

Debusmann, R. (2000). *An introduction to Dependency grammar*. In Hausarbeit fur das Hauptseminar Dependenzgrammatick. SoSe 99. [Online] http://www.ps.uni-sb.de/~rade/papers/dg.pdf. Accessed 21/09/2003.

Dennis, A., Wixom, B., H. and Tegarden, D. (2002). *Systems Analysis & Design: An Object-Oriented Approach with UML.* The United States of America: John Wiley & Son, Inc.

Dougherty, R. C. (1994). *Beginner's Workbook in Computational Linguistics: Preface*. [Online] http://www.nyu.edu/pages/linguistics/acrpref.pdf. Accessed 22/10/2003.

Douglass, B. P. (2000). *Real time UML*. Massachusetts: Addison-Wesley, Inc

Dowling, C. (1999). *Writing and learning with computers* Australian Council for Educational Research.

Eastwood, J. (1999). *Oxford practice grammar: with answers*, Oxford University Press.

Edgar, J. (2000). *CALL is new*. Computer-Assisted Language Learning Interest Section Newsletter volume 18, number 1. [Online] http://www.itp.innoved.org/wiki-files/callisnews/CALLnews18-1.pdf. Accessed 03/10/2003.

Eskew, M. (2001). *What is Grammar?* [Online] http://www.tulane.edu/~germgram/whatgram.html. Accessed 24/01/2003.

Eskew, M. and Carr, A. (2001). *German grammar roadmap*. [Online] http://www.tulane.edu/~germgram/sent.html#sent4. Accessed 13/01/2003.

Exceller Comp. (2003).*Focus on Grammar Series* [Online] http://www.exceller.com/focus-on-grammar.html. Accessed 15/12/2003.

Fast, K., Leise, F., and Steckel, M. (2002). *Creating a Controlled Vocabulary* [Online] http://www.boxesandarrows.com/archives/creating_a_controlled_vocabulary.php Accessed 13/11/2003.

Feldman, S. (1999). *NLP Meets the Jabberwocky: Natural Language Processing in Information Retrieval*. [Online] http://www.onlinemag.net/OL1999/feldman5.html Accessed 12/06/2003.

Flippo, H. (2003). *German Language*. [Online] http://german.about.com/library/weekly/aa032700a.htm. Accessed 09/04/2003.

Flynn, P. (2004). *The XML FAQ* [Online] http://www.ucc.ie/xml/ Accessed 23/06/2004

French, J., Powell, A. Gey, F. and Perelman, N. (2002). *Exploiting a controlled vocabulary to improve collection selection and retrieval effectiveness*. In Proc. of CIKM '01, pages 199-206. ACM Press, Nov. 2001.199-206. [Online] http://metadata.sims.berkeley.edu/papers/cikm01-paper-with-french-powell.pdf Accessed 12/09/2003.

Fromkin, V., and Rodman, R. (1998). *An Introduction to Language*. Fort Worth: Harcourt Brace College Publishers, c1998.

Galliers, R. D. (1992). *Information systems research: issues, methods, and practical guidelines.* Henley-on-Thames, Oxfordshire: Alfred Waller publisher

Granger, S. (Eds.) (1998). *Learning English on computer.* Addison Wesley Longman Limited.

Grishman, R. (1986). *Computational linguistics, an introduction.* Cambridge University Press.

Hamilton-Taylor, A. G. and Kraemer, E. (2002). *Designing an Algorithm Animation System to Support Instructional Tasks.* Interactive Multimedia Electronic Journal of Computer-Enhanced Learning (IMEJ), Vol. 4, No. 2. [Online] http://imej.wfu.edu/articles/2002/2/04/index.asp. Accessed 17/06/2003.

Harrison, N. (1991). *How to design effective computer based training, a modular course.* McGraw-Hill Education - Europe Editor.

Härtl, H. (2003). *Generative Grammar: A Historical Perspective* [Online] http://www2.rz.hu-berlin.de/angl/haertl/haertl_history_slides_1-18.pdf. Accessed 06/05/2003.

Hausser, R. (1999). *Foundations of computational linguistics.* Springer publisher.

Heinrich, E. (1999). *A Multimedia Information System for the Support of Studies of Behaviour.* Unpublished Doctoral Thesis, Massey University, New Zealand.

Heinrich, E. and Kemp, R. (2000). *A Flexible Scheme for Representing and Retrieving Multimedia Contents in Computer-Based Educational Systems.* In Kinshuk, C. Jesshope and T. Okamoto, IWALT 2000, Los Alamitos, California, IEEE Computer Society. 213 – 214pp.

Heinrich, E., and Kemp, E. (2000). *Description and Retrieval across Multiple Media Formats.* 2nd International Workshop on Query Processing and Multimedia Issues for Distributed Systems at the DEXA2000 conference (Greenwich/London). A.M. Tjoa, R.R. Wagner and A. Al-Zobaidie (Eds.). IEEE Computer Society, Los Alamitos, California. 940-944pp.

Hilferty, J. (2002). *English syntax lecture notes* [Online] http://lingua.fil.ub.es/~hilferty/SynNotes.pdf. Accessed 23/03/2003.

Hopkins, R., and Park, C. (1993). *Instructional conditions for using dynamic visual displays: a review.* Journal of Instructional Science 21 Kluwer Academic Publisher 427-449pp.

Hudson, R. (2002). *Word grammar* [Online] http://www.phon.ucl.ac.uk/home/dick/wg.htm. Accessed 12/06/2003.

Hyerle, D. (1996). *Visual Tools for Constructing Knowledge.* [Online] Chapter 1 http://www.ascd.org/publications/books/1996hyerle/chapter1.html. Accessed 24/12/2003.

Intelligent Systems Research, (2004). *DTDChart: XML Document Structure Charts* [Online] http://www.intsysr.com/dtdchart.htm Accessed 03/06/2004.

Java (2004). *Java Media Framework API* [Online] http://java.sun.com/products/java-media/jmf/index.jsp. Accessed 21/03/2004.

JavaTech, (2004). *Java2D* [Online] http://www.particle.kth.se/~lindsey/JavaCourse/Book/Supplements/Chapter06/java2d.html Accessed 10/06/2004.

Jeffrey, P. K. (1994). *English grammar, principles and facts.* Prentice Hall Englewood Cliffs, New Jersey 07632.

Jordan, D. K. (2003). *Chinese language* [Online] http://weber.ucsd.edu/~dkjordan/chin/hbchilang-u.html Accessed 07/03/2003

Kenning, M. M. and Kenning, M. (1990). *Computers and language learning, current theory and practice.* Ellis Horwood Series in Computers and Their Applications. J Ellis Horwood Limited.

Kim, J. and Hahn, J. (2003). *Reasoning with multiple diagrams: focusing on the Cognitive Integration process.* [Online] http://hci.yonsei.ac.kr/non/e02/97-Cogsci-Reasoning_with_Multiple_Diagrams.pdf Accessed 27/04/2003.

King, M. (Eds.) (1983). *Parsing Natural Language.* Academic Press, London, England.

Klinger, W. (2003). *Effects of pictures on memory and learning* [On-line]. http://www2.ice.usp.ac.jp/wklinger/QA/articles/kiyou2000/kiyou2000.html. Accessed 04/10/2003.

Koning, H. Dormann, C. and Vliet, H. (2002). *Practical Guidelines for the Readability of IT-architecture Diagrams.* In Proceedings of the 20th annual international conference on Computer documentation. 90-99pp. [Online] http://www.cs.vu.nl/~henk/. Accessed 12/09/2003.

Kroch, A. (2003). *Tree drawing tool animation* [Online] http://www.ling.upenn.edu/~kroch/trees/demo.html Accessed 10/09/2003.

Korman, W. (2003). *Climbing the data tree* [Online] http://www.go2net.com/internet/deep/1997/05/07/body.html. Accessed 04/11/2003.

Kromann, M. T., (1998). *Visum Project* [Online] http://www.id.cbs.dk/~mtk/files/visumtree.pdf Accessed 12/01/2003.

Kromann, M. T., (2002). *Dependency grammar and local optimality parsing.* [Online] http://www.id.cbs.dk/~mtk. Accessed 02/07/2003.

Krüger, A. and Geurts, B. (1996). *RECALL.* [Online] http://www.infj.ulst.ac.uk/~recall/deliver/d5.html. Accessed 05/05/2003.

Larkin, J. H. and Simon, H. A. (1987). *Why a diagram is (sometimes) worth ten thousand words.* Cognitive Science Volume 11. 65-99pp.

Learning Center, (2004). *ACEREADER® speed reading software* [Online] http://www.certified-learning-centers.com/acereader.html Accessed 04/06/2004.

Lebanon Valley College (2000). *Free-Text vs. Controlled Vocabulary* [Online] http://www.lvc.edu/library/guides/free-controlled.html. Accessed 12/07/2003.

Leech, G. N. and Candlin, C. N. (1986). *Computer in English language Teaching and research.* Longman Group limited, London, 1986.

Lehtola, A., Tenni, J., Bounsaythip, C. and Jaaranen, K. (1999). *Controlled languages as the basis for Multilingual Catalogues on the WWW* [Online] http://www.vtt.fi/tte/language/publications/emmsec99.pdf. Accessed 06/05/2003.

Levie, W. H. (1987). *Research on Pictures: A Guide to the Literature.* In D.M. Willows and H. A. Houghton (Eds.). The Psychology of Illustration. Volume 1: Basic Research.

Levy, M. (1997). *Computer assisted language learning, context and conceptualisation.* Clarendon Press Oxford.

Lingsoft Comp. (2003). *A Short Introduction to ENGCG.* [Online] http://www.lingsoft.fi/doc/engcg/intro/. Accessed 12/09/2003.

Louden, K. C. (1997). *Compiler construction, principles and practice*. PWS Publishing Company.

Luo, M. (2000). *User Interface for FSCL/FSQL, Structure editor for novice user interface.* Honours project report. Massey University.

Luo, T. (1993). *TreeDraw: A Tree-Drawing System.* Unpublished mater thesis, Waterloo, Ontario, Canada.

Lynn, E. (2003). *Learn English – FAQs* [Online] http://www.learnenglish.de/FAQspage.htm. Accessed 12/02/2004.

Martin, J. and McClure, C. (1985). *Diagramming techniques for analysts and programmers.* Prentice Hall, INC., Englewood Cliffs.

Martinez, J. M. (2003). *MPEG-7 Overview of MPEG-7 description tools* [Online] http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm. Accessed 14/01/2004.

Massey University (2004). *Using XML tutor* [Online] http://is157250.massey.ac.nz/h/Module_2_Extras/XML_Tutor/Using_XML_Tutor_S oftware.doc. Accessed 10/9/2004.

Max, A. (2002). *Syntax student's companion* [Online] http://www-clips.imag.fr/geta/aurelien.max/SSC/frames.html. Accessed 04/01/2004.

Mayer, R. E. and Gallini, J. K. (1990). *When is an illustration worth ten thousand words?* Journal of Educational Psychology, 82, 715-726pp.

Mayer, R. E. and Sims, V. K. (1995). *For Whom is a Picture worth a thousand words? Extensions of a Dual-Coding Theory of Multimedia Learning.* Journal of Educational Psychology, 86 (3), 389-401pp.

Mayer, R. E., Steinhoff, K., Bower, G. and Mars, R. (1995). *A generative theory of textbook design: using annotated illustrations to foster meaningful learning of science text.* Educational Technology, Research and Development 43(1). 41–43pp.

McKeachie, W. J. (1998). *Teaching tips: Strategies, research and theory for for college and university teachers.* Houghton-Mifflin Higher Education publisher

Michael, G., Christel, M. G., Winkler, D, B. and Taylor, C. R., (1997). *Improving Access to a Digital Video Library.* This paper accepted for publication in Human-Computer Interaction: INTERACT97, the 6th IFIP Conference on Human-Computer Interaction. [Online] http://www.informedia.cs.cmu.edu/documents/INTERACT97tagged.pdf. Accessed 14/10/2003.

Microsoft Cop. (2003). *Natural Language Processing [Online]* http://research.microsoft.com/nlp/. Accessed 07/02/2004.

Microsoft Corporation, (2004). *XPath tutorial* [Online] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/xmmthtransformnodetoobject.asp Accessed 08/05/2005.

Mitamura, T. (1999). *Controlled language for multilingual machine translation,* [Online] http://www.lti.cs.cmu.edu/Research/Kant/PDF/MTSummit99.pdf. Accessed 05/09/2003.

Moen, S. (1990). *Drawing dynamic trees.* IEEE Software, 28. 7-21pp.

Moy, K. F. and Eliëns, A. (1994). HyCES: *a Hypermedia Chinese Education System.* Vrije Universiteit, Amsterdam [Online] http://www.cs.vu.nl/~dejavu/papers/hyces.ps.gz Accessed 15/07/2004.

Najmi, M. (2004). *XML, Platform-Independent and Well-Supported Technology [Online]* http://www.techiwarehouse.com/Articles/2002-09-02.html Accessed 10/05/2004.

Noldus Information Technology (2003). *The Observer Video-Pro*, Version 4.0 for Windows, Specifications The Observer.

Noldy, N. E., Stelmack, R. M., and Campbell, K. B. (1990). *Event-Related Potentials and Recognition Memory for Pictures and Words: The Effects of Intentional and Incidental Learning.* Psychophysiology, 27 (4), 417-428pp.

Ollershaw, A., Aidman, E., and Kidd, O. (1997). *Is an Illustration Always Worth Ten Thousand Words? Effects of Prior Knowledge, Learning Style and Multimedia Illustrations on Text Comprehension.* International journal of instructional media, 24 (3), 227-238pp.

OMG new, (1997). *What Is OMG-UML and Why Is It Important?* [Online] http://www.omg.org/news/pr97/umlprimer.html Accessed 28/06/2004.

Oppermann, R. (2002). *User-interface design. Handbook on information technologies for education and training.* Pawlowski J. M. Springer publisher.

Oracle Corporation, (1999). *Syntax diagrams* [Online] http://www.cit.uws.edu.au/docs/oracle/sqlref/ap_syntx.htm Accessed 23/06/2004.

O'reilly Media (2004). *A Technical Introduction to XML* [Online] http://www.xml.com/pub/a/98/10/guide0.html. Accessed 12/11/2004

Park, C. and Hopkins, R. (1993). *Instructional conditions for using dynamic visual displays: a review.* Instructional science 21, 427-449 pp.

Patrick, J. (1985). *The capture and analysis of behavioural events in real time.* ACM, ACM Annual Conf., Denver pp. 92-98pp. [Online] http://delivery.acm.org/10.1145/330000/320466/p92-patrick.pdf?key1=320466&key2=2117392801&coll=portal&dl=ACM&CFID=20576184&CFTOKEN=82528375. Accessed 24/03/2004.

Pennington, M. C. (1993). *Computer-assisted writing on a principled basis: the case against computer-assisted text analysis for non-proficient writers.* Journal of language and education, vol.7, No.1. 43-59pp.

Penoff, B. and Brew, C. (2003). *TREX-Q: A query language based on XML Scheme* [Online] http://xml.coverpages.org/PenoffIRCS.pdf. Accessed 11/11/2003.

Pérez, R. E. (2004). *A New Algorithm for Learning From Examples* [Online] http://dimacs.rutgers.edu/~edperez/ Accessed 08/02/2005

Perry, G. (1993). *C By Example*. Que Publisher

Phillips, C. (2003). *Teaching Syntax with Trees* [Online] http://www.ling.upenn.edu/~kroch/trees/glot-review.html. Accessed 10/06/2003.

Poling, D. (2000). *Issues and Suggestions* [Online] http://www.xerlin.org/listarchives/xerlin/2000/0001.html Accessed 09/02/2003.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. and Carey, T. (1994). *Human-Computer Interaction.* Wokingham, UK: Addison-Wesley publisher.

Pulman, S. G. (1996). *Controlled language for knowledge Representation.* Proceeding of the first international workshop on controlled language applications, CLAW96, 233-242pp.

Radford, A. (1997). *Syntax: a minimalist introduction.* Published Cambridge University Press.

ReadingSoft, (2004). *Speed reading software* [Online] http://www.readingsoft.com/freader.html Accessed 25/05/2004.

Reinghold, E. M. and Tilford, J. S. (1981). *Tidier Drawings of Trees.* IEEE Transactions on Software Engineering, Vol. Se-7, No.2.

Rogers, E. A. (1995). *Cognitive Theory of Visual Interaction.* In Diagrammatic Reasoning: Cognitive and Computational Perspectives, Glasgow, J., Narayanan, H.N. and Chandrasekaran, B. (Eds.). AAAI Press, 481-500pp.

Rosé, E. (2003). *C++ objects to manage and draw trees dynamically.* MacTech magazine [Online] http://www.mactech.com/articles/mactech/Vol.11/11.04/CplusplusTrees/. Accessed 05/10/2003.

Szabo, M. and Kanuka, H. (1998). *Effects of violating screen design principles of balance, unity, and focus on recall learning, study time, and completion rates.* Journal of Educational Multimedia and Hypermedia, 8 (1), 23-42pp.

Schallert, D. L. (1980). *The Role of Illustrations in Reading Comprehension.* In Rand J. Spiro, Bertram C. Bruce, and William F. Brewer (Eds.), Theoretical Issues in Reading Comprehension. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Schlieder, T. (2003). *ApproXQL: Design and Implementation of an Approximate Pattern Matching Language for XML* [Online] http://www.inf.fu-berlin.de/inst/ag-db/publications/2001/report-B-01-02.pdf Accessed 06/09/2003.

Schwitter, R., Ljungberg, A. and Hood, D. (2002). *Ecole: A look-ahead editor for a controlled language* [Online] http://www.comp.mq.edu.au/~rolfs/papers/CLAW03-ECOLE.pdf. Accessed 04/01/2004.

Shah, V., Sivitanides, M., and Martin, R. (1997). *Pitfalls of Object-Oriented development* [Online] http://www.westga.edu/~bquest/1997/object.html. Accessed 10/06/2004.

Shoebottom, P. (2001). *How to learn grammar* [Online] http://www.fis.edu/eslweb/esl/students/teanotes/gram.htm Accessed 23/09/2003.

Shu, N. C. (1986). *Visual programming languages: a perspective and a dimensional analysis, in Visual Languages*. Chang, S.-K., Ichikawa, T., Ligomenides P. A (Eds.), Plenum publishing corporation.

Sikora, T. (2001). *The MPEG-7 Visual Standard for Content Description—An Overview.* IEEE Transactions on circuits and systems for video technology, VOL. 11, No. 6. 696-702pp.

Skilled software, (2003). *SQL diagrams* [Online] http://www.skilledsoftware.com/sqldiagrams.htm Accessed 10/06/2004.

Smeaton, A. F. (1997). *Information Retrieval: Still Butting Heads with Natural Language Processing?* [Online] Pazienza, M.T. (Ed.), Information Extraction, A Multidisciplinary Approach to an Emerging. Information Technology, Frascati, Italy. http://gunther.smeal.psu.edu/smeaton97information.html Accessed 19/03/2004

Stauffer, T. (2004*). HTML By Example* [Online] http://docs.rinet.ru/EtoHTML/ Accessed 12/11/2004

Steinberg, E. R. (1991). *Computer-assisted instruction: a synthesis of theory, practice, and technology.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Stern, L. and Naish, L. (2002). *Animating Recursive Algorithms in Interactive Multimedia.* Electronic journal of computer enhanced learning Volume 4, Number 2. [Online] http://imej.wfu.edu/articles/2002/2/02/index.asp. Accessed 04/01/2004.

Sun Microsystems 1, (2004). *Java Media Framework API (JMF)* [Online] http://java.sun.com/products/java-media/jmf/index.jsp Accessed 5/06/2004.

Sun Microsystems 2, (2004). *What Is an Interface?* [Online] *http://java.sun.com/docs/books/tutorial/java/interpack/interfaceDef.html* Accessed 2/07/2004.

Syddansk Universitet (2003). *About VISL.* [Online] http://visl.hum.sdu.dk/visl/about/ Accessed 05/01/2003.

Szabo, M. and Kanuka, H. (1999). *Effects of Violating Screen Design Principles of Balance, Unity, and Focus on Recall Learning, Study Time, and Completion Rates.* Journal of Educational Multimedia and Hypermedia 8(1), 23-42pp.

Tapanainen, P. (2003). *Dependency Diagram* [Online] http://www.ling.helsinki.fi/~tapanain/dg/doc/TR-1/node41.html. Accessed 14/04/2003.

TechTarget, (2004). *Rational Rose* [Online] *http://searchvb.techtarget.com/sDefinition/0,,sid8_gci516025,00.html* Accessed 31/06/2004.

The Onestop Magazine (2003). *Teaching Teenagers Grammar* [Online] http://www.onestopenglish.com/News/Magazine/children/teaching_grammar.htm Accessed 14/12/2003.

Thomas, L. (1993). *Beginning syntax*. Blackwell Publishers.

Thomas, P. K. and Muriel, R. S. (1996). *Analysing English grammar*. Pearson Longman publisher.

TOSCA Research Group, (2003). *TOSCA Tree Editor* [Online] http://lands.let.kun.nl/TSpublic/tosca/te_elaborate.html Accessed 24/08/2003.

Tschichold, C. (2003). *Focus on grammar software.* [Online] http://www-writing.berkeley.edu/chorus/call/reviews/focusongrammar/index.html. Accessed 21/3/2003.

Ullman, D., Wood, S. and Craig, D. (1990). *The Importance of Drawing in the Mechanical Design Process.* Computer & Graphics Vol.14, No. 2, pp. 263-274, 1990 [Online] http://my.fit.edu/~swood/drwg.html. Accessed 10/04/2004.

Valin, R. D. V. (2001). *An introduction to syntax*. Cambridge University press.

Verspoor, M. and Sauter, K. (2000). *English Sentence Analysis, An introductory course.* Jojn Benjamins Publishing Company.

VIA, (2003). *Visualized Interactive Analysis grammar training program* [Online] http://progresso.dk/via/. Accessed 17/10/2003.

Voss, D. (2001). *Upstream-video searching* [Online] http://www.technologyreview.com/articles/upstream0701.asp Accessed 17/09/2003.

W3C(1), (2004). *XML Schema* [Online] http://www.w3.org/XML/Schema Accessed 12/06/2004.

W3C(2). (2004). *Introduction to XSLT* [Online] *http://www.w3schools.com/xsl/xsl_intro.asp* Accessed 20/06/2004.

W3Schools, (2004). *Introduction to SQL* [Online] http://www.w3schools.com/sql/sql_intro.asp Accessed 15/06/2004.

Walker, II., J. Q. (1990). *A node-positioning algorithm for general trees*. Softw. Pract. Exp., 20(7). 685-705pp.

Walsh, A. E. (2001). *XHTML: Example by example.* Upper Saddle River, N.J. London: Prentice Hall PTR Publisher.

Wagner, R and Mansfield, R. (2003). *XML All-in-One Desk Reference For Dummies.* New York, Hungry Minds - Wiley publisher

Warschauer, M. and Kern, R. (2000). *Network-based Language Teaching: Concepts and Practice.* Cambridge: Cambridge University Press.

Wielbut, V. (2004). *Image based content* [Online] http://sunsite.berkeley.edu/Imaging/Databases/Fall95papers/vlad2.html Accessed 10/3/2004.

Wilkinson, I. A. G. and Townsend, M. A. R (1999). *From Rata to Rimu: Grouping for instruction in "best practice" New Zealand classrooms.* The Reading Teacher.

Winn, B. (1987). *Chart, Graphs and Diagrams in Educational materials. The Psychology of illustration.* Volume 1: Basic research. New York, Springer-Verlag publisher.

Wong, S. H. S. K. (2001). *Lexical Functional Grammar -- a brief introduction* [Online] http://www.fi.muni.cz/usr/wong/teaching/mt/notes/node15.html.iso-8859-1#SECTION00051200000000000000. Accessed 04/11/2003.

WordNet, Princeton University (2004). *A lexical database for the English language* [Online] http://wordnet.princeton.edu/ Accessed 07/02/2004.

Yang, S. (2004). *Research Interest: Internet Spiders* [Online] http://www.csis.hku.hk/~yang/spider.html Accessed 19/06/2004.

Ye Hedge School, (2003). *Resources for Diagramming* [Online] http://hedgeschool.homestead.com/diagrams.html. Accessed 12/11/2003.

Yin, K. R. (1994). *Case study research: design and methods.* Thousand Oaks: Sage Publications