A COMPREHENSIVE CHINESE THESAURUS SYSTEM

By

Chen Hong Yi

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Department of Computer Science

The Chinese University of Hong Hong

June 1995

Abstract

This thesis discusses the design of a thesaurus building and maintaining system. This system is referred to as **TheSys**, which stands for **The**saurus **Sys**tem. The design objectives of the system are:

1. To help users build and maintain the thesaurus according to their own requirements.

2. To make the system used in any specialty field rather than targeting for a particular specialty field.

This goal is achieved by two features of the system. The first is the modular design for the system architecture. The second is, for the thesauri built from the system, the powerful ability to capture various semantic relationships.

The architecture of the system is designed in a modular way. In the most inner level is a thesaurus. On its surface, there are a set of application program interfaces (API) through which other customized application systems (CAS) can access the thesaurus. The API and internal thesaurus are composed of an *Application Package* which the CAS can employ as a building block. This application package can be portable to any specialty field so that it makes the system can be applied comprehensively. To facilitate the use of the system, we build a Chinese user interface (UI) on the basis of API (In this sense, the UI can be regarded as a CAS too.). Such an interface supports the building work for Chinese thesaurus more easily. But it doesn't mean the system can not

serve for other language thesaurus construction at all. Actually, other language application, such as a Japanese application system, can employ the application package for helping to build its own language thesaurus.

In TheSys, we propose a thesaurus model which provide sufficient structures to represent different semantic relationships among concepts and terms. The policy is to allow users to specify arbitrary binary relationships they want the thesaurus to capture.

As the knowledge information captured in a thesaurus can be very large, we build the semantic relationships only among a set of representatives of concepts, which is referred to as *semantemes*. Since thesaurus provide a mapping from a semanteme to a group of synonyms, which are referred to as *entry terms*, the relationships among entry terms can be deduced from the relationships among corresponding semantemes. Also, the structures representing relationships among semantemes are referred to as the *thesaurus frame*, which uses weighted relationship links to represent the semantic relationships. This approach can effectively reduce the size of the thesaurus yet the intelligence of the thesaurus is not compromised.

Currently, we have used this tools to built up a Chinese thesaurus containing 10,000 entry terms among which synonymous, quasi-synonymous, antonym and part-of/composed-of relationships are embedded. The experiment illustrates that this tool is helpful for construction of a thesaurus.

Acknowledgement

I would like to express my deepest gratitude to my research advisor, Dr. Chin Lu. Her constant encouragement and advices contributed a great deal in this research work.

I would also like to thank Mr. Kin-Hong Lee and Dr. Chi-Hung Chi for marking my thesis. Mr. Lee also gave much constructive idea in this research. Special thanks are presented to the external examiner, Professor Sun Yu Fang of Chinese Academic Institute.

I am also thankful to Mr. Chi-Wai Chan. He help me use the Motif platform to build up the system.

Finally, I am also grateful to all the friends I met in the Department of Computer Science, The Chinese University of Hong Kong. Their encouragement, companions and help made my life in CUHK delightful.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A thesaurus, by definition in the *Oxford Dictionary Of Current English*[1], is a dictionary of words and phrases grouped together according to similarities in their meanings. Users can make use of a thesaurus to check out which word/phrase is conceptually similar to which word/phrase or whether there exists synonymous relationship between one word/phrase and another word/phrase. In other word, a thesaurus captures and reveals the knowledge information of semantic relationships, in terms of synonymous relationships, among tens and thousands of different words/phrases.

Thesauri books (dictionaries) sold in the bookstores usually include synonyms, quasi-synonyms and antonyms only. In modern time, a thesaurus can be computerized as an *electronic thesaurus* [2], which is possible to capture other relationships that may reveal more semantic relations among terms, such as hierarchical relationships. (thereafter, the term "thesaurus" in this thesis means electronic thesaurus unless stated otherwise). A thesaurus may be applied directly as well as employed by other information processing systems for helping exploit knowledge on semantic relationships [3]. For example, it could be used independently for preliminary studies on the structure of different subject domains. It could be used in a word (text) processor for helping authors to enrich their vocabulary. Also, it is usually embedded in a document analysis system or a text retrieval system, to help with indexing documents and searching for

documents more intelligently [4] [5] [6] [7] [8].

The construction of thesaurus is a very kind of intensive work [2]. However, in this field, there lacks of a helpful thesaurus building and maintaining tool for users to construct their own thesaurus. The construction works on the majority of existing thesauri, which are for western languages, such as English and French, are dependent on the development of the application systems which they support [5] [6] [7]. This made the thesauri so *ad hoc.* due to the following reasons,

1. Although these thesauri can capture particular types of semantic relationships well, they are in short of providing the flexibility to extend relation types according to users' requirements.

2. Since these thesauri are built inside the application systems, they haven't public interfaces for external access, making them difficult to be ported to other application environments.

Thus, not many users can rely on this way to get a suitable thesaurus at their requests. Moreover, most of the thesaurus end users, which can best understand their own needs, hope that they can build and maintain the thesaurus in a straight forward way without involvement into its internal structures. That is they expect a thesaurus is built from a tool and they can use this tool to maintain the thesaurus in return. In addition, they even want to use this tool to construct their own thesaurus and combine it with the tool as a building block portable to other application systems. This situation motivates our project on developing of such a tool.

In this thesis, we present a thesaurus system, referred to as *TheSys* [1], which is a tool for users to build and maintain thesauri according to their own requirements. Our goal is to design a comprehensive thesaurus building tool which can be used in any specialty field rather than targeting for a particular specialty

[1] pronounce ['θisis]

field. It is achieved by two features of the system. One is the high usability of the system. The other is the capability of capturing various semantic relationships for the thesauri built from the system.

We know that a thesaurus could be used independently as well as employed by other application systems very often. Thus, the architecture of the system is designed in a kind of three-level way.

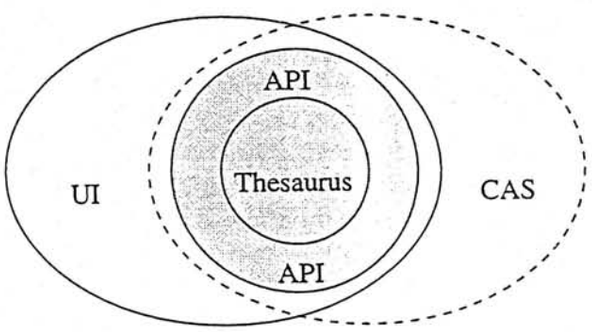

Figure 1.1: Relationships Among Thesaurus, API, Ui and CAS

The most inner level is a thesaurus. The middle level is a set of application program interfaces (API) through which other customized application systems (CAS) can access the thesaurus. The API and internal thesaurus are composed of an *application package* which the CAS can employ as a building block. This way, the internal structures of the thesaurus are hidden and the thesaurus is portable. The most outer level is a friendly window-based user interface (UI) which is built on the basis of API. It is for the system being used independently. As CAS, the UI can not directly access the thesaurus data model but through the API. (In this sense, the UI can be regarded as a CAS too.) Such a scheme makes the inner thesaurus independent from the UI which is often customized to local environment, such as language environment. Therefore, the thesaurus which is built through the UI in one environment could be ported to a new environment other than the former one. The relationships among the Thesaurus,

API, UI, and CAS is illustrate in Figure 1.1. In this figure, the grey part is the application package.

Since the building of a thesaurus is always rooted in particular language environment, the language factor make effect on the design of the system. For example, some special features of the UI, such as Chinese input method for Chinese environment, need to be customized to meet with different language environments. However, the comprehensive application of the system in different environments, including language environment, is still our goal. At this point, the architecture of the system supports this goal again. In *TheSys*, the application package is designed in a language independent way, and the UI is responsible to dealing with different language environments. Through the CAS, of which the UI is one kind, users can employ the system for various language thesaurus building work. Thus, the system still can be regarded as a comprehensive system. In our design, we develop a Chinese UI for the *TheSys*. This is because Chinese is one of the most popular languages in the world, a Chinese thesaurus building tool is needed broadly. As a product, *TheSys* is classified as Chinese application system due to that its Chinese user interface facilitates the Chinese thesaurus building work. But it doesn't mean that it can just used for Chinese at all. Actually, other language application, such as a Japanese application system, can employ the application package for helping build its own language thesaurus.

The effectiveness of a thesaurus depends on its ability to capture various type of semantic relationships. Thus, any thesaurus built from *TheSys* must provide sufficient structures to represent different type of semantic relationships among concepts and terms. In fact, each of them not only provides predefined relation types of synonyms, quasi-synonyms, antonyms, and broader/narrower terms, but also lets thesaurus builders specify other application specific relation types, such as "related-to" relationships.

As the number of synonyms for a given term can be very large, it is inefficient to build relationship links with all the terms in the thesaurus. Instead, we call all synonyms of a concept as *entry terms* and choose one among the synonyms as the *semanteme* and build relationship links among only the semantemes. The structures representing relationships among semantemes are referred to as the *thesaurus frame*, which uses weighted relationship links to represent the semantic relationships. This approach can effectively reduce the size of the thesaurus yet the intelligence of the thesaurus is not compromised.

Every entry term in a thesaurus has a unique meaning, otherwise ambiguous words exist. In *TheSys*, we use a *semantic classification tree* to eliminate ambiguities in case a term may carry multiple meanings. The semantic classification tree is strictly a hierarchical tree with each node representing one semantic classification. It could be defined by users in line with their own application environments provided that when an ambiguous term is classified under a particular semantic classification, its meaning become unique.

The rest of this paper is organized as follows, Chapter 2 discuss the background information and thesis scope. Chapter 3 presents the overall system design principles. The system architecture and thesaurus data model will be addressed. Chapter 4 describes some implementation details. It includes the data structures of the thesaurus and functional modules of the system. Chapter 5 is the conclusion and future works. In the end, three appendixes will give out a user manual of this system.

Chapter 2

Background Information And Thesis Scope

As we want to develop a thesaurus building tool, we must first understand the structures of a thesaurus. In practice, there are various thesaurus structures proposed. By analyzing these structures' characteristics, we can conceive our scheme. However, before that, we must have some consolidate background knowledge in this field, such as the basic concepts and terminologies. Also, since this tool could be applied in Chinese processing environments, we also need to know some related knowledge about the processing of Chinese information. On this background, we can clarify our research scope and go ahead to the system design.

2.1 Basic Concepts and Terminologies

The field on thesaurus construction has its own concepts and terminologies. It includes the word/term, the concepts, the semantic classification of word, the relationships, and the semantic closeness.

2.1.1 Semantic Classification Of A Word

In thesaurus building, *Word* and *Concept* are a pair of closely related terminologies. Every word can be linked to a particular concept according to its

semantical meaning. A concept is usually represented by one of the words being linked. Words being linked to the same concept are called *Synonyms*. For example, in Chinese legal field, three words 律師, 大狀and 師爺 have the same meaning "lawyer", they are synonyms and could be linked to a concept 律師. However, due to the ambiguity of the words, i.e. one word may take with different meaning in different context, there is no one-to-one relationship between words and concepts. Referred to the above example, 師爺 has the meaning of "lawyer" in legal field, but it has another meaning of "adviser" in military affairs. At this point, we can observe that a word can be classified semantically for reducing its ambiguity in some degree [23]. This classification scheme relys on the specific attributes of the words which are referred to as *Semantic Classifications* of the words. The attributes can be grammatical classifications of the words, such as noun, verb and adjective etc, or the subject classifications of the words in particular applications, such as the subject catalogs in library.

2.1.2 Relationship Link And Relationship Type

A thesaurus is used to reveal three types of conceptual/semantic *relationships*:

1. relationships among concepts;
2. relationships among words;
3. relationships between concepts and words which indicate the mapping from words to concepts or vice versa.

All these are binary relationships and each of them belongs to a specific *Relationship Type*, such as synonymous, quasi-synonymous, broader/narrower, or other related-to relationship types. Every relationship type has a unique name and some attributes describing the relationship types, such as the *Traversal Direction*. For example, quasi-synonymous relationship is symmetrical while

broader relationship is asymmetrical. Logically, when one relationship embedded on two objects(words or concepts), we say there is a *Relationship Link* being connected between the objects which are called *nodes*. It is obvious that different relationship types have different relationship link types.

It is also noted that, except synonymous relationships which are strictly referred to words, other relationships built on the words could also be reflected by the same types of relationships built on the corresponding concepts into which the words are mapped. For example, in Chinese, 愉快, 快樂and 高興 correspond concept 快樂 (happy); 憂傷 , 悲哀 and 傷感 correspond concept 悲哀 (sad). The "antonym" relationship between 愉快 and 憂傷 could also be deduced from the "antonym" relationship between concept 快樂 and 悲哀. The inverse deduction also stands.

However, not every existing thesaurus includes the relationships corresponding all three conditions mentioned above. Actually, most of them only capture the relationships corresponding the second condition [4] [5] [9].

2.1.3 Semantic Closeness, Link Weight And Semantic Distance

Semantic Closeness indicates the closeness degree between two words/concepts in terms of semantic meaning. For example, 快樂 has quasi-synonymous relationships with 開朗 (optimistic) and 興高采烈 (joyful). It is readily observed that the former relationship is semantically closer than the latter one according to the evaluation of the emotional feeling degree. In thesaurus, semantic closeness is represented in two forms. One is *Link Weight* which is set on the relationship link. Another is *Semantic Distance* which indicates the closeness between two words/concepts connected by a minimum path, which is referred to as *search_path*. This path is composed of a set of relationship links. The relationships link can belong to one relation type only or different relation types. For example, the search_path about quasi-synonymous relationship just includes the

quasi-synonymous links. The search_path about broader and quasi-synonymous relationship will be a minimum path among all the possible paths including broader and quasi-synonymous links. Strictly speaking, link weight is also a kind of semantic distance whose path is just one link.

The evaluation or assignment of semantic closeness is a very kind of subjective thing, especially for the link weight. So, only a few thesaurus allows the relationships to be weighted [4].

2.1.4 Thesaurus Model And Semantic Net

In thesaurus, various relationships among concepts/words are constructed as a network of associations. This network is called *Thesaurus Model*. Another knowledge model is very similar to this one and referred frequently in thesaurus design [5] [7]. This is *Semantic Net* which is usually a more fine-grained representation of knowledge than one finds in a thesaurus. It is mainly used in Natural Language Processing and Machine Translation.

2.1.5 Thesaurus Building And Maintaining Tool

We have explained that our research motivation is due to lack of a thesaurus building and maintaining tool. In our mind, such a tool should allow user easily construct their own thesaurus without knowing the its internal structure. In addition, this tool should also provide with the flexibility to allow users define arbitrary relation types at their requests.

2.2 Chinese Information Processing

Chinese language has its own characteristics and, therefore, we should understand them before we can find a good solution for processing Chinese information. In this section, we would like to introduce some common knowledge

about the processing of Chinese information which will affect the philosophy of designing a Chinese thesauri by using *TheSys* system.

2.2.1 The Segmentation of Chinese Words

It is commonly accepted that the basic syntactic and semantic unit of Chinese language are words 詞 but not characters 字[15]. While extracting the words from English or other alphabetical language documents faces little problem, it is far more difficult in Chinese because Chinese sentence are composed with strings of characters without natural delimiters to mark words. Although some successful tools and systems have been built to deal with Chinese word segmentation [9] [10] [16] [17], there are debates but not common agreements on this issue. The most controversy one is what is and what is not a word. For example, somebody thinks 自私自利 (selfish) is a word, but others think it is not a word but a phrase composed of two words 自私(selfish) and 自利(self-interested). Needless to say, this creates considerable problems in the construction of Chinese thesaurus.

2.2.2 The Ambiguity of Chinese Words

As the same as other human language, Chinese words may carry different meanings in different context. Such phenomenon gets thesaurus construction in trouble because it is nonsense to define the relationship between two words when not knowing their exact meanings. For example, 黃色 has two meanings "yellow" and "pornographic". When building a relationship "kind-of" between 黃色 and 淺顏色 (light colour), we have implied that this word is carrying the former meaning – "yellow". However, the thesaurus builder should let both the users and system know this. So, it is one of our tasks to solve this problem in this research.

2.2.3 Multiple Chinese Character Code Set Standards

It is well known that there exist several popular Chinese character code sets, such as GB and Big5. The machine code of a Chinese word may be explained as different information in different code set. For example, code 0xB5FC in B5 represents 詞(word), in GB it represents 迭 (repetition). So, if a thesaurus built in one code set environment is used in another environment, errors maybe occur unless there exists a code converter.

2.3 Related Work

There are three common types of system models and approaches in thesaurus design , i.e. the hierarchical model, the graphic model, and the knowledge based approach [2] [4] [5] [6] [7]. Hierarchical model is designed mainly to capture broader/narrower relationships among terms. Graphic based model was introduced to capture the more complicated "related to" relationships among different terms. The knowledge based approach combines semantic net with thesaurus to assist in information processing while the semantic net provides the knowledge in specific field.

One of the earliest and the most comprehensive study in indexing languages and thesauri was done by Soergel in the 70's. [2] Soergel pointed out that a thesaurus should provide conceptual structure as well as terminological control. He also pointed out the need for different thesaurus in different subject fields as the same term can have different meanings in different subject fields. He proposed a poly-hierarchical model which combines the hierarchical model used in classification with cross-reference links to terms that do not fit the exclusive hierarchical structure. The poly-hierarchical model is designed mainly to capture the nature of terms that inherently have more than one broader terms. However, it lacks the ability to find relationships between related terms. Systems using

the hierarchical model based on classification have been developed. The Medical Subject Headings of the National Library of Medicine [13] and the Computing Reviews Classification structure (CRCS) of the ACM [14] are examples of such.

Graphic based models are later introduced to capture the more complicated "related to" relation among different terms. McMath, *et al.* [5] proposed a cognitive graphical model in which the closeness of relationships is measured by the minimum number of links between two terms. In this model, different types of relationships, such as broader terms or narrower terms, are not distinguished. In other words, all edges are treated the same. McMath argued that the shortest path between any two terms is a measure of how closely related they are semantically.

The knowledge based approach combines semantic net with thesaurus to assist in information retrieval. The semantic net provides the knowledge in a specific field. The knowledge based approach used by Rada *et al.* in [7] combines a dyspnea knowledge base with a hierarchical thesaurus to obtain distance between any two terms. The dyspnea knowledge base is a network in which terms are represented by nodes and different types of relationships such as "is-a" or "generalization" are represented by different types of edges. As pointed out by Morris, *et al.* [6], the drawback of semantic net is the words or ideas are represented by "physical closeness". Kim *et al.* [4] proposed an improvement method based on [7]. This method combines a knowledge base with a hierarchical thesaurus except that the edges in the thesaurus is weighted so that links in the thesaurus hierarchy can identify closeness quantitatively. However, the quantified weights can represent the closeness of generalization relationship only.

Chinese is one of the most commonly used languages in the world. High quality Chinese information processing (CIP) systems are extensively desired. Yet, Chinese electronic thesaurus research has only started last 10 years. Just

a few CIP systems including a thesaurus have been developed so far. The most notable two are CIRPON developed by Beijing Document Service [8] and the BitiFTRS [9] developed by Beijing Information Technology Institute. Both systems support natural language full text retrieval and both use thesauri in post–control for retrieval. However, the thesaurus frame is mostly limited to synonym constructs only. An alternate expert system approach is used in the development of ExpCIR, an intelligent Chinese IRS [10], but the rules used in such systems are so *ad hoc.* and very difficult to maintain.

Generally speaking, most current thesauri are so *ad hoc.* because they are designed for particular systems or fields. One side, they are dependent on the system which they support, making them difficult to be ported into other application systems. On the other side, although the application environments may change and the users' need may change, few systems support users tuning the thesauri to adapt to their local environments. Therefore, the extensive application of such thesauri has been limited.

2.4 Thesis Scope

The implementation of *TheSys* is done by using C programming language. Its API is conforming POSIX 1.1 so that other customized application systems can readily employ it. It also provides a graphical user interface based on the OSF/Motif 1.1. In our design, this user interface will not directly access the thesaurus but through the API. So, it can be also regarded as an application system which is independent to the internal thesaurus. However, We will not envisage other window-based system which can build a graphic interface in this approach.

As described in section 3.1, words in Chinese documents have to be segmented first. However, *TheSys* only accepts segmented words as input. Thus, any Chinese systems including *TheSys* must have its own word segmentation

module. We employ a strategy by leaving the judgment authority of the legality of a Chinese word to the thesaurus builder or systems employing it. Therefore, a employer should initially provides a word dictionary and a word existence checking routine, such as *int LookUpWord(char *word)*, to check the existence of the word. In this way, *TheSys* can guarantee the validity of all the input words/terms in order to work consistently with the local environment. If such a word dictionary not exist, any input word is legal.

Since different Chinese system may use different Chinese Code Set standard, and we didn't have a code converter by hand. We have to limit a thesaurus built in one code set environment to be used or extended in other code set environment. For example, we can not add a GB Chinese word into a thesaurus built with B5 code set.

The requirements for revealing the semantic relationship among words/concepts vary from one system to another. In this thesis, we just discuss the binary relationships between words/concepts. In addition, when evaluate the semantic distance between two words/concepts', *TheSys* just consider the relationships in on relation type. That means the semantic distance caculation for cross-relation-type relationships, such as quasi-synonymous plus broader, is not implemented yet.

Chapter 3

System Design Principles

The objective of *TheSys* is to build a tool for users to construct and maintain thesauri according to their own requirements. It provides a set of interface functions through which the customized application systems can access the internal thesauri. Also, on the basis of API, a Chinese user interface is built for facilitating the thesaurus construction. Most importantly, it provides sufficient structures and schemes for the thesauri to capture various relationships among words and concepts so that the system can be comprehensively applied in different specialty fields. Since a thesaurus can store a large volume of knowledge information, we have carefully designed its internal frame so that its size can be effectively reduce yet its intelligence is not compromised.

3.1 Application Context Of TheSys

We realize that a thesaurus could be used independently as well as included by other application systems very often. Even though such a thesaurus is an integral part of the application system it supports, if it is included, it should maintain certain degree of independence. That is, it should be a modular part of the application system. In other words, the application system should not have direct access to the thesaurus. In our design, *TheSys* provides a set of interface functions, which are *Application Program Interface* (API), for external access. Further, the API along with the thesaurus constitute an *application*

package, which can be used by other application systems that need the same thesaurus. This way, the internal structure of the thesaurus can be independent of the application system and the thesaurus is portable.

Figure 3.1. shows such a relationship between User Applications and *TheSys*.

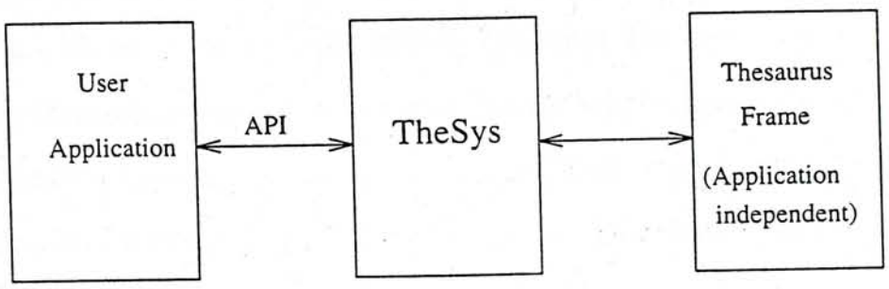

Figure 3.1: Relationship Between User Application & TheSys

3.2 Overall System Architecture

The overall architecture of *TheSys* is shown in Figure 3.2.

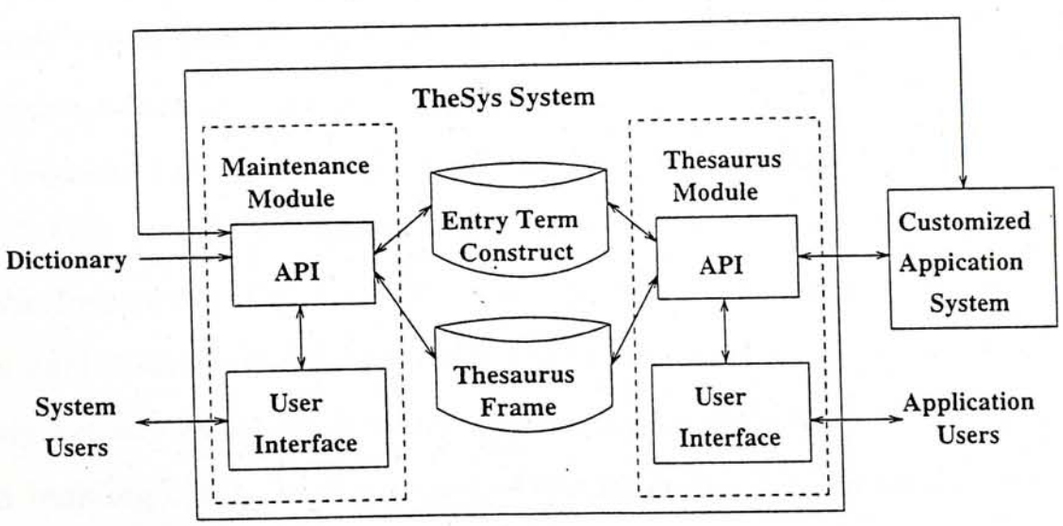

Figure 3.2: System Architecture

From Figure 3.2, we can see that the system is composed of two data components, *entry-term construct* and *thesaurus frame* which make up of a thesaurus. A thesaurus is supposed to reveal various semantic relationships among words and concepts. The key design strategy for our system is that we build the relationships on the basis of concepts. In our system, except synonymous relationships, all other relationships among concepts are represented by a graphic structure, referred to as the *thesaurus frame* where concepts are denoted by nodes and relationships are denoted by links. Each node in the thesaurus frame is referred to as *semanteme*, which is an arbitrarily chosen word among the synonyms of the same concept. On the other side, the synonymous relationships among words is modeled in the *entry-term construct*. The entry-term construct is a separate linear structure which contains all the synonyms we would like to index on, referred to as the *entry terms*. All the synonyms are stored in the entry term construct and linked together by pointers. In addition, the mapping from entry terms to semantemes is provided in the entry term construct. The system also has two main functional modules for building and querying the thesaurus: the *thesaurus module* and the *maintenance module*. The thesaurus module has an API which provides routines for all query related functions. It is the core component that uses the thesaurus frame and the entry-term construct to carry out requests from application systems. The maintenance module also has an API which is used to construct and maintain all the internal data structures of the thesaurus. The two APIs are intended for people to use them in their own application systems. Combined with the internal thesaurus, a API effectively become an *application package* which the application system can employ as a building block. In this regard, a thesaurus builder can link his customized application system (CAS) with the maintenance module alone to create a new thesaurus or to update an existing one. On the other hand, the thesaurus module can be linked to another CAS, such as a document analysis system, along

with the well-constructed thesaurus to acquire information for document analysis. The modular design makes it possible to link either one module or both with their application systems. To facilitate interactive access/update to a thesaurus, *TheSys* provides two interactive user interfaces in Chinese processing environment, one for application users (thesaurus query users) and one for system users (thesaurus builders/authors). The two interactive interfaces correspond to the thesaurus module and the maintenance module, respectively. As the CAS, they are also built on the basis of API. This scheme is also for making the thesaurus independent to the application environment into which user interface is rooted. Therefore, no matter in which way a thesaurus is built, either through a CAS or user interface, it can be easily ported to other application environments. Such a philosophy for the system architecture design is illustrated in Figure 3.3.

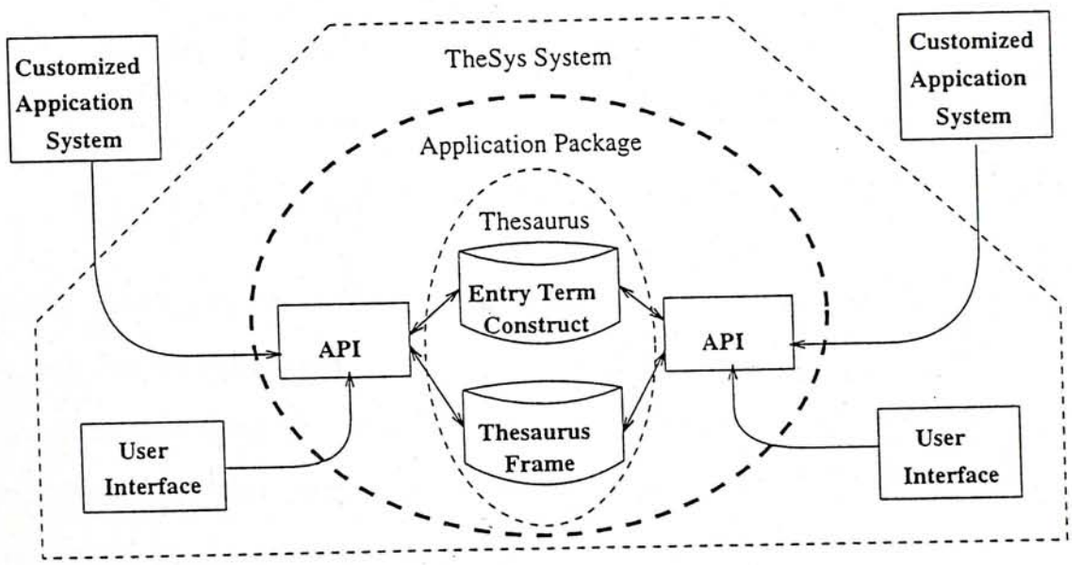

Figure 3.3: System Architecture Design Philosophy

It is noted that, as described in our above thesis scope, the system can be linked to an external dictionary for term verification if needed when building a new thesaurus. Without this dictionary, any new term entered into the entry-term construct is considered correct.

3.3 Entry-Term Construct And Thesaurus Frame

In a thesaurus, relationships are represented by binary links. When connecting terms through links, the thesaurus effectively becomes a graph. Figure 3.4 shows such a graph. In this example, we have four terms in Chinese. Among these, the term "快樂" (happy) and the term "愉快" (pleased) has a synonymous relationship. They both have a quasi-synonymous relationship with the term "興奮" (exciting) and an antonym relationship with the term "悲哀" (sad).

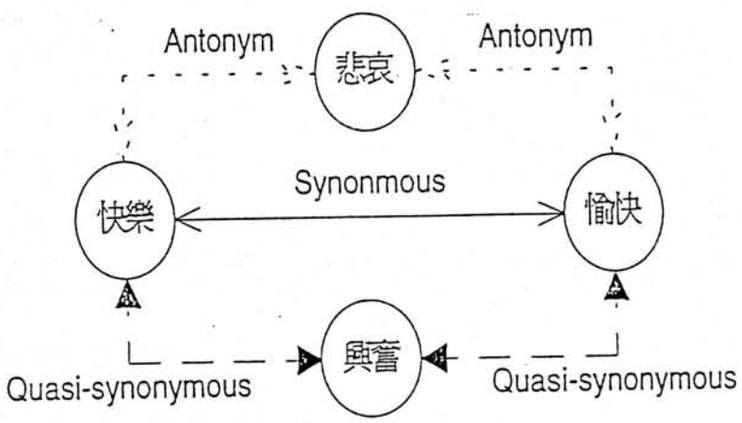

Figure 3.4: Semantic Relationships Among The Terms

For efficient management of this graph, we want to maintain minimal number of links, yet the information represented by the graphic should not be reduced. We know that one term can have several synonyms which represent the same concept. Building relationship links on the basis of terms may create many redundant links if not designed carefully.

First, consider how to link among synonyms. If a term has n number of synonyms (including itself), the synonym links can be built in two ways: (1) to build a ring of synonyms which requires exactly n links, and (2) to build a link for every pair of terms which requires C_n^2 number of links. Obviously the synonym ring yields minimal number of links. As an example, consider the term 快樂 (happy) with synonyms 愉快, 歡快, 暢快 and 高興 . The ring connection

requires only 5 links, whereas the full connection needs 10 links. The following figure illustrates such a comparison.

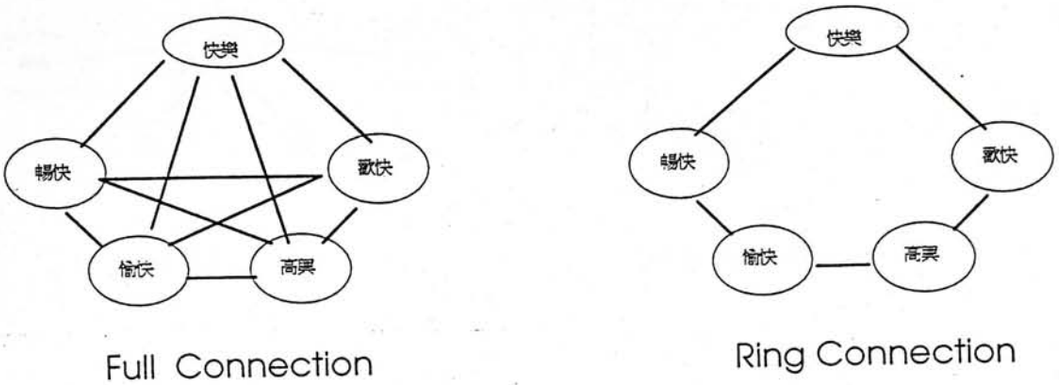

Figure 3.5: Synonyms Linking Scheme Comparison

Secondly, consider how to link terms that have relationships other than synonyms . Suppose the term sad 悲哀 is the antonym term for 快樂and all its synonyms. We can have the antonym link for 快樂 and every synonym of 快樂. This would require n links. To reduce the number of links, however, we can link with 快樂 only and use the synonym ring to obtain all the antonyms of 悲哀 and vice versa. In this case, 快樂 acts as the representatives for all its synonyms when relating to other terms in the thesaurus. Figure 3.6 illustrates such condition intuitively.

In the study of word semantics, Huang [18], He [19] and Mei [22] have found that the large volume of terms can be represented by a relatively small set of *semantemes*, which can be regarded as controlled terms or representative words. Based on this result, we build the relationship links (except for synonymous relationships which are represented by the ring structure) with semantemes only. This approach can effectively reduce the size of thesaurus yet the intelligence of the thesaurus is not compromised. In the following subsections, we will describe the details of such a scheme.

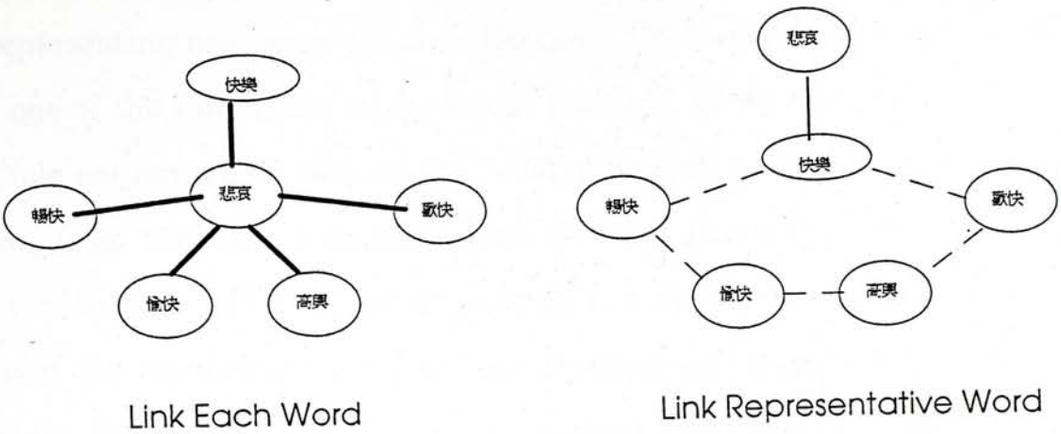

Figure 3.6: Relationship Link Building Scheme Comparison

3.3.1 Words, Entry Terms And Entry Term Construct

As described above, one of keys to our design of *TheSys* is that we build the relationship links (except for synonymous relationship links) on the basis of semantemes rather than terms/words. The semanteme will represent a group of synonymous terms/words and denoted by an arbitrarily selected term/word from the synonyms.

However, there is still one problem ahead. We have mentioned that some words may have multiple meanings. In this case, it is hard to classify an ambiguous term into any group of synonyms if we don't indicate what meaning it is carrying. In TheSys, we use a *semantic classification tree* to eliminate the term ambiguities. The main idea of this approach is that when an ambiguous term is classified under a particular semantic classification which indicates the grammatical and/or semantical attributes of the word, its meaning becomes unique [23] [22]. For example, in Chinese, a term 材料 is an ambiguous word. When it is classified under the semantic classification of 人 (human being), it means a person of quality. Whereas, if it is classified under the semantic classification of 物 (matter), it means material. A set of semantic classifications can be organized

as a semantic classification tree which is strictly a hierarchical tree with each node representing one semantic classification. The construction and use of this tree is one of the important topics in our research, so we will defer to discuss it in a whole section which address the issue of dealing with term ambiguity.

Now, from the data structure point of view, the constitution of an entry term is more clear. That is, an entry term is a synonym of a concept which is composed of a word and a semantic classification (or the semantic classification notation) . We can see that word is just a general concept. It is only a string of characters. Entry term is the concept in *TheSys*. It is a string of word characters along with a notation indicating its semantic classification. The most significant distinction between word and entry term is that word may be ambiguous in terms of meaning, but entry term must be unique in terms of meaning.

Based on this structure, a *Entry Term Construct* is built. A Entry Term Construct is a separate liner structure which contains all the entry terms that an application may want to search for along with the synonym links. All the synonyms are stored in the entry term construct and linked together. In each group of synonyms, one synonym (entry term) will be chosen as *semanteme* to represent all its synonyms. In addition, it is noted that entry term construct actually provides the synonymous relationship model.

Figure 3.7 displays the logical architecture of the entry term construct. The example is for several Chinese terms frequently used in computer science. They are the term "computer" 計算機 and its synonym 電算機, the term "database" 數據庫 and its synonym 資料庫, the term "operating system" 操作系統 and its synonym 作業系統, the term "hard disk" 硬盤 and its synonym 硬碟, the term "software" 軟件 with its synonym 軟體 , and the term "hardware" 硬件 and its synonym 硬體. All these terms are classified under a semantic classification of "computer science". Basically, in each group, the former terms are frequently used in Mainland China, the latter ones are used in Taiwan. We use a capital abbreviation CS to stand

for the semantic classification.

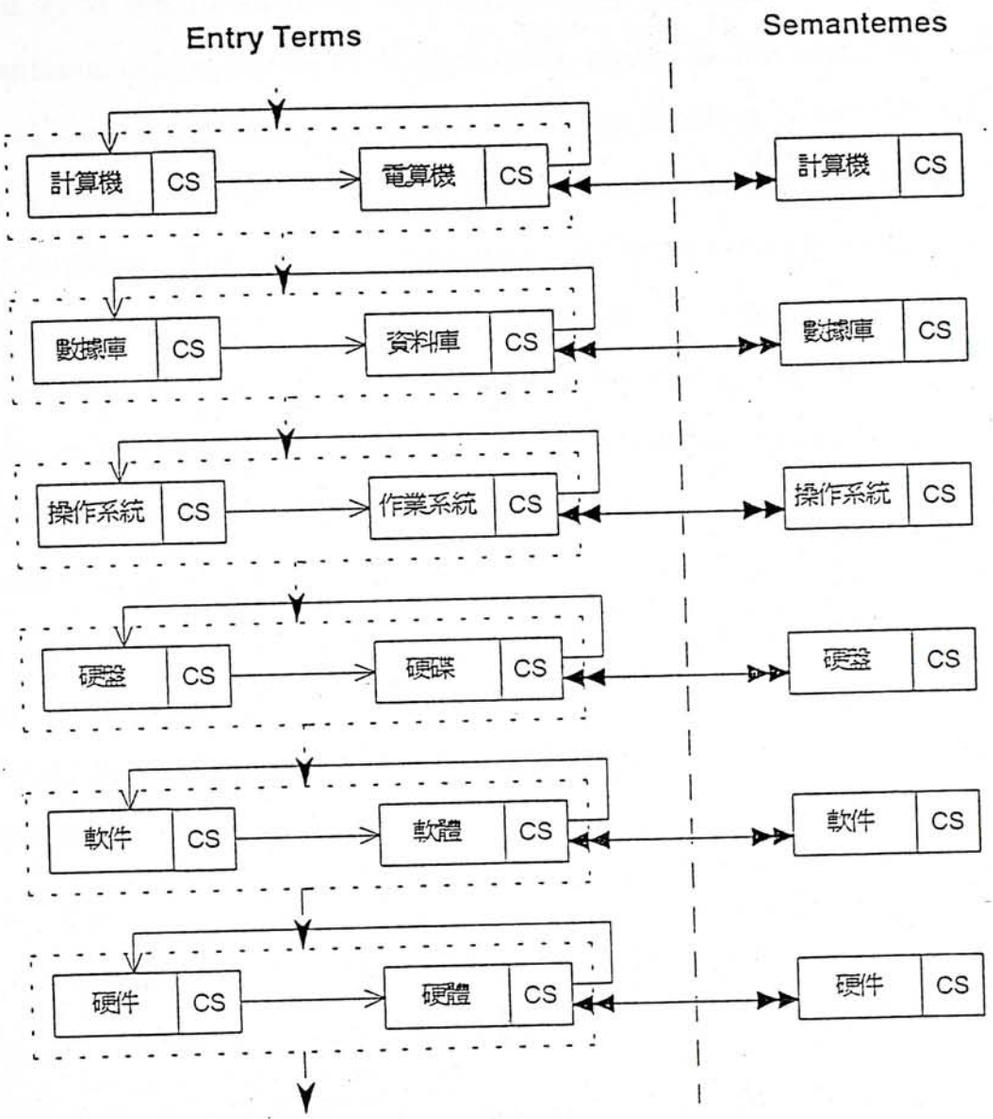

Figure 3.7: Entry Term Construct Structure

3.3.2 Semanteme, Relationship And Thesaurus Frame

In *TheSys*, semantemes and the relationships linking them are represented by a graphic structure, referred to as the *thesaurus frame* where semantemes are denoted by nodes and relationships denoted by links.

The choice of a semanteme among a set of synonyms is arbitrary since they are considered equivalent semantically. For example, in the synonym group of 數位 and 數碼, people in main land China may prefer to use the former one as semanteme, while people in Taiwan may prefer to use the latter one. No matter which one is chosen, it will not affect the function of semanteme. If one concept has only one synonym, then the semanteme is represented by this one without question. The mapping from entry terms to semantememes is provided by entry term construct, nor in thesaurus frame. This can be seen from figure 3.8. Thus, with the thesaurus frame, we can avoid building the relatively large synonymous relationship links in the thesaurus and also eliminate links that do not provide additional information. Consequently, the search speed are improved and management is easier. A logical display of the thesaurus frame is shown in Figure 3.8. (Note, every rectangle in the diagram represents a semanteme. The Chinese words within the rectangle are the chosen terms which are defined by the thesaurus builders or by system default to indicate corresponding semantemes.)

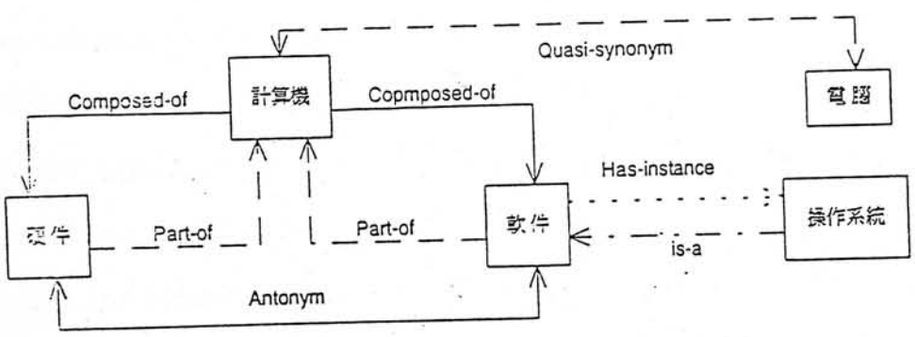

Figure 3.8: Logical Display Of Thesaurus Frame

Each type of relationship in *TheSys* is identified by its link type. Each

link type has an unique name. Traversal through this type of links can either be *symmetric* or *asymmetric*. For instance, synonymous and antonym relationships are symmetric relationship, while narrower-terms is asymmetric relationship. In some applications where quantitative measurement of semantic/conceptual closeness between terms are needed, the weight values on links are needed. Therefore, each link type must also specify its *Weight Type*, either *standard* or *weighted*. On the links with weight type as standard, the weight values are constants for all links of the same type, e.g. constant integer 1 in most of the existing thesaurus. Whereas, on the links with weight type as weighted, different links of the same type may have different values in the range of 0 to 1. For example, the weight type of "*quasi-synonym*" is weighted whereas "*antonym*" is standard.

In *TheSys*, in order to describe the conceptual closeness between two words, we introduce a weighting scheme, referred to as *Comparison-based scheme*. In our scheme, the smaller the weight value is, the more closely they are related to each other. Consequently, the weighted value between two synonymous words is 0. As to the weighted linked, *TheSys* requires users classify the weight value range symbolically. For example, users could classify the weight value from 0 to 0.3 as *closer*, from 0.3 to 0.6 as *normal*, and from 0.6 to 1 as *looser*. This is because the application user will hardly understand the meaning of a number value but a symbolic value. So, they just need to assign the symbolic value as they insert the relationship links. *TheSys* need the above information to switch the symbolic value into numeric value. This mapping is defined in a separate structure pointed by a address pointer in relation type definition record. If the weight type of the relation type is standard, this field is a *nil* value because the weight value of this type is constant and users needn't assign the weight. Since the weighting scheme is one of the important topic in this paper, we will discuss its detail in later section.

We intend to build a comprehensive thesaurus system which is able to make the thesaurus capturing various relationships effectively. The synonymous relationships has been naturally caught due to the entry term construct. The other types of relationships can also be captured easily. The procedure is to define the relation type first, then lay down each relationship link of this type. After evaluating various thesaurus model, including both Chinese and Western systems, we find that there are six relationship types that are used most frequently in various application environments. They are "*is-a*", "*has-instance*", "*part-of*", "*composed-of*", "*quasi-synonym*", and "*antonym*". Their properties are shown in the following table,

Name	Traversal Direction	Weight Type	Weight Range Pointer
is-a	asymmetrical	Weighted	(address)
has-instance	asymmetrical	Weighted	(address)
part-of	asymmetrical	Weighted	(address)
composed-of	asymmetrical	Weighted	(address)
quasi-synonym	symmetrical	Weighted	(address)
antonym	symmetrical	standard	nil

These six relationships are referred to as *built-in relationships* in our system. It means that *TheSys* has prescribed such pre-defined relation type and allow thesaurus builder to directly capture such types of relationships among semantemes. However, a specific thesaurus that does not need the pre-defined types can simply keep these definitions intact without creating any link of that type. In addition, the system is designed in a way that users can define additional relationship types, such as "related-to" relation, if application environment so requires. Actually, in our design, the definition of every relation type will be stored in a *relation type definition file* . Each record of this file contains the attributes of a relation type, such as the name (identity), the traversal direction *etc.* Therefore, users can use the routine provided in API or UI to add their own definition record into this file for extending relation types.

When a particular relation type is defined, the relationship links of this type can be added in. In *TheSys*, different type of relationship links are stored in

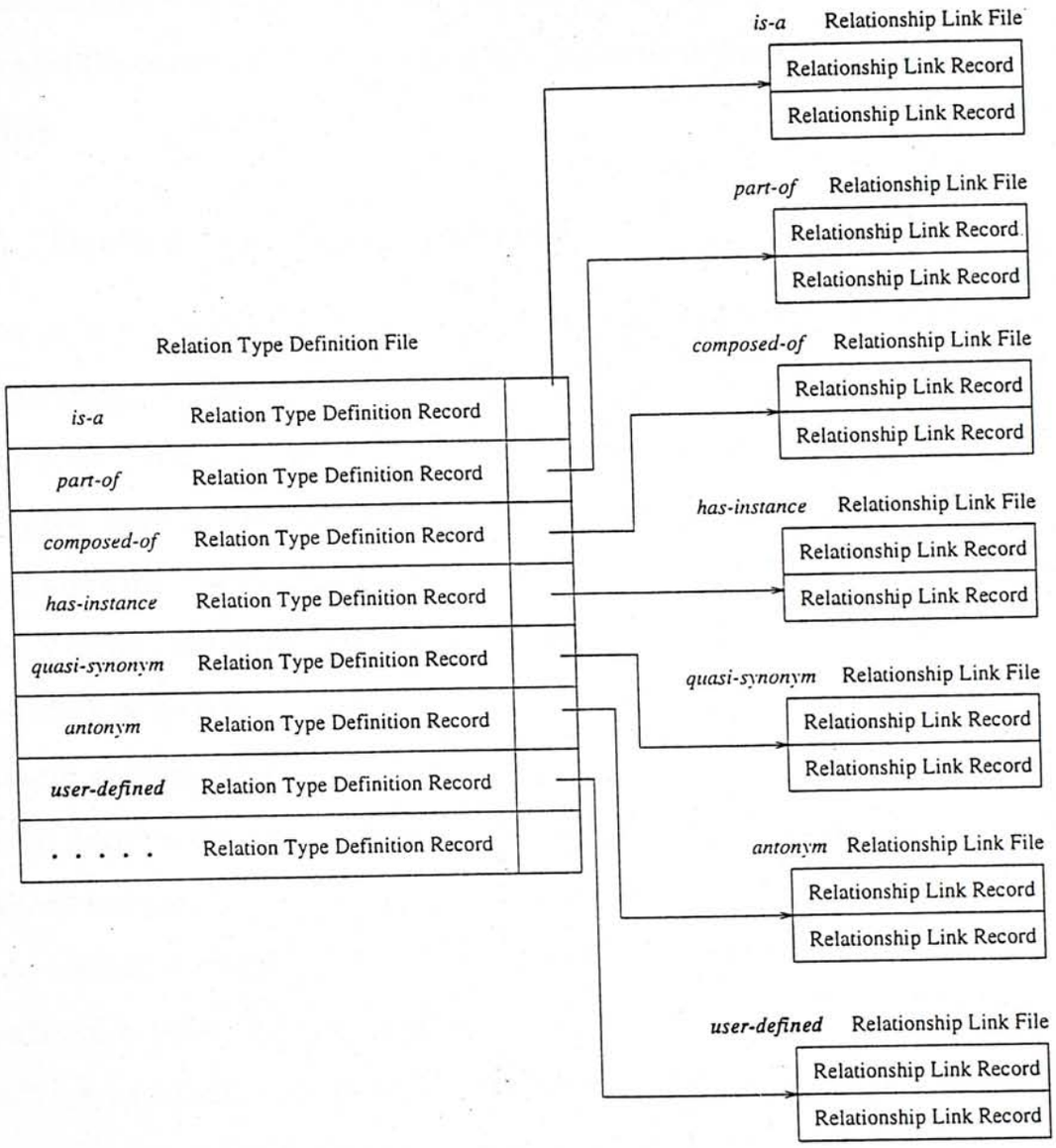

Figure 3.9: Relation Type Definition File And Relationship Link File

different *relationship link file* This way, by including different relationship link files, a well-constructed thesaurus can be ported to other applications in a flexible way. For example, a desk-top word editor may just need to include the quasi-synonym and antonym relationship link files, and a information retrieval system may include all the predefined-type relationship link files. The following figure illustrates the logical plan of the relationship type definition file and relationship link files.

3.3.3 Dealing With Term Ambiguity

Terms in real life can carry multiple meanings which are not directly related. For instance, the term "race" means "origin" , "compete", or "competition". When specifying a relationship between two terms, the relationship must be associated with a specific meaning. It does not make sense to link two terms before knowing the exact semantic meaning for an ambiguous term.

In *TheSys*, we eliminate the term ambiguities by using a *semantic classification tree*, which is a strictly hierarchical tree with each node representing one semantic classification. The main idea of this approach is that when an ambiguous term is classified under a particular semantic classification, its meaning becomes unique.

A semantic classification of a term is the grammatical or semantical attributes of a term [23] [22]. For example, "noun", "verb", "adjective" and "adverb" are semantic classifications which present the grammatical functions of terms. On the other hand, "entertainment", "sports", and "computer science" are also semantic classifications which present the usage scale of terms. For instance, the term "program" under the "entertainment" classification means "performance", whereas under the "computer science" classification means "a series of codes". A classification can be divided into sub-classification further.

For example, the classification of "computer science" can be divided into "Artificial Intelligence", "Software Engineering", "Algorithm" etc. According to the study of Mei*et al*[22], the association among semantic classifications can be organized as a hierarchical tree.

Each term with its particular meaning(s) can classified under particular semantic classification. In other word, if a term is classified under a particular semantic classification, its meaning will be limited into the scale of this semantic classification. Refer to the "race" term, if it is classified under the semantic classification of "verb" , its meaning is the same as "compete". On the other hand, if it is classified under the classification of "noun", its meaning is either "origin" or "competition". So, we can see that when a semantic classification is attached to an ambiguous term, the ambiguity of the term will be reduced in some degree. If we classified the ambiguous term in sub-classification further, its meaning becomes unique finally. For example, if we attach the classification "noun-anthropology" to the term "race", then its meaning will be clarified as "origin".

The semantic classification tree is strictly a hierarchical tree with each node representing one semantic classification. The classification represented by children nodes are sub-classifications of the classification represented by farther node. The semantic classifications stood by the leaf nodes are called *terminal semantic classification*, other semantic classifications are called *intermediate semantic classification*. As one word has a entry into this tree, i.e. a node, its meaning is consequently classified under the semantic classification which the node represented. It is obvious that if one word carries two meanings, there would be two entries in the tree with different semantic classifications associated with them. The most important point is when one word being attached to the terminal semantic classification, it will not ambiguous in the sense of meaning. This is what the incense of the construction of entry terms which is

composed of a term with a terminal semantic classification.

The construction of semantic classification tree is subject-oriented, which is up to the the users according to the application specialty field where the application is required. So, different thesaurus may have different semantic classification tree. As a comprehensive thesaurus system, *TheSys* provides a well–constructed semantic classification tree which is base on the work of a Chinese thesaurus dictionary [23] . This dictionary contains 70,000 Chinese words classified under 12 major, 94 medium and 1428 minor classification. Its classification relies both on the semantical and grammatical attributes of a word. The following figure shows the major and medium classification of this semantic clasification tree.

In *TheSys*, thesaurus builders can use a set of mini-tools to build the tree provided that any term classified under the terminal classification becomes disambiguous. Such tools includes insertion of a node, deletion of a node, update of a node and browse of the tree. However, if there won't be any ambiguous word in a special application field, such a tree need not to be created.

When a semantic classification is built up, it is possible that the ambiguity of a new ambiguous word could not be eliminated even it is classified under any of the semantic classification. Thus, a so call *Split* method is used in this case. Such an approach is shown as follows step by step,

1. When a term entered into the thesaurus, attach a particular meaning to it. If it is found that the term is currently carrying more than one meaning, just let the term with one of its meaning go into next step, and leave it with other meanings stand here for next round consideration.

2. Check if this term can be classified under a particular semantic classification as deeply as possible. This classification, which is referred to as sc, is rather than the root classification that is a dummy classification under

总　目

Figure 3.10: A General Semantic Classification For Chinese Words

which any word can be classified. The checking result will be one of the following conditions,

3.1 It can be only classified under the root classification.

3.2 It can be classified under **sc** and there is no a same term classified under **sc**.

3.3 It can be classified under **sc** and there is a same term classified under **sc**.

3. If condition 3.1 met, goto step 4. If condition 3.2 met, goto step 5. If condition 3.3 met, goto step 6.

4. Insert a semantic classification node down to the root node. goto step 7.

5. goto to step 7.

6. Substitute the **sc** with a new node **new-sc** and attach the whole sub-tree leading by **sc** to **new-sc**. Then, down to the **new-sc**, insert a new node **other-than-sc** under into which the new entered term will be classified. goto step 7.

7. If the input term has no other meaning, goto step 8, else let it carry another specific meaning, and go back step 3.

8. Continue to consider other terms.

In this procedure, the newly inserted semantic classification must be a terminal classification.

3.4 Weighting Scheme

We build various conceptual relationships on the basis of concepts to simulate the real world. However, the concepts are discrete, but the world is a continuum. A

network of concepts can never be a perfect model of the real world. Therefore, we introduce a weighting scheme to smooth the transition from one concept to another concept. Since the thesaurus frame is an association network, two relevant concepts can either be directly connected by one relationship link, or indirectly connected by more than one link which is a search path. We refer to these two concepts in former condition as *directly linked concepts*, whereas in the latter condition as *indirectly linked concepts*. The proposed weighting scheme can capture two kinds of conceptual closeness corresponding above two conditions. We will describe how such a so called *comparison-based weighting scheme* works as follows.

3.4.1 Assumption

Before discussing such a weighting scheme, we have two assumptions.

1. When a new relationship link is built, such a link connects an owner semanteme and a member semanteme. The owner semanteme may be currently connected to other member semantemes by same type of semantic relationship links. Each of these links may represent different semantic closeness. The thesaurus builder should be able to tell such difference. That is the builder can tell which pair of owner–member semanteme is closer or looser in terms of semantic closeness.

2. Based on the study of McMath, *et al.* [5] who proposed a cognitive graphical model in which the closeness of relationships is measured by the minimum number of links between two terms, we have an assumption as below,

   - There are three semantemes, **S**, E_1 and E_2.
   - **S** and E_1 is connected by a search path which has minimun number of links, **n**.

- **S** and E_2 is also connected by a search path which has minimun number of links, **m**.
- **n** > **m**.

⇒ The semantic closeness between **S** and E_1 is looser than that between **S** and E_2.

Such an assumption means that starting from a semanteme, the semantic relevancy to the lower level semantic must be looser. As for the comparison of same level semantemes, the calculation will be discussed later. If the thesaurus builder thinks that, compared with a higher level semanteme, a lower level semanteme has closer relationships to the starting semanteme, he should re-organize the thesaurus frame. That is to upgrade the level of such semanteme so that the new organization can reflect its status.

3.4.2 Quantify The Relevancy Between Two Directly Linked Concepts

In our research, every relationship link is a directed binary link which connects a owner and a member. As for assigning the weight value on each relationship link, the basic idea is that we believe a weight value can make sense only in the context of owner-members. Concretely speaking, for assigning a weight value to a new added-in link, we just need to know the relative closeness of this link compare to all other links that connect the owner and owner's other members. The smaller the weight value is, the semantically closer the member is to his owner. We have confined all weight values in the range of 0 to 1. In reality, the weight values may change as a result of relative closeness change due to the addition of new links. The necessity for supporting change of relation weight values is due to the fact that semantic closeness is in itself a fuzzy concept which is usually captured more precisely in the presence of a set of words. However, such change

is harmful to the calculation on the relevancy between two indirectly linked concepts whose linking search path goes through this owner-member context. The reason is that the calculation of relevancy of two indirectly linked concepts is related to the weight values on all the links along the path. In light of this, we decide to generally classify the *semantic closeness degree* into 5 classes, i.e. *very strong*, *strong*, *medium*, *loose*, *very loose*. Every class is assigned to a constant weight value, e.g. *very strong* is assigned to 0.2. Linguists or other experts only need to indicate the closeness degree for a new added-in link, and the value is automatically attached. In this way, the phenomenon of frequent shaking of weight values can be avoided. On the other hand, the basic principle described above is still held. Of course, adjustment is needed when we find that the relative semantic closenesses of some categories of old links are outdated.

3.4.3 Quantify The Relevancy Between Two Indirectly Linked Concepts

As for quantifying the closeness degree between two indirectly linked concepts, we need to first introduce a new concept, *semantic distance*. A semantic distance between two concepts reflects the semantic closeness or relevancy between two concepts. The semantic distance of two directly linked concepts is the weight value on this link. The computation of semantic distance of two indirectly linked concepts depends on a shortest path linking these two concepts.

Before presenting this algorithm, we would like to demonstrate a computation of semantic distance defined in Kim's system[4]. After indicating potential semantic inadequate hidden in their scheme, we will then show an effective and efficient algorithm which can overcome such a semantic shortcoming. Firstly, consider the following example shown in figure,

This diagram shows the quasi-synonym relationships among several concepts. The values on the links are weight values. Now, the system is required to reveal

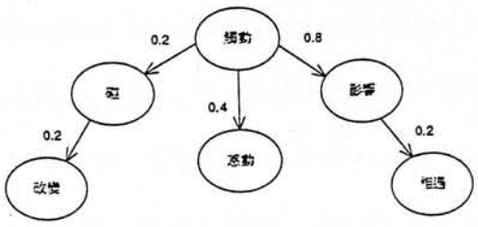

Figure 3.11: Semantic Distance Calculation

the semantic information about a word 觸動 (touch). This word is referred to as *starting concept* in terms of the revealing course. Initially, the semantic distances between the starting concept and all its neighbor concepts are calculated. The satisfactory concepts, i.e. whose semantic distance are within the specified search range, are called *activated concepts* and collected. Afterwards, the neighbor concepts of the activated concepts are activated in turn. In Kim's scheme, the semantic distance between concept 觸動 and concept 相遇 (run into) is calculated by simply accumulating the weight value of 觸動 $\longrightarrow$ 碰 (knock) and the weight values of 碰 $\longrightarrow$ 相遇. The accumulated value is 0.2+0.2= 0.4. However, the diagram shows that the semantic relevancy between concept 觸動 and 影響 (affect) is looser than that between the above two concepts because the weight value of 觸動 $\longrightarrow$ 影響 ,which is 0.8, is greater than 0.4. Such a calculation works out an obvious counter-semantic result. Semantically, the relevancy of one node with his child must be closer than that of this node with his grandchildren, or else the grandchild should be upgraded to be his child. Due to this reason, we modify the semantic distance calculation so that if the search for relevant concepts spreads out one level, i.e. the search depth or path length is increased by 1, a constant will be added to the semantic distance. In our case, this constant is 1, the upper bound of weight value scale. Such a modification guarantees that conceptual closeness of an activated concept with longer search path to the starting concept must be semantically looser.

Up till now, the comparison of semantic distances of different concept levels from a certain starting concept has been solved satisfactorily. The general idea is that the deeper the search path is, the semantically looser is the activated concept to the starting concept. But, we have not explained how to compare the relevancy of concepts on the same level. As to the above example, how do we compare the relevancy of 觸動 ⟶ 相遇 and 觸動 ⟶ 改變(change)? In this research, we introduce a *relative weight value* to reflect such relevancy. This relative weight value is subject to the location of starting concept and the search path. Every node in the search path has a relative weight value. Initially, the relative weight value of starting concept is 1. The relative weight value of one concept in the search path is calculated by multiplying its owner's relative weight value by the weight value on the link connecting its owner to itself. In the above case, the relative weight value of 相遇 is 1*0.2*0.2 = 0.04, while the relative weight value of 影響 is 1*0.8*0.7 = 0.56. According to this result, we conclude that the relevancy of 觸動 ⟶ 相遇 is semantically closer.

Generalizing the above study, we now describe the overall formulation for calculating the semantic distance as follows, (note, N_i, X_1, X_2, ..,X_n, N_j is a search path starting from concept N_i and ending at concept N_j , Weight (X_i, X_{i+1}) is the weight value on link $X_i \longrightarrow X_{i+1}$)

Semantic distance formulation:

1. Relative_Weight_Value(N_i) = 1

2. Relative_Weight_Value(X_1) = Weight(N_i,X_1)

3. Relative_Weight_Value(X_i) = Weight(X_{i-1}, X_i) * Relative_Weight_Value(X_{i-1})

4. Distance(N_i,N_j) = (search_path_length -1)*1 +
   Weight(X_n, N_j)*Relative_Weight_Value(X_n)

From this formulation, we can find that the semantic distance is mainly

determined by the former part ((search_path_length-1)*1), because the latter part is always less than 1. In this formulation, the search_path_length is n+1. If a comparison of semantic distance between two concepts required, the latter part can only make sense when the former part is the same as each other.

3.5 Term Ranking

Term ranking is a procedure to find out all the relevant words according to search type specification, such as search relation type, which can be default or specified by the user. Such a procedure is dependent on the well built thesaurus.

Once the system accepts a query including a word and the service type indicating the retrieval relation type and search depth, the process of term ranking is initiated. This algorithm can be called spreading activation which is similar to the *spreading activation* introduced in Rada's system [7], but behaves little different in the starting condition and ending condition. Our spreading activation routine starts from one concept to which the segmented word is mapped, and activates all the semantemes connecting to it in the lines of the query specification. That means only those semantemes, which link to the starting semanteme with the indicated relationship types, and with the semantic distances smaller than required, can be activated. These activated semantemes will be recorded into a set. Then, constrained by the specification, other semantemes connected to the recorded semantemes are activated in turn. When no more semanteme can be found, the routine stops. Finally, system collects all the words which correspond to the recorded concepts, and sorts out a list of words according to the priority specified by the query. The list of words can be returned to other CAS for proceeding process. As an example, we suppose the system is required to search the quasi-synonyms of word 觸動 in the semantic closeness range of 1.05. At the beginning, system works out that the Distance (觸動 ⟶ 碰) is 0.2, Distance (觸動 ⟶感動) is 0.4, and Distance (觸動⟶影響) is 0.8. So, all these three concepts

are activated and recorded. Following, the neighbor concepts of these activated concepts are considered. Distance (觸動 ⟶ 相遇)is ((2-1) + (0.2*0.2)) = 1.04, and Distance (觸動⟶ 改變) is ((2-1) + (0.2*0.8)) = 1.16. Therefore, concept 改變 is discarded. By now, no more semanteme can be activated. Consequently, all the synonyms of concepts 碰, 感動, 影響 and 相遇 will be returned as results.

3.6 Thesaurus Module and Maintenance Module

The thesaurus module provides basic access routines for all search and retrieval related functions. It is the core component that uses the thesaurus frame and entry-term construct to carry out actions requested by other applications. It takes a word plus service types, such as retrieval of synonym terms, "is-a" terms, etc. as its input and returns the relevant words as its output. These returned words reflect particular semantic information about the inputed word in line with the retrieval specification.

The maintenance module builds, updates and maintains all the internal data structure of the system. Entry term construct, thesaurus frame and semantic classification trees are built through this module. Also, as it is inevitable that query the thesaurus in the process of building and maintaining a thesaurus, maintenance module will include all the routines provided in thesaurus module. Such arrange is due to that most frequently, some system just need to employ a thesaurus with its querying functions, so separate of a thesaurus module and maintenance module is more efficient for this purpose. In our design, *TheSys* system can maintain several thesauri inside at the same time. So, whenever a user want to access a thesaurus, he must tell the system which thesaurus is to be operated. The method is giving each thesaurus a unique name and designating this name before any operation on the corresponding thesaurus. It is noted that once a thesaurus is nominated, the subsequent processes are on the this thesaurus unless user nominating another thesaurus.

3.6.1 The Procedure Of Building A Thesaurus

To build a thesaurus, users must use the maintenance module. The procedure mainly include five steps:

1. Name the thesaurus;
2. Build the semantic classification tree;
3. Define the relation type;
4. Insert entry terms and map them to the semantemes;
5. Set the relationship links among created semantemes.

As described above, every thesaurus must have a unique name as identity. This name is assigned in the first step of building a thesaurus. In a new thesaurus, the issue of whether a semantic classification is needed depends on the application environment. If all the terms that thesaurus may index on are not ambiguous, the semantic classification tree is not needed, else, users must build it up before insert the entry terms. We have described that each relationship link has a relation type. *TheSys* has provided several frequently used relation type definition. However, if users want capture other relation types, such as "associate" relationships, they must define the relation types in step 3. Actually, we can regard step 2 and step 3 as local convention description which provides a flexibility to meet different applications. After the local convention defined, users can begin to insert entry terms , map entry to semanteme and capture relationships among the semantemes. The most important point for these insertion steps is that users must select the semantemes first, and then they can lay down the relationships among them.

3.6.2 Thesaurus Nomination

TheSys allows more than one thesaurus existing in the system. However, in any time, only one thesaurus can be selected for processing. Thus, a thesaurus nomination step is needed. Actually, it is simply to pass a thesaurus name, which is a string, to the system. Both thesaurus module and maintenance module provides thesaurus nomination scheme. The initial status of the system is no nomination of any thesaurus. Users can nominate a thesaurus through four ways,

1. Name a thesaurus when start to build a new thesaurus;

2. Use the functions provided by the system to *open* a thesaurus;

Whenever the users nominate a thesaurus, the one nominated before will be closed automatically. Also, the system provides a *close* function for close the current used thesaurus and not open a new one. In addition, users can also use a function to check which thesaurus is currently being processed. That which thesaurus is active.

3.6.3 Semantic Classification Tree Construction

The semantic classification tree construction is done through maintenance module. The provided functions includes insertion of the tree nodes, deletion of the tree nodes, update the tree nodes and browse the whole tree. In most conditions, users had better make this word done before going into procedure of insertion of the entry terms. Because the change of the structure of the tree will affect the entry terms' value. For example, if the name of a semantic classification is modified from **a** to **b**, then all the entry terms which are classified under **a** have to be updated to classified under **c**. However, the system still provides such tools for meeting users' need in case the knowledge information does change. The insertion of a node is very simple that user just need to indicate the new

added node along with its farther node. Deletion of a node leads all its children node to become its farther node's children. The deletion of a terminal semantic classification is restricted unless all the entry terms attached to this classification deleted. Update of a node just do a name change work. Browse of a tree provide user online information of the tree.

3.6.4 Relation Type Definition

TheSys has six pre-defined relation type definition. User can browse their formation through a relation type browse function. In case users want to capture their own relation type, they need to make the definition first. This work is also done through the maintenance module. Basically, system needs user provide the following information,

- relation type name
- relation traversal direction type, either symmetrical or asymmetrical
- weight type, either standard or weighted
- weight rang description

This information is contained in a record structure which user need to fill it up one by one.

3.6.5 Entry Term Construct Construction

The entry term construct construction include the insertion and deletion of entry terms and update of semanteme.

The insertion of a entry term requires three arguments, a word, the semantic classification of this word, and the mapped semanteme. The former two data constitute an entry term. It is noted that system only accepts terminal semantic classification because a word classified under a terminal semantic classification

will be definitely unique in meaning. Since the mapped semanteme is also an entry term, at first sight, it should be also composed of a word and semantic classification. However, due to that the semanteme is the synonym of the inserted entry term, its semantic classification must be as the same as inserted one's. So, system just require the word of this semanteme enough. In addition, through this way, system can guarantee the entry term inserted is the synonym of the semanteme. In case a new concept collected by the thesaurus, that is no semanteme of this concept exists, the entry term will be automatically selected as the semanteme. In this case, the semantem word is itself.

The deletion of an entry term requires two arguments, a word and the semantic classification. System will delete all the word(s) under this semantic classification. Other than insertion of an entry term, the semantic classification need not to be a terminal classification. It even can be a nil value. For example, if users want to delete an entry term by indicating the word "race" and the semantic classification *nil*, then all the entry terms race(compete), race(competition) and race(origin) will be deleted. Whereas, if the semantic classification is *noun*, then only the race(competition) and race(origin) are deleted. There is a very important restriction on this operation, that is that if a semanteme is just mapped by one entry term, the deletion of this entry term is forbidden.

The update of a semanteme means select its another synonym as semanteme. It requires the old word, a new word, and their semantic classification. System will search out the semanteme and check whether the new entry term is mapped to the semanteme.

3.6.6 Thesaurus Frame Construction

Thesaurus frame construction includes insertion and deletion of the relationship link.

The insertion of a relationship link requires four arguments, relation type,

owner semanteme, member semanteme and the weight value. In case the definition of this link type indicates its weight type standard, the weight value need not to be filled. Actually, system will check the the link type definition to decide if skip this argument.

The deletion of a relationship link just requires three arguments, relation type, owner semanteme and member semanteme. These three arguments are enough to indicate a relationship link.

3.6.7 Thesaurus Query

All the thesaurus query related functions are provided in thesaurus module. They mainly divided into two groups, One is for browsing the system information, such as the existing thesauri, the semantemes, the semantic classification tree, and relation type definition. Another is for revealing the semantic relationships among words, such as getting synonyms and related words.

The browsing of system information help users know various definitions immediately. They just need to nominate the thesaurus and call the corresponding routines. The search for the synonyms of a word require two arguments, the word and its semantic classification. System will search out all the synonyms of the entry terms which are made up of by the word and the sub classifications under the given classification. Since synonyms search is frequently used, we separate this function from the search of related word. The search related word is more complicated. Users need to provide three arguments, the word, the relation type and the semantic distance scale. This search will be done on the assigned relation type, and within the search scale. It use the term rank technique discussed above. Now, we can not allow cross-relation search.

Chapter 4

System Implementation

The implementation of *TheSys* is done using C programming language. Its API is conforming with the POSIX 1.1 [25], thus it can be comprehensively ported to other platforms. Also, an interactively window-based user interface is built on the basis of API and OSF/Motif1.1.

4.1 Data Structure

In this section, we presents the internal data structure of the thesaurus. This structure is more efficient for Chinese data. However, it will not affect its ability to process other language data, such as English data. Basically, the system treat all the input data as byte strings. As the minimum processing unit of Chinese data, i.e. Character, is double byte length, all the record fields which store the data are even byte length. At this point, we can see that the English data can also fit into this structure thoroughly.

4.1.1 Entry Term Construct

Entry Term Construct basically includes three types of tables and corresponding index tables. They are *Entry Term Table*,*Entry Term Index Table*, *Synonym Table*, *Semanteme Table* and *Semanteme Index Table*.

Entry Term Table And Entry Term Index Table

Entry Term Table is used to record all the entry terms and assign an unique *word code* for every term. This word code is the internal identity of the corresponding entry term and will be cited in other data structures. Its structure is displayed as follows.

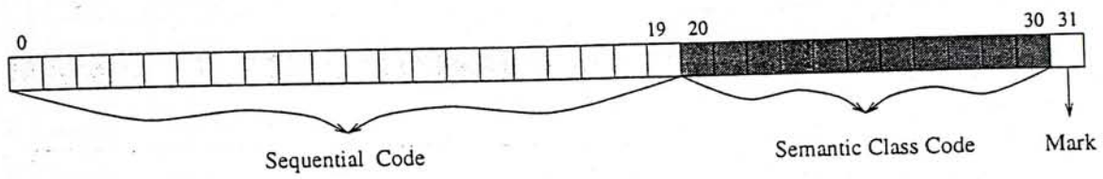

Figure 4.1: Internal Word Code Data Structure

From this logical display, we can see that internal word code is composed of three fields. The first one is *Sequential Code* which occupy 20 bits. It is the record number of the entry term. In entry term table, the number of record is maintained in the table header. Once a new entry term is inserted, system will increase the total record number by 1 and assign the new number into this field. The second field of word code is *Semantic Class Code*. It is used to represent the semantic classification of the entry term. As discussed in last chapter, users need to specify the terminal semantic classification of a new entry term. This classification is represented by a symbol. When system accepts this symbol representation, it will switch the string characters into a numerical value and assign the result to this field. This numerical value is the internal code of the semantic classification. The last field is a *Mark* bit which indicate the validity of the entry term. It is observed that the deletion of an entry term is seldom executed. Therefore, in our design, if an entry term is deleted, system just switch the value of this bit from 1 to 0 instead of replace the whole record with other entry term. This way, there is no need for packing the file so that the management is more simple and efficient even some memory is wasted. Further,

if such a entry term is inserted again, system can find out this record and directly switch the mark bit to 1 to enable the entry term validate.

The logical structure of Entry Term Table is shown in the following figure.

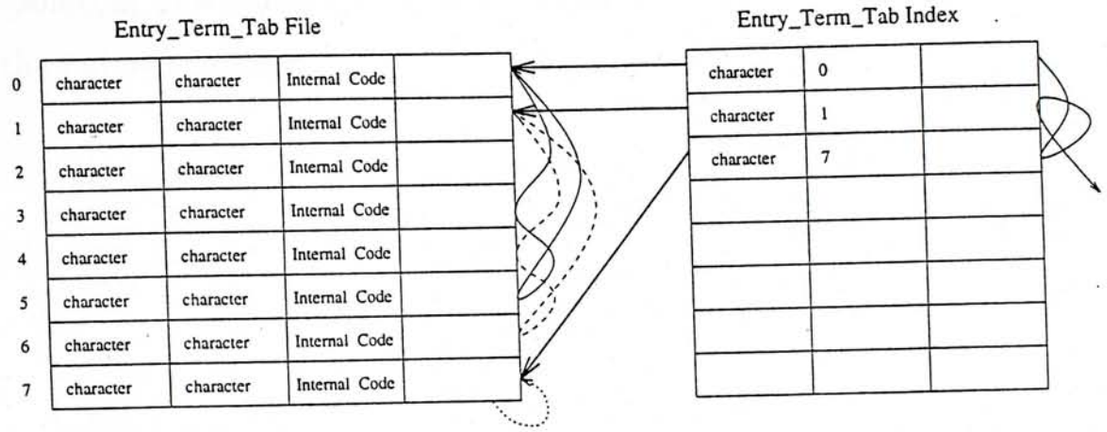

Figure 4.2: Entry Term Table Structure

In the entry term table, each column is one record which has four fields : the first two for storing the characters of the word; the third for internal word code, and the last is a pointer that points to next word which is with the same leading character. Based on the study result in [24], most of the Chinese words are one-character or two-character words. Among this, one-character words account for 12.1kinds of words can fit into this structure. For the words that consist of over two characters, system need two or more records to contain them. To recognize a word, system will check whether the internal word code of next record is the same or not. If yes, it means that the word continues on the next record, otherwise, the word is retrieved.

The entry term index table is more simple. Each record of this table contains two fields. One is an indexing character. The other is an address pointer pointing to a record in entry term table. This record stores the first entry term whose term is led by the indexing character.

Synonym Table and Semanteme Table

Synonym table is used to group the synonyms together. Semanteme Table is used to register the semantemes. There is a pointer in each semanteme record for pointing to a group synonyms in synonym table. Their structures are shown in the following figure.

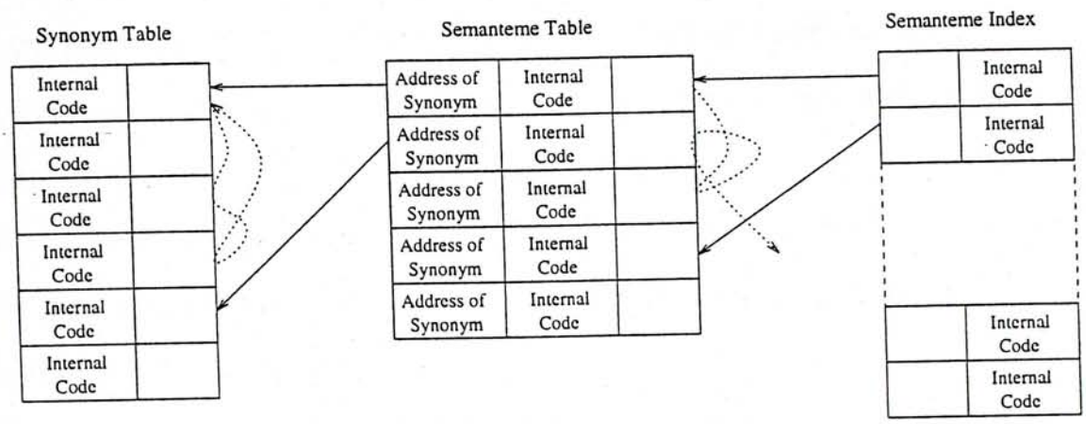

Figure 4.3: Synonym & Semanteme Table Structure

In this figure, we can see that each semanteme record contains three fields. The first is an internal word code which is the representation of the semanteme. The second is an address pointer pointing to a record in synonym table. And the last one is a pointer to next semanteme which belongs to the same semantic classification. This table is indexed by the semanteme index whose each record contains two fields. One is a class code, another is a address pointer to semanteme table. The pointed record is the first semanteme belonging to this semantic classification. The synonymous relationship among entry terms is represented in synonym table. Each record of this table has two fields. One is an word internal code representing an entry term. Another is a pointer pointing to its synonym record. In such a way, we can search from a semanteme to retrieve all the synonyms under this semanteme. Also, the mapping from semanteme to entry term is provided.

4.1.2 Thesaurus Frame

The thesaurus frame maintains one relation type definition file for the definition of link types and a set of relationship link files for recording each type of relationship links respectively.

Four fields are used to define each link type. The first field defines the name of the link type. The second field indicates link traversal direction. The third one indicate the weight type. When a thesaurus builder wants to define a relation type, quasi-synonyms for instance, he may want to assign different values for different quasi-synonyms of a given term. In case the weight type is specified as a range, the fourth field specifies the pointer to the entries in the range table where the range(s) are specified. Entries inside the relationship link definition table should not be deleted freely unless all the links of that type is deleted.

Besides synonyms, *TheSys* has defined several well known link types such as antonyms, is-a/has-instant, part-of/composed-of. A thesaurus that does not need the pre-defined types can simply keep these definitions intact without creating any link of that type. That is, initially, each relationship link file is empty until any such type of relationship link is inserted. We want to emphasize that our system provides methods to create different types of links because *TheSys* is a building tool. A specific thesaurus, however, does not need to use all of them at all.

Each relationship link of a particular link type is defined by three fields: the two semantemes and the weight value of the relationship link. Since the order of the two semantemes is important especially for a uni-directional link, each semanteme in a link is identified either as an owner or as a member with the direction pointing from the owner to the member. In case the weight is not a constant for this link type, the weight value needs to be assigned. The following figure shows the logical format of a relationship link file. Each column is one record.

Link Weight Value	Owner Semanteme	Member Semanteme
Link Weight Value	Owner Semanteme	Member Semanteme
Link Weight Value	Owner Semanteme	Member Semanteme
Link Weight Value	Owner Semanteme	Member Semanteme
Link Weight Value	Owner Semanteme	Member Semanteme

Figure 4.4: Relationship Link File

4.2 API

The API (Application Program Interface) of TheSys is conforming with the POSIX 1.1 which is an operation system interface and application environment standards. In this thesis, we won't discuss this standard because it is beyond our research scope.

As mentioned in the system design principles discussion, an application system can employ the whole API for building and querying the thesauri. However, they can also just include the query related functions for thesauri accessing. In our design, we construct the system in two functional module, one is Maintenance Module which contains all the functions, another is Thesaurus Module which only contains the query related functions.

API for thesaurus module has the following routines:

- *int ListSemanticClass (SemClassTable *table);*

  This routine returns the classification tree. The root pointer is stored in *table*.

- *RelationTypeTable *ListRelationType ( );*

This routine returns the relationship link definition *table*.

- *SemantemeGroup *SearchSemanteme (char *word, char *class);*

  This routine returns the semantemes for the *word* under classification *class*. If *class* is not specified, all the semantemes that the word is associated with under all classes are returned.

- *WordGroup *SearchEntryTerm (char *word, char *class);*

  This routine checks if the term specified by *word* is defined in the thesaurus. The semantic classification *class* is an optional parameter to indicate if the caller needs to find out whether the term exists in that particular class. If it unspecified, all the semantic classes that it belongs to are returned.

- *WordGroup *RetrieveRelatedWord (char *word, SearchScale searchscale);*

  This is a generalized routine to retrieve related terms of *word* . The precise specification of "related" is given in *searchscale*. *searchscale* contains the relationship types and with an option of a specified semantic distance. When a distance is given, *RetrieveRelatedWord* retrieve all the terms that falls within the specified distance.

- *WordGroup *SearchSynonyms (char *word, char *class);*

  For a given term *word* of the semantic class *wordclass*, this routine returns all the synonyms. If *wordclass* is not specified, synonym of all classes will be returned. This routine is actually a special case of the function *RetrieveRelatedWord()*. But it searches through the entry term construct only.

Since maintenance work may include all the related routines related to query.

So, in maintenance module, besides the above routines, the following functions are also available :

- *int InsertSemanticClass (char *parent, ClassNode *node);*

  This is the routine to build up the semantic classification tree. A new node contains its name is passed in by the variable *node*. Its parent node is indicated by the variable *parent*. The tree must be built from the root in a top down fashion by calling *InsClass()* repeatedly. The order for the sibling nodes is not importance.

- *int InsertEntryTerm (char *added, char *mapped, char *class);*

  This routine inserts a new entry term into the entry term construct. *added* contains the new term, *class* indicate the semantic classification of the new term and *mapped* provides the synonym that the new term wants to be associated with. Even though *added* can be associated with more than one synonym, only one needs to be given because of the ring structure explained in chapter 3. The newly added word will have the same semanteme as that of the mapped word. If this term does not have any synonyms, *added* itself would serve as the semanteme.

- *WordGroup *DeleteEntryTerm (char *word, char *class);*

  This routine deletes the entry term(s) specified by *word* and *class*. In case a term is also a semanteme and there are other synonyms pointing to it, the deletion is aborted.

- *int ReLinkEntryTerm (char *src, char *srcclass, char *obj, char *objclass);*

  This routine can be considered as a combination of *InsEntryTerm()* and *DelEntryTerm()*

- *int ChangeIdentifier (char *src, char *obj, char *class);*

  This routine changes the current semanteme *src* and replaces it with *obj*. Note that a term can be chosen as a semanteme only if it exists in the entry term construct. In case that an entry term needs to be deleted which happened to be a semanteme, another semanteme must be nominated. In any case, this routine is not used often because deletion in the thesaurus is rare.

- *int BuildRelationFrame (RelationTypeRecord record);*

  This routine defines a new relationship link type. The three fields for the given type is specified in *record*.

- *int AddRelationLink (char *type, LinkRecord linkrec);*

  This routine adds a new link of a specified *type*. The two semanteme and the possible weight value(s) are specified in *linkrec*.

- *int DeleteRelationLink (char *type, Semanteme S_sem, Semanteme E_sem);*

  This routine deletes the link of *type* which connects the two semantemes, *S_sem* and *E_sem*.

As discussed in the thesis scope, a scheme to check the validity of a word may be needed in some application environment. The detail of this scheme can be found in appendix1. The main idea is that user must provide an application interface as follows,

- *int LookUpWord (char *word)*

  This routine checks if the word included in a local dictionary.

4.3 User Interface

Window-based user interfaces have become a common feature of most computer systems [25]. The user interface (UI) of *TheSys* is such an interface which is developed on the OSF/Motif 1.1 platform.

4.3.1 Widget And Its Callback

Widget is a concept in Motif. It is a complex data structure that combines a X window with a set of procedures that perform actions on that window. There are two keys associated with each each Widget, resources and callbacks. Resource is a set of variables to control the appearance and behavior of the widget. For example, a widow that display a label has resources that determine the value of the label, the font used to display the label, and the margins around the label. Another key to widget is callback. A callback is a procedure that is called by a widget whenever certain events occur for the widget. The idea behind callback is that you create a procedure, and tell the widget to call that procedure whenever an event triggers the callback. An event might be, for example, the user push a button in a widget. The callback mechanism is the base for our UI accessing the thesaurus through API.

In our design, UI provides various windows, which are managed by different widgets, for users to read and write data. These data are store in the resource variables of corresponding widgets. When users want to execute some functions, system get the arguments values from the resources and pass these data to callback. Inside callback, a calling to one or more API routines are included. Callback will be responsible to direct the argument data to the routine(s) and activate it/them to access the thesaurus. For example, a user may want to delete an entry term. He put the word and semantic classification into the window and click a **Entry Term Delete** button. The UI application then activate the *entry-term-delete* callback to read the word and semantic classification. Within

the callback, a truly deletion interface function, that is the *DelEntryTerm*, will be called to do this work.

4.3.2 Bilingual User Interface

This UI allows users to interact with the system in Chinese GuoBiao, Chinese Big-5 and English language. Actually, the UI program is written in a way that make no assumptions about the language. Such an idea is supported by using the *message catalogs* mechanism.

The main concept in the message catalog mechanism is the *message text* in program. The message text is the data specific to any particular language, such as the window title and label content in user interface program. For example, in this UI, the heading of the top window for English processing environment is "TheSys - Thesaurus System", while it should be "通意 通用語意詞典系統". If such message text is hardcoded in program, the program can only serve for particular language. So, it is proposed to make these message text held in a message catalog separate from the main body of the program. And the message catalog can be translated into several languages to meet the language requirements of each user. This is the behind idea of message catalog mechanism.

Actually, the message catalogs are simple databases that stores the message texts. Each message catalog serve for particular language. Normally, it is a directory which includes two files, **.msf** file and **.cat** file. The .msf file is a text file that stores message text in particular languages. These message file is organized as follows,

```
$ /*
$  * X/OPEN message catalogue
$  */

$ In xmain.c
```

```
$set 1

1 OK
2 Cancel
3 Help

$set 2
1 Word
2 Semanteme
3 Relationship
```

In this file, the line led by a dollar sign and a space is a comment line, such as the first to fifth line. The lines not led by a dollar sign are the message texts. Each message text is led by a number which will be referenced by other routines. However, these message texts are divided into several sets by the $set line. The number that follows the $set is the set number which will also be referenced by other routines. Actually, the set number, the message text number and the message catalog identity, normally the message catalog directory path, can identify unique one message text. The .cat file is a binary file generated from .msf file. Such two file must work with a **catgets** library routines. This routine is embedded in the program. Each position that the message text should occur will be placed such a routine. In run time of program, this routine will go to particular message catalog to take out the message text to replace its position. So that, in each run time, the real message text is displayed. There are four arguments in this routine. The first is a message path which point to a message catalog. The second is a number indicating the set number and the third is a number indicating the message text number. The last is a pointer to a string which is a default string for the message text. If the routine can't retrieve the message text from the message catalog, such default string will be a substitute.

The following is a segment of a program that shows how the message catalog mechanism work.

```
#include <nl_types.h>
.
.
nl_catd _m_catd;
.
.
_m_catd = catopen(mesgpath, 0);
.
.
printf(catgets(_m_catd, 1, 1, "OK"));
printf(catgets(_m_catd, 1, 2, "Cancel"));
printf(catgets(_m_catd, 1, 3, "Help"));
.
.
```

In *TheSys*, the UI control its language specific data and functions by setting the message path. Different message path direct the program retrieve different data. Currently, there are three message catalogs created, GB, B5 and ENG. These three catalogs server for GuoBiao, Big5 and English respectively. Before running the UI program, users must assign an environment variable **LANG** whose value can only be GB, B5, or ENG. According to this value, the program can make a decision to go to one of above catalogs to retrieve message text data.

4.3.3 Chinese Character Input Method

As Motif system is originally developed in English language processing environment. Most of its tools is convenient for English but Chinese. When building a

Chinese Motif application, we need to write our own tools and windows specially for Chinese processing. The most difficult one is the Chinese character input.

As the common key board is designed for English input, it is needed to use a software method to deal with the Chinese character input problem. The idea behind this method includes two step. First, we need a kind of standard that matchs a string of English character to a list of Chinese characters, such as CangJie standard, WuBi standard etc. According to this standard, if one user want to display a Chinese character on the screen, he can enter the English string which will be mapped to this Chinese character. Following, a set of software tools embedded in an application program will translate the English string to corresponding Chinese character. In motif application, the creation and functions of such software tools are as follows,

1. Write a routine, which is referred to as **ChineseInput**, to initialize a window area, which is referred to as *Chinese Input Area*, for displaying the Chinese Characters.

2. ChineseInput accepts a string of English character entered from keyboard.

3. Such character strings will be passed to another routine, which is referred to as **CheckTable**.

4. According to this string, CheckTable routine go to retrieve Chinese characters in a Chinese character mapping table. This table match the English string to Chinese characters. For example, the string of "zhong" corresponds a Chinese Character "中". Since a string may corresponds more than one Chinese characters, the routine could retrieve more than one corresponding Chinese characters.

5. A list of matched Chinese characters is returned by CheckTable, and passed back to ChineseInput.

6. ChineseInput displays the list of Chinese characters in the Chinese input area for selection. User can select the Character he want.

In Motif, there is no such such Chinese character mapping tables. So, we use some tools and source files in the CxTerm system to help to write the Chinese character input software.

In *TheSys*, the Chinese mapping tables are gotten from CxTerm. They are **CangJie.cit**, **PY.cit** and **Simple.cit** which serve for CangJie, PinYin and Simple CangJie input method respectively. The retrieval routine, which is named **cit2tit.c** is also from CxTerm. It has two header file, **HZinput.h** and **cit2tit.h**. At last, the ChineseInput routine is named **Input_Callback** in a source file input.c. Such a routine is activated in the programs of the UI. The method is,

- Initialize a frame box for loading Chinese character.
- Add an event handler input_callback to this frame box.

Consequently, the Chinese input area will be displayed in the bottom of the window. And the selected Chinese character from input area can be displayed the frame box.

Chapter 5

Conclusion And Future Work

Thesaurus can be used for intelligent processing of information in comprehensive area. The TheSys system presented in this thesis help users build the thesauri according to their own requirements. The main feature of TheSys is that it allows users define arbitrary relationships among terms in a way useful to the application environments. Its API design mechanism make the system as a comprehensively applicable package which can be ported to different applications easily. Currently, we have use this tools built up a Chinese thesaurus thesaurus containing 10,000 entry terms among which synonymous, quasi-synonymous, antonym and part-of/composed-of relationships are embedded.

However, this system still has some aspects needed to be improved. They are the search of related word in cross-relation context, the weighting scheme model and the automatic recognition of relationship.

The current version of TheSys just support searching related words in particular relation type. However, users may need to do this search in the multiple relation type context. For example, a user may want to search the related words of "race" in the scale of quasi-synonymous and broader/narrow relationships. His expectation may be to first search out all the quasi-synonyms of the "race", and then search out all the broader/narrower terms of the quasi-synonyms.

We proposed a weighting scheme in this thesis. However, we need more experiment to evaluate its performance, especially for the link weight assignment.

The link weight assignment directly affect the semantic distance between two semantemes. As a relation type need variable link weight, we have two methods to assign the value. One is classified the range into several scale and let users indicate which scale the value should go down. This method is employed by *TheSys*. Its advantage is to keep the thesaurus frame relatively stable and simplified the management. The second method is dynamically assign the weight in a way that users indicate the relatively closeness among the owner-members context. This method model the reality more closely but make the semantic distance calculation complicated. Therefore, we are trying combine these two methods' advantages to develop more efficient and effective approach. Further, we plan to separate the weighting scheme from the system so that the users can develop their own weighting scheme. At this point, each type of weighting scheme method is referred to as one *weighting scheme model*. The idea behind the weighting scheme model is the system provide an interface to the weighting scheme model. User can follow the system's guideline to design his model. Also, system itself will include several pre-defined weighting scheme models. User can link his weighting scheme model to the system as well as select one of the pre-defined weighting scheme models. This way, the system can be more flexible and powerful.

Now, users define the relationships among terms by the experience. In this way, they may unconsciously capture erroneous information which degrade the function of the thesaurus. Therefore, we want to make the thesaurus building system more intelligent. That is it has the function for automatic recognition of relationship among terms which is a much more challenging topic [2]. The basic idea behind it is that if two words are found frequently occurring side by side, it is possible they have some particular association. Through the analysis and statistic on large volume of document, system can give users hints about these information.

References

[1] "Advanced Learner's Dictionary of Current English", *Oxford University Press*, 1984

[2] Dagobert Soergel, *Indexing Languages and Thesauri: Construction and Maintenance*, John Wiley & Sons, Los Angeles, California, 1974

[3] Igor A. Bolshkov, " Thesaurus In Word Processors: What Should It Be?", *Internal Forum Information and Document*, vol. 16, No. 2, April 1991

[4] Young Whan Kim and Jin H. Kim, "A Model of Knowledge Based Information Retrieval with the Hierarchical Concept Graph", *Journal of Documentation*, Vol. 46, No. 2, June, 1990, pp113-136

[5] Charles F. McMath, Robert S. Tamaru, and Roy Rada, "A Graphical Thesaurus-based Information Retrieval System", *International Journal on Man-Machine Studies*, Vol. 31, 1989, pp121-147

[6] Jane Morris and Graeme Hirst, "Lexical Cohesion Computed by Thesaurus Relations As Indicator Of The Structure of Text", *Computational Linguistics*, Vol. 17, No. 1, March 1991

[7] Roy Rada and Judith Barlow, "Document Ranking Using an Enriched Thesaurus", *Journal of Documentation*, Vol. 47, No 3, September 1991, pp 240-253

[8] Zeng Minzu, Chen Yu, "Towards A Chinese Information Retrieval And Processing System Based on Natural Language Processing", *Proceedings of the 1992 International Conference on Chinese Information Processing*, Vol. 1, Oct. 26-28, Beijing, China, pp108-116

[9] Beijing Information Technology Institute, "The News Information Processing System for People's Daily", *Technical Report*, Beijing Information Technology Institute, August, 1992

[10] Shi Shuicai and Su Dongzhuang, "ExpCIR - An Expert System For Chinese Full Text Retrieval", *Proceedings of the 1992 International Conference on Chinese Information Processing*, Vol. 2, Oct. 26-28, Beijing, China, pp163-170

[11] Kam-Fai Wong and Vincent Lum, "The Chinese Information Retrieval Project Work Plan", Doc No. CHIRP.DD.SE.001, Department of System Engineering, The Chinese University of Hong Kong, June 1, 1993

[12] Zhung Mubin and Su Dongzhuang, "Design and Analysis of Chinese Thesaurus and Retrieval Computer System", *Computer Journal*, Vol. 1, Jan, 1990. Beijing, China.

[13] National Library of Medicine, Medical Subject Heading Section, *Medical Subject Headings, Tree Structures*, National Technical Information Service, Springfield, Virginia, 1986

[14] J. Sammet, and A. Ralston, "The New Computing Reviews Classification System-Final Version", *Communications of the ACM*, Vol. 25, 1982, pp13-25

[15] Keh-Jiann Chen, "Design Concepts for Chinese Parsers", *Proceedings 1992 International Conference on Chinese Information Processing (1)*, Vol. 1, Oct. 26-28, 1992 Beijing

[16] Changning Huang, "A conversation on Large Scale Running Text Processing", *Proceedings 1992 International Conference on Chinese Information Processing (1)*, Vol. 1, Oct. 26-28, Beijing, China, pp36-41

[17] Lin-Shan Lee, Lee-Feng Chien, Long-Ji Lin, James Huang, and K.J. Chen, "An Efficient Natural Language Processing System Specially Designed for the Chinese Language", *Computational Linguistics*, Vol. 17, No. 4, Dec. 1991, pp348-374

[18] Changning Huang and Zuyao Chen, "The First Steps Towards a Semantic Dictionary", Journal of Chinese Information Processing, Vol. 2, No. 3, 1988

[19] He Kekang, "Building Machine Dictionary and Semanteme Analysis", Proceedings of the 1992 International Conference on Chinese Information Processing, Vol. 1, Oct. 26-28, Beijing, China

[20] Roy Rada, Hafedh Mili, Gary Letourneau, and Douglas Johnston, "Creating and Evaluating Entry Terms, Journal of Documentation, Vol. 44, No. 1, 1988

[21] Betty Eddison and David Batty, "Database Design", *Database*, December 1988.

[22] Mei Jiaju, Gao Yunqi, "A Study of the Formalization of Semantics", *Communications of COLIPS*, Vol 2, No 1, 1992, pp. 40-47

[23] K.T.Lua, "A Study of Chinese Word Semantics", *Computer Processing of Chinese & Oriental Languages*, Vol. 7, No.1,

[24] Wang Young Cheng, "Chinese Information Processing Technology and Its Base", *The Press of Shanghai Communication University*, Jan, 1992. June 1993, pp 37-60

[25] John S. Quarterman, Susanne Wilhelm, "Unix, POSIX, And Open Systems", *Addison-Wesley Publishing Company*, 1993

[26] Douglas, A. Young., "The X Window System Programming and Applicaitions with Xt Osf/Motif Edition", *Prentice Hall, Inc*, 1990

Appendix A

System Installation

As a product, *TheSys* is provided as a package of files. Basically, these files are divided into three categories. The first category is a set of management information files which help the system maintain the internal thesauri. The second is a set of data files which store the thesaurus data. The last is a set of C source code files. All these files should be placed under a directory which is referred to as *TheSys Working Directory*. Since the system can parallelly build and maintain more than one thesaurus, each thesaurus has its own set of data files which are put in different directories under the working directory. The following figure shows the organization of the files in *TheSys* system.

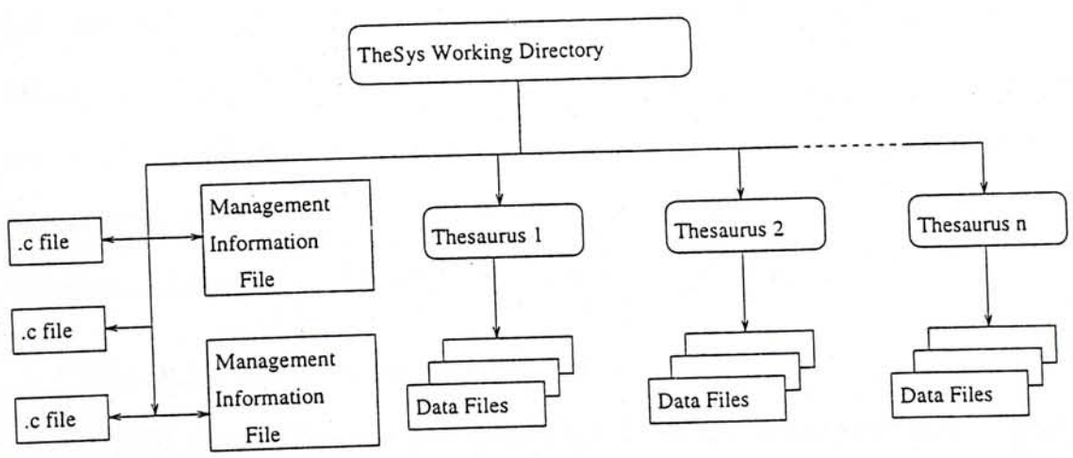

Figure A.1: The Organization Of Files In TheSys

Gerenally speaking, the installation of the system include two steps. The

first is to create the working directory and load the files of TheSys into the directory. The second is to compile the source code files in local environments. In this appendix, we would explain each of the files in *TheSys* first. And then we show how to use *TheSys* as a building block in programming and how to set up the *TheSys* system with user interface. Also, as mentioned in the system design principle, the system allow user verify the validity of the recorded word by using an external dictionary. Such an issue will be discussed in the end of this appendix.

A.1 Files In TheSys

As described above, the first group files in *TheSys* is a set of management information files. They are,

- **ThesauriList.sys**
  This is a file that records all the thesauri maintained in the system. It is directly under the working directory and will be automatically created when the first thesaurus is created in the system.

- **GB B5 ENG**
  These are three message directories for user interface. They respectively serve for GuoBiao, Big5, and English user interface. They are part of the system package so that user must copy them into their own working directory. Inside each of the directories, there are following files,

  - **CangJie.cit PY.cit simple.cit**
    These three files are **cit** tables for Chinese Input method. They corresponds to CanJie, Pingying and Simple CangJie respectively.

  - **Main_menu**
    This is a resource file for Motif window manager to set up environment

for the user interface. It is very simple so that we include it here, user can modify it according to his local environment and requirements:

```
Main_menu*fontList:-*-courier-*-r-*--14-*=english,\
-hku-fixed-medium-r-normal--16-160-72-72-c-160-big5.hku-0=chinese

Main_menu*traversalOn:TRUE

Main_menu*highlightColor:Yellow
Main_menu*highlightThickness:1

Main_menu*keyboardFocusPolicy:pointer
Main_menu*highlightOnEnter:TRUE

Main_menu*foreground:Black
Main_menu*background:White

Main_menu*heading*foreground:Yellow
Main_menu*heading*background:Blue

Main_menu*button*foreground:Black
Main_menu*button*background:LightGray

Main_menu*input_ks*foreground:White
Main_menu*input_ks*background:Blue
```

- message.cat message.msf

  This is the message file and message catalog file for the message title in user interface.

The second group of files is the data files. They are,

- **The***

  These are a series of directories, such as The1, The2, ... Each of such directory stores all the data files of a thesaurus. This directory and inside data files will be automatically created when users create a new thesaurus. They are,

  1. **ENTRY_TBL SEMANTEME_TBL CLASS_TBL SYNONYM_TBL**

  2. **ENTRY_IDX SEMANTEME_IDX CLASS_IDX SYNONYM_IDX**

  3. **RelationType.sys**

  4. **Relation*.rel**

  The first set of files are used to record the entry terms, semantemes, semantic classifications of words and synonyms. The second set of files are the index files corresponding the above files respectively. The third set of file is the relation type definition file. The last set of files are a set of relaitonship link files.

The third group files are the C source codes and make files. Users must compile them in their own system environment. These files are,

- **TheSys.h**
- **library.c input.c cit2tit.c app.c relation_cb.c word_cb.c semclass_cb.c semanteme_cb.c xmaintenance.c xapplication.c fileman.c naming.c semclass.c con.c rc.c tr.c**
- **MakefileAPP MakefileUI**

The above third set of files are make files. They respectively serve for application package usage and independent system with user interface. When users want to use the the system as one of the above two ways, they must rename the make files into **makefile**

A.2 Employ TheSys As Application Package

When other systems want to employ the *TheSys* as a building block, it must follow the procedure below:

1. Make a directory as *TheSys Working Directory.*
2. Copy all the ***.c** files and ***.h** header files into this directory
3. Copy the **MakefileAPP** file into this directory and change its name to "makefile".
4. If **ThesauriList.sys** exists, copy it into the directory.
5. Make the makefile
6. If a C program, say **externapp.c**, want to call the routines in *TheSys*, it just need to include the **thesys.h** header file.
7. When compiling the externapp.c, user need to add below line into its make file,
   cc -o externapp externapp.c -lThesys
   That is to link a local library named **libTheSys.**

The content of the **MakefileAPP** is as follows,

```
XLIB = -li -g
CFLAGS = -c -o -Wf,-XNh2000 -Olimit 2000 -DSTRINGS_ALIGNED
         -D_NO_PROTO -DNO_REGEX
```

```
SRC = fileman.c naming.c semclass.c con.c rc.c tr.c
OBJS = fileman.o naming.o semclass.o con.o rc.o tr.o
HDRS = thesys.h
CC = cc
all: $(OBJS)
ar rv libTheSys.a $(OBJS)
#$(OBJS): $(SRC) $(HDRS)
# cc $(CFLAGS) $(OBJS) $(SRC)
fileman.o: fileman.c thesys.h
        $(CC) $(CFLAGS) fileman.c
naming.o: naming.c thesys.h
        $(CC) $(CFLAGS) naming.c
semclass.o: semclass.c thesys.h
        $(CC) $(CFLAGS) semclass.c
con.o: con.c thesys.h
        $(CC) $(CFLAGS) con.c
rc.o: rc.c thesys.h
        $(CC) $(CFLAGS) rc.c
tr.o: tr.c thesys.h
        $(CC) $(CFLAGS) tr.c
```

A.3 Set Up TheSys With UI

The setting of the system with user interface include the following steps.

1. Make a directory as *TheSys Working Directory.*
2. Copy all the *.c files and *.h header files into this directory
3. Copy the message catalog directories **B5, GB, and Eng** into this directory.

4. If **ThesauriList.sys** exists, copy it into the directory.

5. Copy the **MakefileUI** file into this directory and change its name to "makefile".

6. make the makefile. If a user just want to use the application module, he need to do the make work as follows,
   make application
   or, if he want to use the maintenance module, he need to do make work as follows,
   make maintenance

The content of the **MakefileUI** is as follows,

```
XLIB = -lXm -lXt -lX11 -li -lTheSys -g
XLIBPATHS= -L/usr/X11R5/lib -L/usr/motif1.1/lib
CFLAGS = -I/usr/X11R5/include -I/usr/motif1.1/include
         -c -Wf,-XNh2000 -Olimit 2000 -DSTRINGS_ALIGNED
         -D_NO_PROTO -DNO_REGEX
SRC0 = xmaintenance.c
SRC1 = xapplication.c
SRC3 = library.c input.c cit2tit.c app.c relation_cb.c
       word_cb.c semclass_cb.c semanteme_cb.c
SRC4 = fileman.c naming.c semclass.c con.c rc.c tr.c
OBJS0 = xmaintenance.o
OBJS1 = xapplication.o application.o
OBJS3 = library.o input.o cit2tit.o app.o relation_cb.o
        word_cb.o semclass_cb.o semanteme_cb.o
OBJS4 = fileman.o naming.o semclass.o con.o rc.o tr.o
HDRS = thesys.h
CC = cc
```

```
all: maintenance application
maintenance: $(SRC0) $(OBJS0) $(OBJS3) $(OBJS4)
        $(CC) -o maintenance $(OBJS0) $(OBJS3) $(OBJS4)
              ${XLIBPATHS} ${XLIB}
application: $(SRC1) $(OBJS1) $(OBJS3) $(OBJS4)
        $(CC) -o application $(OBJS1) $(OBJS3)  $(OBJS4)
              ${XLIBPATHS} ${XLIB}
# program dependencies ####
xmaintenance.o: xmaintenance.c thesys.h
        $(CC) -c $(CFLAGS) xmaintenance.c
xapplication.o: xapplication.c thesys.h
        $(CC) -c $(CFLAGS) xapplication.c
library.o: library.c thesys.h
        $(CC) -c $(CFLAGS) library.c
input.o: input.c thesys.h
        $(CC) -c $(CFLAGS) input.c
cit2tit.o: cit2tit.c thesys.h
        $(CC) -c $(CFLAGS) cit2tit.c
fileman.o: fileman.c thesys.h
        $(CC) -c $(CFLAGS) fileman.c
naming.o: naming.c thesys.h
        $(CC) -c $(CFLAGS) naming.c
semclass.o: semclass.c thesys.h
        $(CC) -c $(CFLAGS) semclass.c
semclass.o: semclass.c thesys.h
        $(CC) -c $(CFLAGS) semclass.c
con.o: con.c thesys.h
        $(CC) -c $(CFLAGS) con.c
```

```
rc.o: rc.c thesys.h
        $(CC) -c $(CFLAGS) rc.c
tr.o: tr.c thesys.h
        $(CC) -c $(CFLAGS) tr.c
relation_cb.o: relation_cb.c thesys.h
        $(CC) -c $(CFLAGS) relation_cb.c
semclass_cb.o: semclass_cb.c thesys.h
        $(CC) -c $(CFLAGS) semclass_cb.c
semanteme_cb.o: semanteme_cb.c thesys.h
        $(CC) -c $(CFLAGS) semanteme_cb.c
word_cb.o: word_cb.c thesys.h
        $(CC) -c $(CFLAGS) word_cb.c
```

If a user interface is used, users need to assign two environment variable. The first is **APPLRESDIR** which give out the directory path of the working directory installing the system. The second is **LANG** which must be one of the following values, **B5**, **GB**, and **ENG**. It is obvious that this variable corresponds to Big5, GuoBiao, and English processing environment. The above two values direct the system to retrieve resources and message data for setting up different interactive environment.

A.4 Verify The Word Using External Dictionary

As described in the system design principle, the system allow users verify the validity of the recoded word by using an external dictionary. The implementation of such a scheme depends on which way the system is used, either as an application building block or as an independent system with user interface.

When the system is employed as an building block, the verification of the recorded work is done inside the application program which includes the *TheSys*.

That is users themselves provide a verification routine to check the validity of the word. Only successful checking, he can call the *TheSys InsEntryTerm* routine to insert the word as entry term. The usage of this routine is up to users. Also, its syntax is defined by the users. The below example shows a guideline of this scheme, (the bolded routine is user defined routine)

if **lookaword(fp, word);**
InsEntryTerm(word, mappedword, wordclass);
else
printf("Such A Word %s Is Invalidate!", word);

If the system is used through the user interface. The requirement for this verification is strict. Users must do three things,

1. Users need to write a C function to include their own verification routine. This function must obey the syntax as follows,

   int LookUpWord(char *word);

   And the concrete implementation of this function could as follows,

```
int LookUpWord (char *word)
{
FILE *fp;

fp = fopen("/usr/local/thesysprj/dictionary", "r");
/* open the external dictionary to read */

/* call the local verification routine */
if lookaword(fp, word)
     /* if the word exists*/
```

```
    return (1);
else
    /* if the word does not exists*/
    return (0);
}
```

2. Compile the function and use the **ar** command archive this function into local library **libTheSys**.

3. Assign an environment variable **EXTERNDICT** as the value of **ON**. This way, the user interface program know to call the *LookUpWord* function.

Following the above guideline, users can make the system to check the validity of a recorded word.

Appendix B

API Description

Typographic Conventions

This volume uses the following typographic conventions:

- **Boldfaced** strings represents literals; type them exactly as they appear.
- *Italicized* strings represent function arguments.

B.1 thesys.h File

When other application systems want to employ the *TheSys* API in their programs, the **thesys.h** file must be included. This file defines all the data structures application programmer need to know. Below is its content,

```
#ifndef _TheSys_H
#define _TheSys_H
/* data structure*/
/* naming.c */
typedef struct TN
{  char *thesaurusname;
   struct TN *next;
} ThesaurusList;
```

```
/* This linked list structure stores the existing thesauri
   names */
/* semclass.c */
typedef struct
{ char *symbol;
} SemanticClassNode;
/* This structure stores the semantic classification node */
typedef struct SCT
{ char    *symbol;
 struct SCT *child_table;
 struct SCT *nextsibling;
} SemanticClassTable;
/* This structure is the semantic classification tree */
/* rc.c, relation link processing related */
typedef struct SD
{float bottomvalue;
 float uppervalue;
 char  *symbolname;
 struct SD *next;
} SymbolDescription;
/* This structure describes the weight range */
typedef struct
{ char  *name;
   int direction;
   int  weightscheme;
   float bottomlinkweight;
   float upperlinkweight;
   SymbolDescription *linkweight; /* weight range */
```

```
  SymbolDescription *semdistance; /* semantic distance
                                     range */
} RelationTypeRecord;
/* This is one relation type record */
typedef struct RTTable
{ char  *relationname;
  char  *direction;
  char  *weightscheme;
  float bottomlinkweight;
  float upperlinkweight;
  SymbolDescription *linkweight;
  SymbolDescription *semdistance;
  struct RTTable  *nextrelation;
 } RelationTypeTable;
 /* This linked list structure stores
    a set of relation type record*/
 typedef struct
 {char *preferredterm;
  char *wordclass;
 } SemantemeNode;
 /* This is a semanteme node used in next semantic
    relationship Link record */
 typedef struct
 { char *SE_weight;
  SemantemeNode start_Semanteme;
  SemantemeNode end_Semanteme;
  char *ES_weight;
 } LinkRecord;
```

```
/* This is a semantic relationship link
   record */
typedef struct
{ char  *wordclass;
  char  *relationname;
  char  *distance;
} SearchScale;
/* This structure describes the search scale
   which will be used in related word search*/
/* WordGroup structure, used by many prototype */
typedef struct WG
{char *word;
 char *wordclass;
 struct WG *next;
} WordGroup;
/* This structure stores a set of entry terms*/
typedef struct SG
{char *preferredterm;
 char *wordclass;
 struct SG *next;
} SemantemeGroup;
/* This structure stores a set of semantemes*/
/* function prototype */
/* naming.c */
extern int OpenThesaurus(char *name);
extern void CloseThesaurus();
extern ThesaurusList *ListThesaurus ( );
/* con.c */
```

```
WordGroup *SearchEntryTerm(char *word, char *wordclass);
int InsertEntryTerm(char *addedword, char *mappedword,
                    char *wordclass);
SemantemeGroup *SearchSemantemes(char *word, char *wordclass);
WordGroup *SearchSynonyms(char *word, char *wordclass);
int ChangIdentifier(char *srcword, char *objword,
                    char *wordclass);
WordGroup *DeleteEntryTerm(char *word, char *wordclass);
int ReLinkEntryTerm(char *srcword, char *srcwordclass,
                    char *objword, char *objwordclass);
/* semclass.c */
extern int InsertSemanticClass ( char *parent,
                          SemanticClassNode *node);
extern SemanticClassTable *ListSemanticClass ( );
extern int DeleteSemClass(char *class);
/* rc.c relation link related */
extern int BuildRelationFrame (RelationTypeRecord record);
extern RelationTypeTable *ListRelationType ( );
extern int AddRelationLink(char *relationname,
                           LinkRecord linkrecord);
extern int DeleteRelationLink(char *relationname, SemantemeNode
          start_semanteme, SemantemeNode end_semanteme);
extern WordGroup *RetrieveRelatedWord(char *word,
                    SearchScale searchscale);
#endif
```

B.2 API Reference

Manual Page Format

The manual pages in this volume use the following format:

NAME

This gives out the name of the interface function described following.

SYNOPSIS

This section describes the appropriate syntax for using the interface.

DESCRIPTION

This section describes the behavior of the interface.

PARAMETER

This section describes function arguments.

RETURN

This lists the values returned by function interfaces.

SEE ALSO

This lists the related function interfaces.

Reference Pages

NAME

OpenThesaurus

SYNOPSIS

```
#include    <thesys.h>
int         OpenThesaurus (name)
char        *name;
```

DESCRIPTION

OpenThesaurus ( ) is afforded for user to name a new thesaurus to be built or indicate system operate on an existing thesaurus. According to this information to create a new directory or go into a specific directory which store corresponding data files, such as Semantic Classification Tree file, Relationship Register file, entry term file and relationship file etc. On success, an integer will be returned, else 0 will be returned.

PARAMETER

name It is a string variable. It indicates the name for thesaurus to be built or operated.

RETURN

On success, 1 will be returned, else 0 will be returned.

SEE ALSO

ListThesaurus CloseThesaurus

NAME

CloseThesaurus

SYNOPSIS

```
#include    <thesys.h>
int         CloseThesaurus ()
char        *name;
```

DESCRIPTION

CloseThesaurus is used to close the current thesaurus. It is commonly called when users want to exit the whole operation on TheSys system.

PARAMETER

RETURN

On success, 1 will be returned, else 0 will be returned.

SEE ALSO

ListThesaurus OpenThesaurus

NAME

ListThesaurus

SYNOPSIS

```
#include          <thesys.h>
ThesaurusList     *ListThesaurus ( )
```

DESCRIPTION

ListThesaurus ( ) will help users get to know what thesauri had been built in the current TheSys system. It will return a pointer to a table which stores the thesauri list. On fail, NULL value will be returned.

RETURN

It will return a pointer to ThesaurusTable. The data struct of ThesaurusTable is in thesys.h file.

SEE ALSO

OpenThesaurus CloseThesaurus

NAME

InsertSemanticClass

SYNOPSIS

```
#include          <thesys.h>
int               InsertSemanticClass (parent, node)
char              *parent;
SemanticClassNode *node;
```

DESCRIPTION

InsertSemanticClass ( ) is afforded to specify the semantic classification tree. Users use this routine to insert one node of this tree each time. The argument *node* defines a semantic classifications node. Its parent is indicated by argument *parent*. On success, integer 1 will be returned, else will be returned.

PARAMETER

parent It is a string variable. It indicate which semantic classification is the parent of the inserted node. If the inserted node is the first level semantic classification, it must be assigned as (char *) 0;

node It is a variable of structure SemanticClassNode. The data structure of SemanticClassNode is in thesys.h

It is noted that duplicate naming of symbol is prohibited.

RETURN

On success, integer 1 will be returned, else 0 will be returned.

SEE ALSO

ListSemanticClass DeleteSemClass

NAME

DeleteSemClass

SYNOPSIS

```
#include    <thesys.h>
int         DeleteSemClass (class)
char        *class;
```

DESCRIPTION

DeleteSemClass ( ) is afforded to delete a semantic classification node from the semantic classification tree. The argument *class* defines a semantic classifications node. Since the duplicate semantic class name is prohibited, such a name is the identity of the semantic classification On success, integer 1 will be returned, else will be returned.

PARAMETER

class It is a string variable. It indicate which semantic classification is deleted. It is noted that a terminal semantic classification can not be deleted unless all the entry associated with it is deleted.

RETURN

On success, integer 1 will be returned, else 0 will be returned.

SEE ALSO

ListSemanticClass InsertSemanticClass

NAME

ListSemanticClass

SYNOPSIS

```
#include                <thesys.h>
SemanticClassTable      *ListSemanticClass ( )
```

DESCRIPTION

ListSemanticClass ( ) is used to help users get to know all the semantic classification defined in the current thesaurus. It will return a pointer to a table which stores the description of all the semantic classification types. If fail, NULL value will be returned.

RETURN

It will return a pointer to structure SemanticClassTable. The data struct of SemanticClassTable is in thesys.h.

SEE ALSO

InsertSemanticClass

NAME

SearchEntryTerm

SYNOPSIS

```
#include      <thesys.h>
WordGroup     *SearchEntryTerm (word, wordclass)
char          *word;
char          *wordclass;
```

DESCRIPTION

SearchEntryTerm ( ) searches for entry terms Which have the same character codes with the word and under the semantic classification wordclass. On success, a pointer to a group of words which also denotes their respective semitic classifications will be returned. If no such entry term exists, a NULL value pointer will be returned.

PARAMETER

word It is a string variable. It stores the characters codes, such as Big5 or GB code, of a Chinese word.

wordclass It is a string variable. It indicates what class of the semantic classification the returned entry terms should be under. Its value must be identical to one of the semantic classification symbol specified in the semantic classification tables, or a NULL value. If NULL is assigned, it means the semantic classification of the returned Chinese word could be any type of classification specified. If this _wordclass hasn't children class, which means it is a terminal classification, the returned entry term have exactly the same word classification as _wordclass. Else, the returned entry terms have the terminal classifications which are under the classification of _wordclass.

RETURN

WordGroup The return of this routine is a pointer to a group of Chinese words along with their semantic classifications respectively. The data structure of WordGroup is in thesys.h.

If search fails, then a NULL value will be returned.

SEE ALSO

SearchSemanteme, SearchSynonyms

NAME

SearchSemanteme

SYNOPSIS

```
#include          <thesys.h>
SemantemeGroup    *SearchSemanteme (word, wordclass)
char              *word;
char              *wordclass;
```

DESCRIPTION

SearchSemanteme ( ) firstly searches for a group of entry terms which are determined by the word and wordclass. Then it will find out all the semantemes representing the entry terms just retrieved. On success, a pointer to a group of identifier words which also denotes their respective semitic classifications will be returned. If no such semanteme exists, a NULL value pointer will be returned.

PARAMETER

word It is a string variable. It stores the characters codes, such as Big5 or GB code, of a Chinese word.

wordclass It is a string variable. It indicates what class of the semantic classification the returned entry terms should be under. Its value must be identical to one of the semantic classification symbol specified in the semantic classification tables, or a NULL value. If NULL is assigned, it means the semantic classification of the returned Chinese word could be any type of classification specified. If this _wordclass hasn't children class, which means it is a terminal classification, the returned entry term have exactly the same word classification as _wordclass. Else, the returned entry terms have the terminal classifications which are under the classification of _wordclass.

RETURN

SemantemeGroup The return of this routine is a pointer to a group of Chinese words along with their semantic classifications respectively. All these Chinese words are the identifier words of the wanted semantemes. The data structure of SemantemeGroup is in thesys.h.

If search fails, then a NULL value will be returned.

SEE ALSO

SearchEntryTerm, SearchSynonyms

NAME

InsertEntryTerm

SYNOPSIS

```
#include    <thesys.h>
int         InsertEntryTerm (addedword,mappedword,wordclass)
char        *addedword, mappedword;
char        *wordclass;
```

DESCRIPTION

InsertEntryTerm ( ) adds a new entry term which is specified by addedword and wordclass into EntryTerm database, and links it to a semanteme which contains an entry term –mappedword and wordclass.

It is noted that the semantic classification of addedword must be identical to the mappedword.

Another notable point is that, if addedword is the same as mappedword, it will make a creation of new semanteme which will be denoted by the new entry term itself.

Since inserting an entry term requires the mapped semanteme being unique, the wordclass here must be terminal semantic classification.

PARAMETER

word It is a string variable. It stores the characters codes, such as Big5 or GB code, of a Chinese word.

wordclass It is a string variable. It indicates what class of the semantic classification the returned entry terms should be. In this case, it must be a terminal classification. NULL value or intermediate classification symbol are illegal.

RETURN

On success, it will returns integer 1, else return 0.

SEE ALSO

ChangeIdentifier, RelinkEntryTerm

NAME

SearchSynonyms

SYNOPSIS

```
#include      <thesys.h>
WordGroup     *SearchSynonyms (word, wordclass)
char          *word;
char          *wordclass;
```

DESCRIPTION

SearchSynonyms ( ) firstly searchs out a group of entry terms which are determined by the word and wordclass. And then it will find out all the synonyms of the retrieved entry terms. On success, a pointer to a group of words which also denotes their respective semitic classifications will be returned. If no such synonyms exists, a NULL value pointer will be returned.

PARAMETER

word It is a string variable. It stores the characters codes, such as Big5 or GB code, of a Chinese word.

wordclass It is a string variable. It indicates what class of the semantic classification the retrieved entry terms should be under. Its value must be identical to one of the semantic classification symbol specified in the semantic classification tables, or a NULL value. If NULL is assigned, it means the semantic classification of the retrieved Chinese word could be any type of classification specified. If this wordclass hasn't children class, which means it is a terminal classification, the retrieved entry term have exactly the same word classification as wordclass. Else, the retrieved entry terms have the terminal classifications which are under the classification of wordclass.

RETURN

WordGroup The return of this routine is a pointer to a group of Chinese words along with their semantic classifications respectively. The data structure of WordGroup is in thesys.h.

If search fails, then a NULL value will be returned.

SEE ALSO

SearchSemanteme, SearchEntryTerm

NAME

ChangeIdentifier

SYNOPSIS

```
#include    <thesys.h>
int         ChangeIdentifier (srcword, objword, wordclass)
char        *srcword, objword;
char        *wordclass;
```

DESCRIPTION

ChangeIdentifier ( ) substitutes the identifier word srcword of semanteme (srcword, wordclass) by using objword. The srcword and objword must be synonyms. If succeed, an integer 1 will be returned, else 0 is to be returned.

It is note that, as the same reason as in *InsertEntryTerm ( )*, the wordclass must be a terminal classification.

PARAMETER

srcword, objword	It is a string variable. It stores the characters codes, such as Big5 or GB code, of a Chinese word.
wordclass	It is a string variable. It indicates what class of the semantic classification the entry terms should be. In this case, it must be a terminal classification. NULL value or intermediate classification symbol are illegal.

RETURN

On success, an integer 1 will be returned. Else, an integer 0 will be returned.

SEE ALSO

InsertEntryTerm, ReLinkEntryTerm

NAME

ReLinkEntryTerm

SYNOPSIS

```
#include    <thesys.h>
int         ReLinkEntryTerm (srcword, srcwordclass, objword, objwordclass)
char        *srcword, *objword;
char        *srcwordclass,*objwordclass;
```

DESCRIPTION

ReLinkEntryTerm ( ) re-maps one entry term to another semanteme. Argument srcword and srcwordclass specifies the source entry term. objword and objwordclass indicate corresponding semanteme. On the operation completion, the semantic classification of the entry term will be changed to objwordclass.

It is noted that, as the same reason in *InsertEntryTerm ( )*, the srcwordclass and objwordclass have to be the terminal semantic classification.

PARAMETER

srcword, objword	It is a string variable. It stores the characters codes, such as Big5 or GB code, of a Chinese word.
srcwordclass, objwordclass	It is a string variable. It indicates what class of the semantic classification the entry terms should be. In this case, they must be terminal classifications. NULL value or intermediate classification symbol are illegal.

RETURN

On success, an integer 1 will be returned, else 0 will be returned.

SEE ALSO

InsertEntryTerm, ChangeIdentifier

NAME

DeleteEntryTerm

SYNOPSIS

```
#include      <thesys.h>
WordGroup     * DeleteEntryTerm (word, wordclass)
char          *word;
char          *wordclass;
```

DESCRIPTION

DeleteEntryTerm ( ) firstly searchs out a group of entry terms which are specified by word and wordclass. Then it will delete these entry term(s) from the entry term database.

This operation may direct a complicated situation. That is deletion make a semanteme empty, it means no entry term mapped to it. If it occurs, the deletion will be prohibited. In another word, semanteme deletion is prohibited.

On return, a ponter to a group of entry term which have been deleted will be returned. If no entry term being deleted, a NULL pointer will be returned.

PARAMETER

word It is a string variable. It stores the characters codes, such as Big5 or GB code, of a Chinese word.

wordclass It is a string variable. It indicates what class of the semantic classification the deleted entry terms should be under. Its value must be identical to one of the semantic classification symbol specified in the semantic classification tables, or a NULL value. If NULL is assigned, it means the semantic classification of the deleted Chinese word could be any type of classification specified. If this wordclass hasn't children class, which means it is a terminal classification, the deleted entry term have exactly the same word classification as wordclass. Else, the deleted entry terms have the terminal classifications which are under the classification of wordclass.

RETURN

WordGroup The return of this routine is a pointer to a group of Chinese words along with their semantic classifications respectively. The data structure of WordGroup is in thesys.h.

If no entry term deleted, then a NULL value will be returned.

SEE ALSO

SearchEntryTerm

NAME

BuildRelationFrame

SYNOPSIS

```
#include          <thesys.h>
int               BuildRelationFrame (record)
RelationTypeRecord record;
```

DESCRIPTION

BuildRelationFrame ( ) lets users define their local convention of the relation type. They must fills in the parameter record which contains all the information *thesys* system need to know about a new type of relation. The definition of the new type of relation will be keep in a *relation register file*. On success, an integer 1 will be returned. Else, an integer 0 will be returned.

PARAMETER

record It is a variable of structure RelationTypeRecord. The data structure of RelationTypeRecord is as follows,

```
typedef             struct
{
char                *name;
int                 direction;
int                 weightscheme;
float               bottomweight;
float               upperweight;
SymbolDescription   *linkweight;
SymbolDescription   *semdistance;
}                   RelationTypeRecord;
```

The explanation of each element is as follows;

name It is a string which is the name of the relationship. In case the registered relationship is bi-directional relationship, such as *is-a/has-instance*, a symbol / need to be placed for separating two subname. On acceptance of this name, routine will create a new *relationship file* in this name for storing relationship link records.

direction Its legal value is integer -1, 0, and 1. -1 means the relationship is non-directional relationship. 0 means the relationship is uni-directional relationship; 1 means the relationship is bi-directional relationship.

weightscheme Its legal value is integer 0 or 1. 0 means there is the relationship link without weight assignment. 1 means that this type of relationship link has to be quantified.

bottomweight *upperweight* It only makes sense in the case that weightscheme is assigned integer 1. It indicates the bottom and upper bound of the weight value on the relationship links.

linkweight *semdistance* It is a pointer to a structure of *SymbolDescription*. It describes a friendly way in which users use some symbols to assign link weight or semantic distance value, rather than use concrete float value. The data structure of SymbolDescription is in thesys.h.

RETURN

On success, an integer 1 will be returned, else 0 will be returned.

SEE ALSO

ListRelationType

NAME

ListRelationType

SYNOPSIS

```
#include <thesys.h>

RelationTypeTable          *ListRelationType ( )
```

DESCRIPTION

ListRelationType ( ) is used to help users get to know all the relation types defined in the current thesaurus frame. It will return a pointer to a table which stores all the information about the relation types. If fail, NULL value will be returned.

RETURN

It will return a pointer to RelationTypeTable. The data struct of RelationTypeTable is in thesys.h.

SEE ALSO

BuildRelationFrame

NAME

AddRelationLink

SYNOPSIS

```
#include      <thesys.h>
int           AddRelationLink ( relationname, linkrecord)
char          *relationname;
LinkRecord    linkrecord;
```

DESCRIPTION

AddRelationLink ( ) adds one relationship link into the relationship file which is determined by the relationname. The information about this relationship link is stored in the parameter linkrecord. This record indicates two semanteme nodes connected by the link, and the weight on the link if there is such requirement. On success, integer 1 will be returned, else 0 will be returned.

PARAMETER

relationname It is a string variable. It indicates which relationship file the linkrecord will be stored.

linkrecord It is variable of structure LinkRecord. The data structure of LinkRecord is in thesys.h.

According to the corresponding record in *relation register file*, routine can determine what information in this record should be extracted and how it can be interpret. For example, if this is a kind of uni-directional relationship with weight, then the content of ES_weight won't be extracted. And if this type of relationship without weight, then the SE_weight and ES_weight is functionless.

RETURN

On success, integer 1 will be returned, else 0 will be returned.

SEE ALSO

BuildRelationFrame

NAME

DeleteRelationLink

SYNOPSIS

#include	<thesys.h>
int	DeleteRelationLink (relationname, start_semanteme, end_semanteme)
char	*relationname;
SemantemeNode	start_semanteme, end_semanteme;

DESCRIPTION

DeleteRelationLink ( ) deletes one relationship link which connects start_semanteme and end_semanteme from relationname relationship file. On success, integer 1 will be returned, else 0 will be returned.

PARAMETER

relationname	It is a string variable. It indicates which type deleted relationship link belongs to.
start_semanteme	It is a variable of structure SemantemeNode. The data
end_semanteme	structure of SemantemeNode is in thesys.h.

RETURN

On success, integer 1 will be returned, else 0 will be returned.

SEE ALSO

BuildRelationFrame

NAME

RetrieveRelatedWord

SYNOPSIS

```
#include      <thesys.h>
WordGroup     *RetrieveRelatedWord (word, searchscale)
char          *word;
SearchScale   *searchscale;
```

DESCRIPTION

RetrieveRelatedWord ( ) searchs out the related words of word in compliance with the parameter searchscale. The searchscale will indicates the relationship types search is to go into and the scope of semantic distance. If related words are found, a pointer to a group of words will be returned, else a NULL value will be returned.

PARAMETER

word It is a string variable. It stores the characters codes, such as Big5 or GB code, of a Chinese word.

searchscale It is a pointer to structure SearchScale. The data structure of SearchScale is as follows:

```
typedef struct      SS
{
char                *relationname;
char                *distance;
struct SS           *next;
}                   SearchScale;
```

The definition of the distance can be found in the BuildRelationFrame () routine. Users just need to use a symbol instead of concrete figure to represent the semantic distance scale.

RETURN

WordGroup The return of this routine is a pointer to a group of Chinese words along with their semantic classifications respectively. The data structure of WordGroup is in thesys.h.

if no related word is found, a NULL value will be returned.

SEE ALSO

BuildRelationFrame

Appendix C

User Interface Reference

In this volumn, we go through each of the windows of the window-based user interface. We assume that users are familiar with the concepts and architecture of *TheSys*. Further, users must at least know one of the three Chinese character input methods, i.e. CangJie, PinYin, and Simple CangJie.

The top window of maintenance user interface is shown as follows,

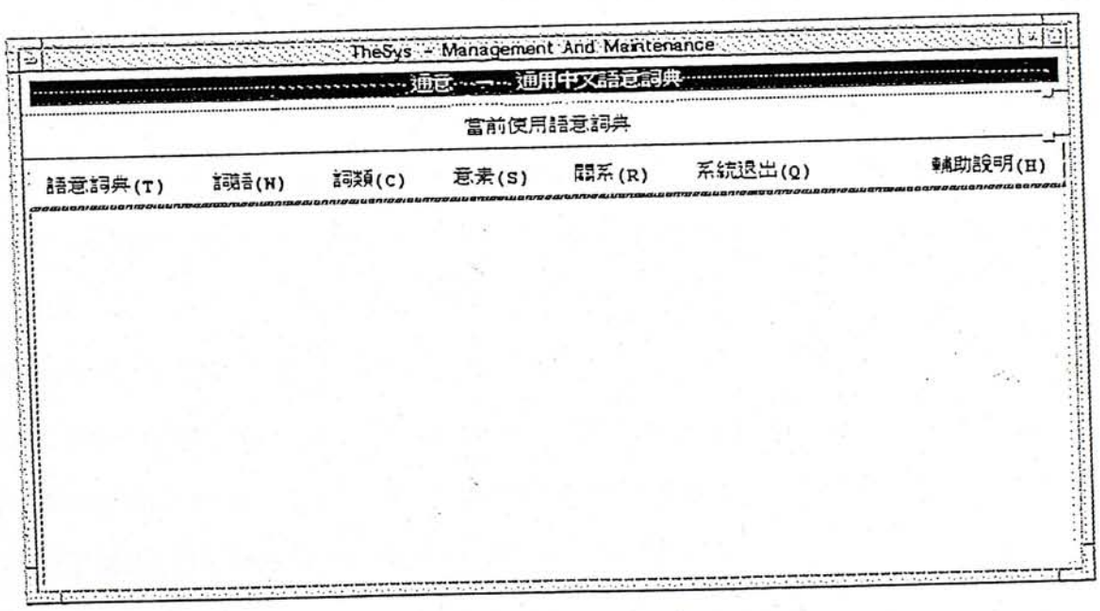

Figure C.1: Maintenance User Interface Top Window

We can see that there are seven menu items in the menu bar. They are 語意詞典 (Thesaurus), 詞語 (Word), 詞類 (Word Class), 意素 (Semanteme), 關系 (Relation), 系統退出 (Quit) and 輔助說明 (Help). Click the Thesaurus menu, we get three options,

they are 創建 (Create), 開啓 (open) and 關閉 (close). Create option offers a window for users build a new thesaurus. The popped up window is as follows,

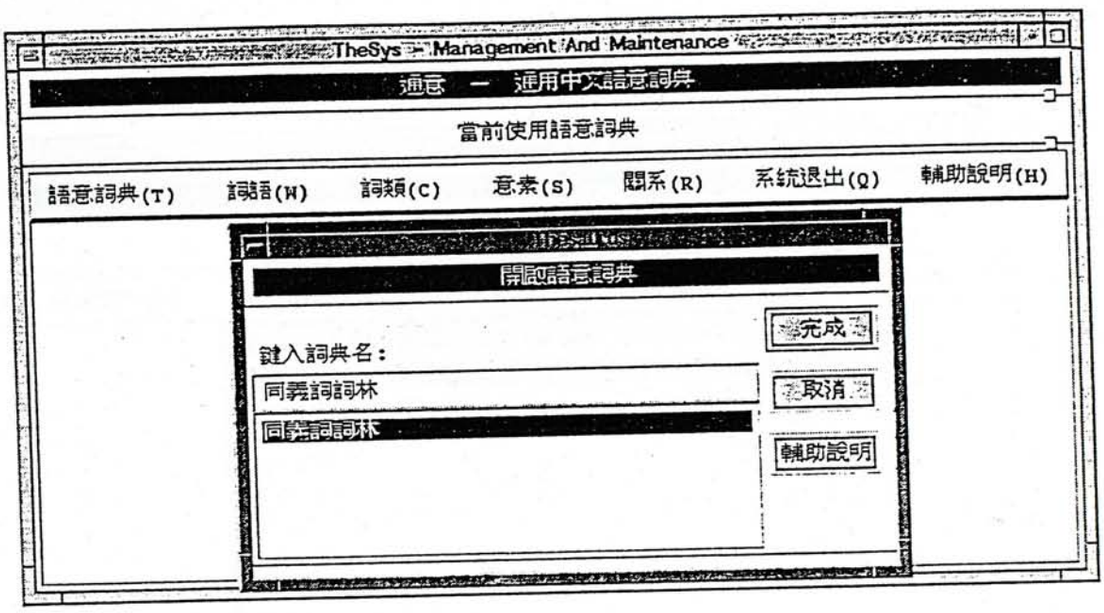

Figure C.2: Create A New Thesaurus

In this window, there is a frame bar under the prompt "輸入詞典名" (Input Thesaurus Name). Also, there is a Chinese character input prompt area on the bottom of the window. The system provide three Chinese input methods, i.e. CangJie, PinYin and Simple CangJie. The default input method is CangJie. User can shift the input method circularly by pressing the shift and ~ keys together. For creating a new thesaurus, users need to move the mouse cursor to the frame, and enter the thesaurus name from the keyboard. When completing, click the 完成 (OK) button. If the thesaurus creation succeed, this window disappear and the thesaurus name is written on the top of menu bar. If fail, the button will sound. The problem is that the entered thesaurus name is the same as the existing one. Users need to enter another name for the new thesaurus.

The 開啓 option offers the users a window to select a existing thesaurus in the *TheSys*.

Figure C.3: Open An Existing Thesaurus

In this window, there is a scroll list which may include a list of existing thesaurus names. For opening a thesaurus to operate, users need to double click one of them so that the thesaurus name is shown off on the frame bar which is under the prompt "輸入詞典名". Subsequently, users press the 完成 button. The thesaurus name will be displayed on the top of menu bar. If there is no thesaurus maintained in the system. The scroll list is empty.

The 關閉 option inquiry if you want to close the current thesaurus being processing. It is just a Yes/No dialog.

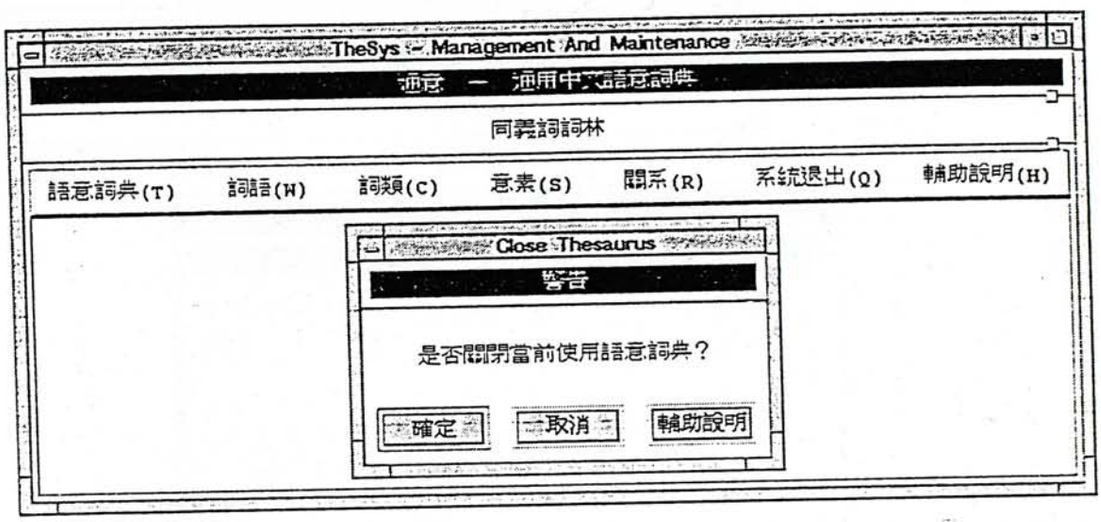

Figure C.4: Close A Thesaurus

When you click 詞語 menu, you can get three options. Those are 插入 (Insert), 刪除 (Delete) and 覽 (Browse). **Insert** option pops up the following window,

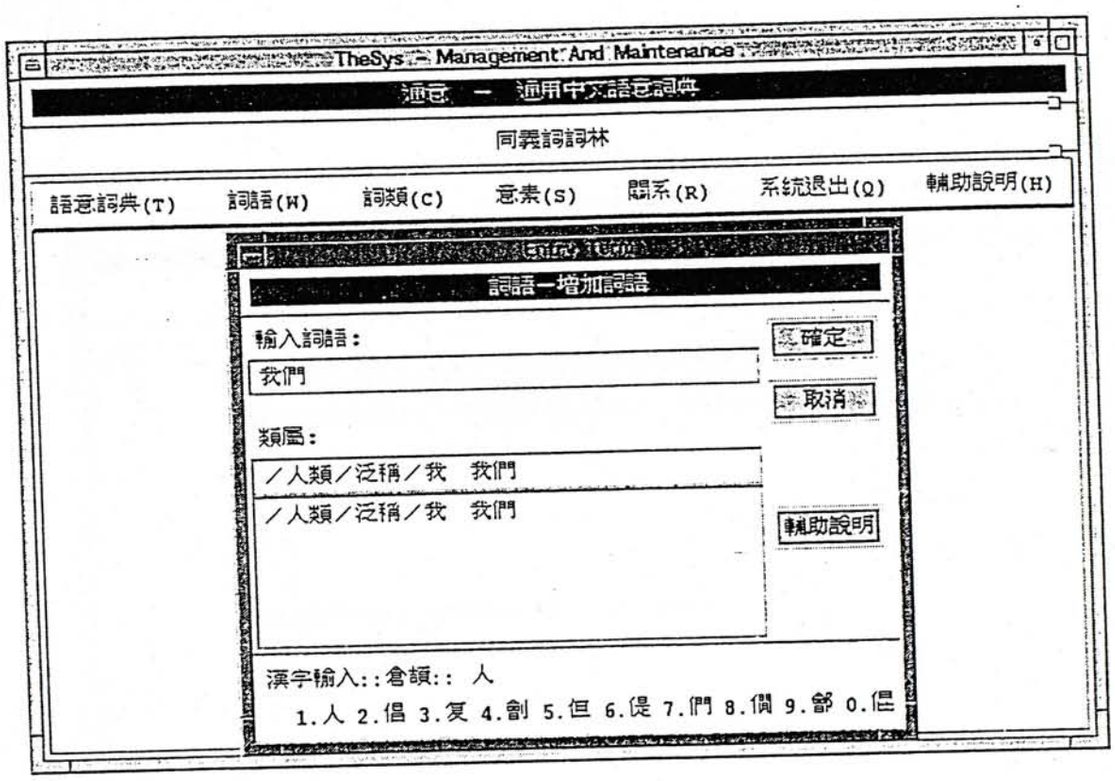

Figure C.5: Insert An Entry Term – Window 1

In this window, there are a frame bar under the prompt "輸入詞語" (Input Word) and a scroll list. The scroll list include a set of word semantic classifications. The toppest one is the parent classification of all other classifications. If the parent classification is a back slash sign, it means all other classifications in the list are the first level classifications. By clicking the parent classification, users make the scroll list display the siblings of the parent classification. By clicking other classification, users make the scroll list display its children classifications. If the classification is a terminal classification, the clicking make no children classification displayed but pop the classification up to the frame bar under 類屬 (Word Semantic Classification). That is the way user select a terminal

semantic classification. For inserting an entry term, users need to enter the word on the frame bar under "Input Word" prompt and select the terminal semantic classification. After filling these two data, press the 確定 (confirm) button. If fail, the button will sound. The reason may due to such word is invalid. User need to check if such word is included in a local word dictionary. If succeed, a window as below is popped up,

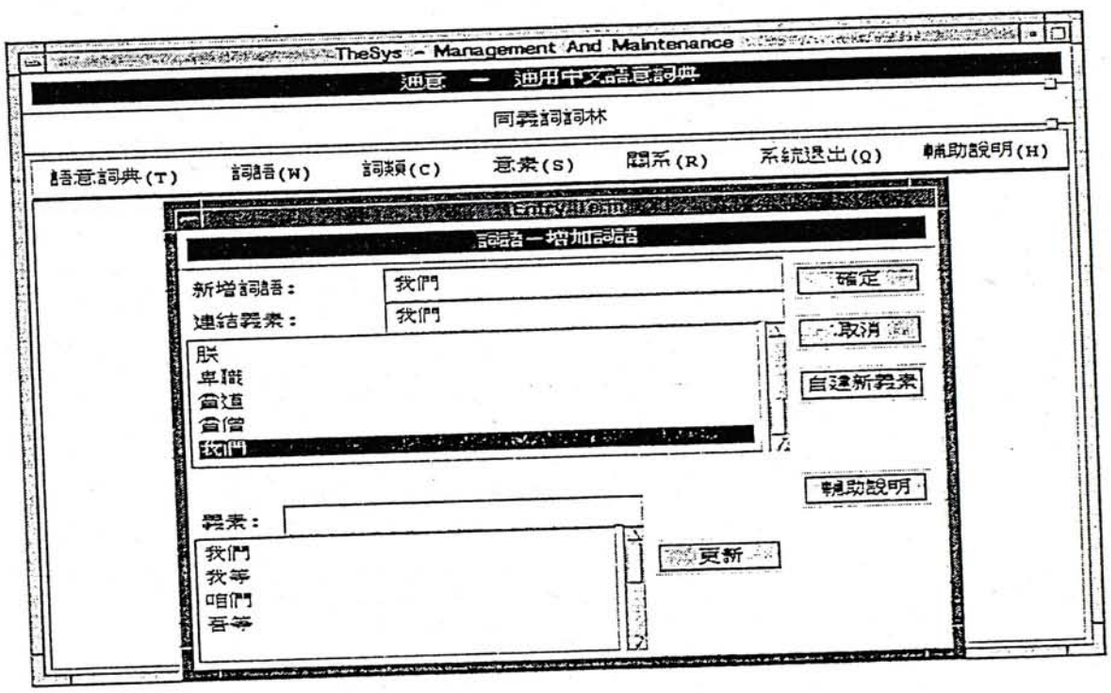

Figure C.6: Insert An Entry Term – Window 2

This window will ask user which the semanteme into which the newly inserted entry term should be mapped. Also, it shows the existing synonyms of particular semanteme. In the upper scroll list, a set of semantemes which are under the word classification defined by last window are shown. User selects one of them by double clicking the semanteme. When the semanteme is selected, it will be displayed on the top of the upper scroll list and all its synonyms will be displayed on the lower scroll list at the same time. At this point, user can select arbitrary synonym as a new representative of the semanteme by double clicking the synonym. When all these data settle down, press the confirm button to

complete the entry term insertion work.

Delete Entry Term window is similar to the Insert Entry Term window. The same information is needed. However, the semantic classification needn't be a terminal classification. So, the scroll list allow user select a non-terminal classification by double clicking any of the classification. When filling in the two frame bar, press the 刪除 button.

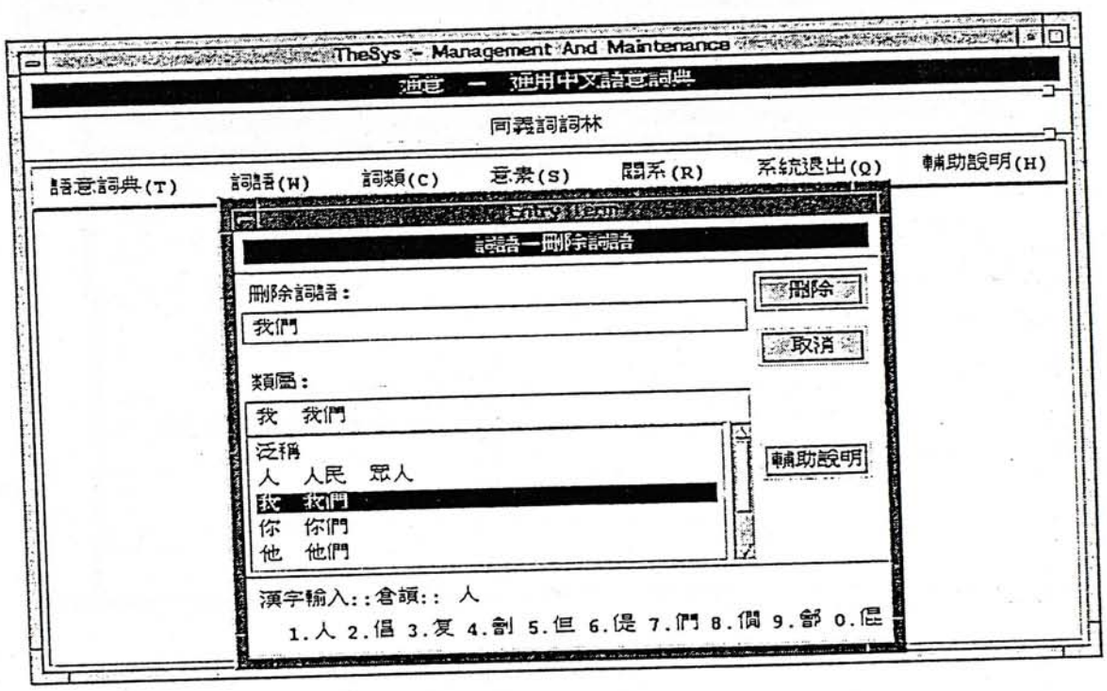

Figure C.7: Delete An Entry Term

If users want to browse the entry terms, select the 覽 option to pop up the Entry Term browse window. Different from the above two windows, this window has an extra area to display the browse result.

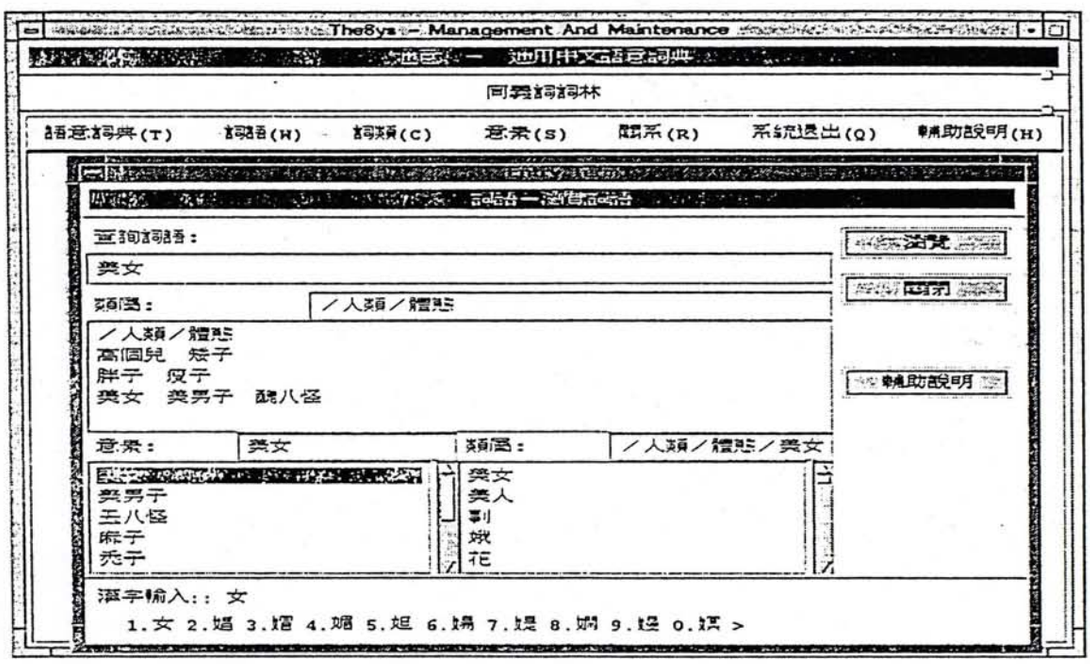

Figure C.8: Browse The Entry Term

User enter the word and word classification information as above described. It is noted that the word classification needn't be terminal classification too. So, user can select the classification by double clicking it. After do that, user press the 覽 button to ask the system to search out all the entry terms matching with the word and word classification. If there are such entry terms, the semantemes into which the entry term mapped will be displayed on the right scroll list of the result area. By double clicking one of them, the synonyms of the corresponding semanteme will be display on the left scroll list. At the same time, the semanteme will be displayed on the top of right scroll list and the word classification of such semanteme will be display on the top of the left scroll list. If search fails,

If users want to browse the entry terms, select the 覽 option to pop up the Entry Term browse window. Different from the above two windows, this window has an extra area to display the browse result.

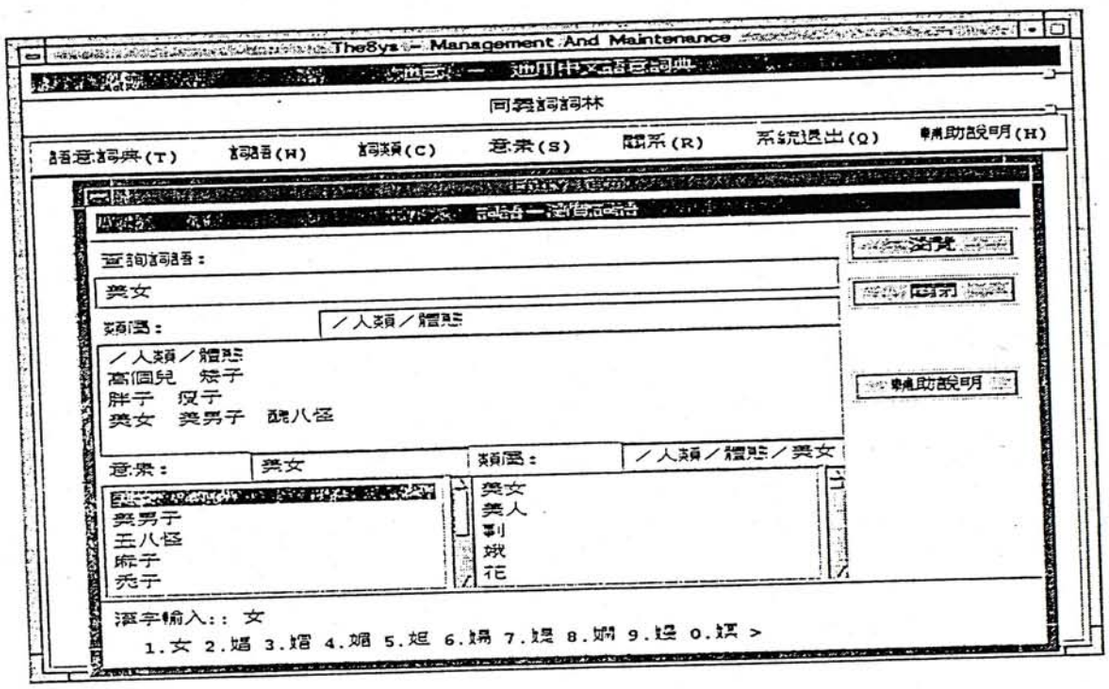

Figure C.8: Browse The Entry Term

User enter the word and word classification information as above described. It is noted that the word classification needn't be terminal classification too. So, user can select the classification by double clicking it. After do that, user press the 覽 button to ask the system to search out all the entry terms matching with the word and word classification. If there are such entry terms, the semantemes into which the entry term mapped will be displayed on the right scroll list of the result area. By double clicking one of them, the synonyms of the corresponding semanteme will be display on the left scroll list. At the same time, the semanteme will be displayed on the top of right scroll list and the word classification of such semanteme will be display on the top of the left scroll list. If search fails,

both the scroll list are empty.

In the 詞類menu, there are three options 插入(Insert), 刪除(Delete), and 覽(Browse).

The **Insert** option pops up the Word Classification Insertion window as follows,

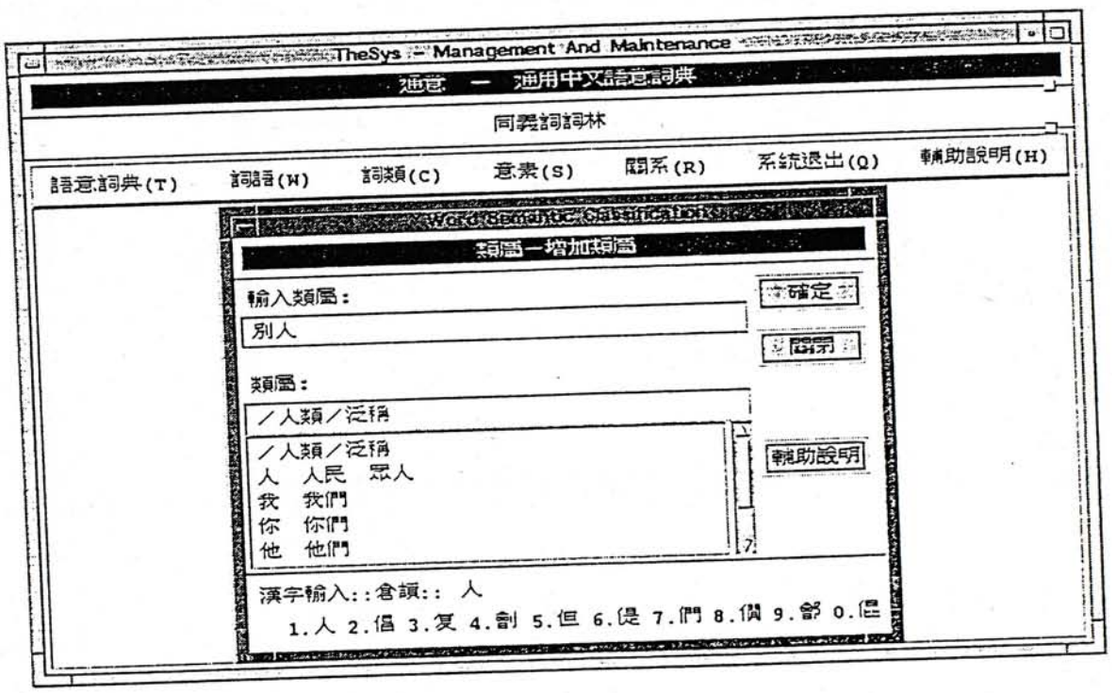

Figure C.9: Insert A Word Class

This window requires user enter two data. One is a name for the newly inserted classification, another is its parent classification. The classification name should be entered into the frame bar from the key board and the parent classification should be selected from the scroll list by double clicking the classification. After filling in these two data, user need to press the 確定 button. If succeed, the window disappear, else the button will sound as a bell. The problem is the inserted class name is duplicate to the existing one. In such case, user need to use other names and redo the insertion.

The word classification deletion window is as follows,

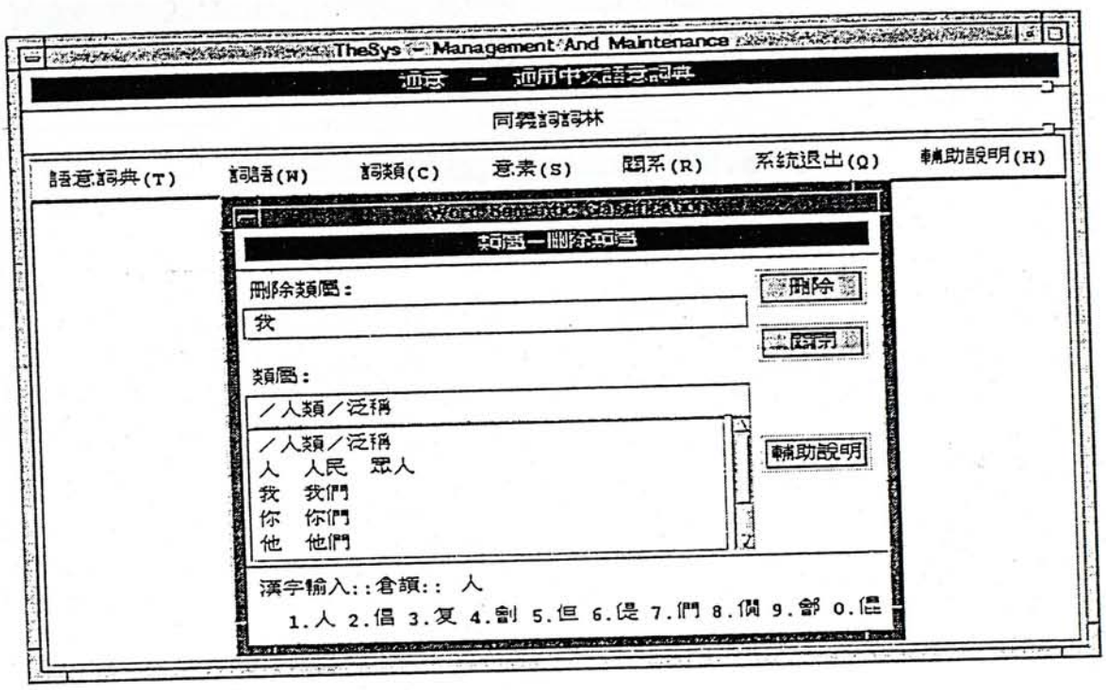

Figure C.10: Delete A Word Class

This window is similar to the above one. But the operation is different. User needn't input the classification name. He just need to select the classification from the scroll list. Such classification will be display on the frame bar. It is noted that the terminal classification can not be selected unless all the entry terms associated to it are deleted. After selecting the classification, press the 刪除 button. The newly adjusted classification list will be displayed in the scroll list.

The word classification browse window is as follows,

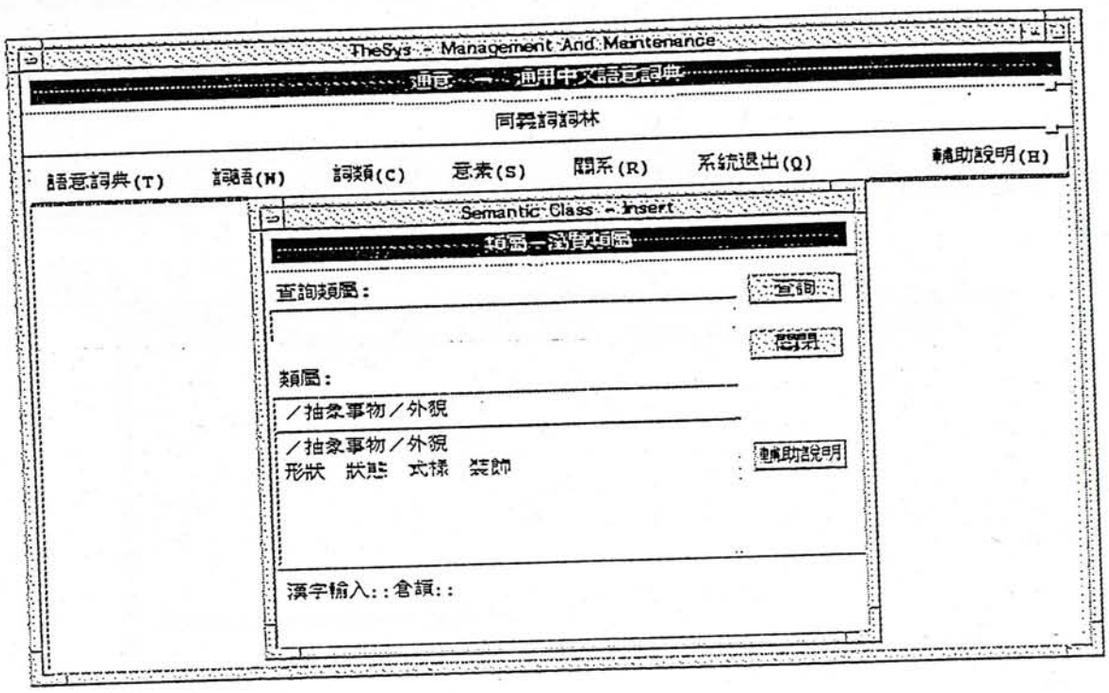

Figure C.11: Browse A Word Class

User enters a semantic classification name into the frame bar and press the 查詢 button. If such a classification exists, its parent and sibling classification will be displayed in the scroll list.

In the 意素 menu, there is only one option 覽(Browse).

The semanteme browse window is as follows,

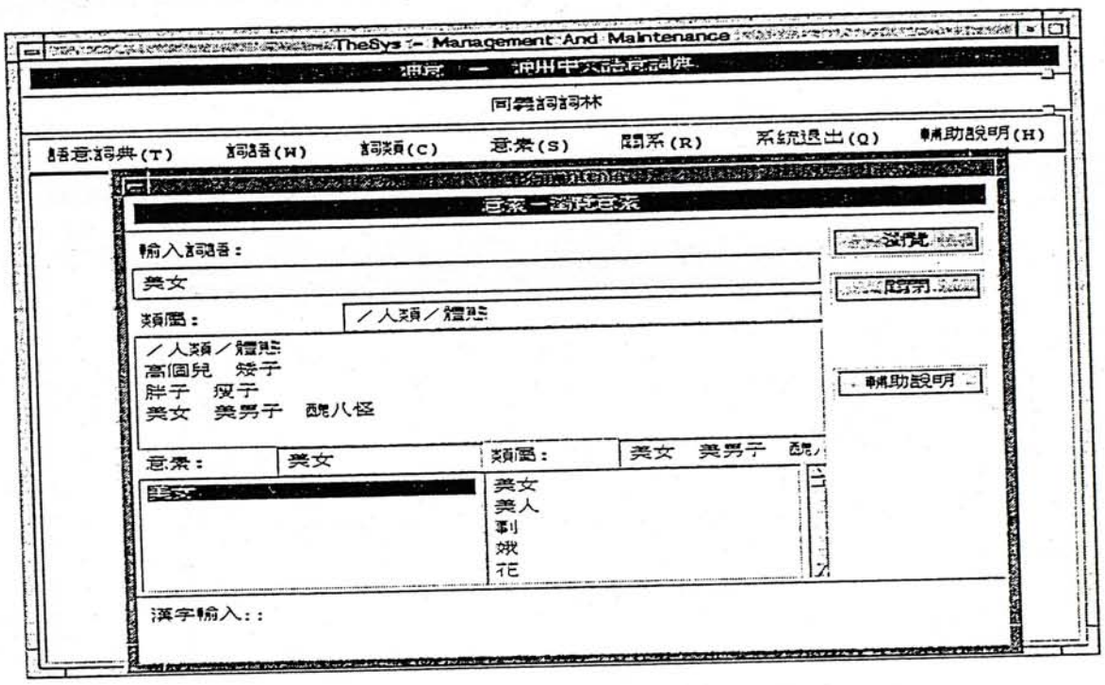

Figure C.12: Browse Semanteme

Actually this window is the same as the entry term browse window. Users needn't know what the representation of a semanteme is, they just need to know the synonyms of such semanteme. The word and word classification which indicates one synonym should be filled in the two frame bars.

The last functional menu is related to the relations which has three options the 類型定義(Relation Type Definition) option, 關系鏈插入(Relationship Link Insertion) option and 關系鏈刪除(Relationship Link Deletion) option.

The relation type definition window is as follows,

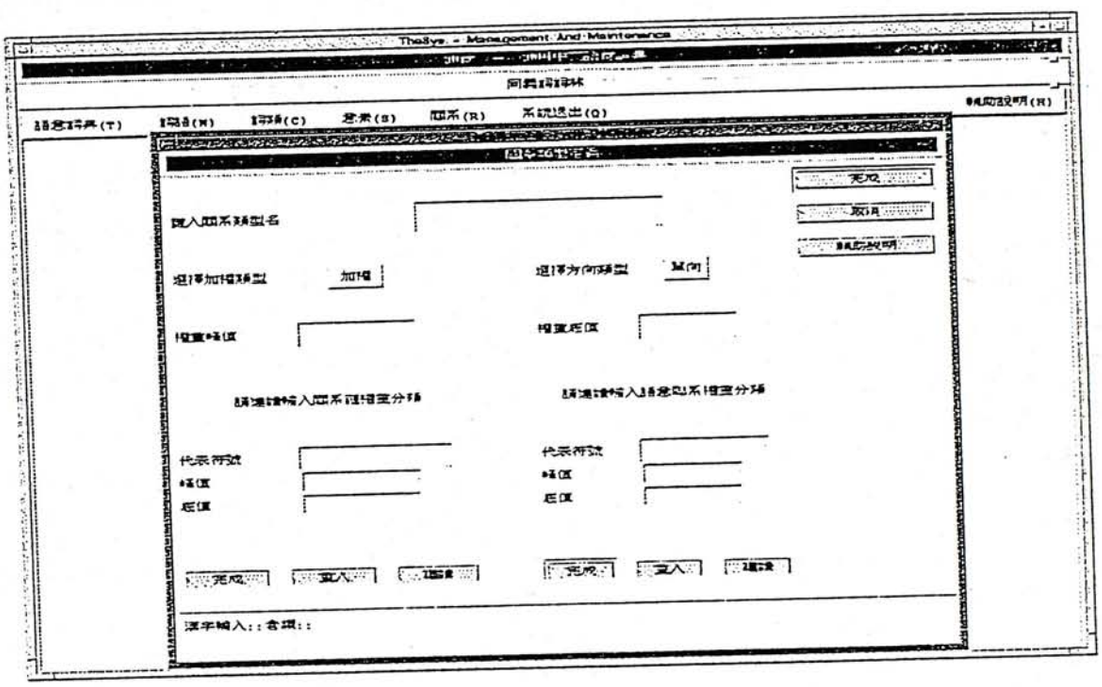

Figure C.13: Define A Relation Type

This window is divided into left and right parts. The left part is a group of action buttons. The right part is used for storing the relation type information. In the right part, the toppest frame bar is used for entering the relation type name. Below the name frame bar are two selection boxes. User can click these boxes to select the relation weight type and traversal direction. Below these two boxes are two frame bar for users entering the top and bottom values of the link weight. Of course, if the weight type is defined as *standard*, user needn't filling in these frame bars. Below these two frame bar are two group of input areas for defining the weight range and distance range. Users enter a pair of bottom and

top values and the symbol representation of such range on the frame bars. By pressing the continue button, user can enter the next range definition. When completed, users click the 完成 button. After filling all the left part, user can click the 完成 button on the right part of the window. If the definition succeed, this window will disappear, else the button will sound. The problem is mostly likely due to the duplicate relation type name. Another problem may be the weight range or distance range are wrongly defined. In both case, user need to redo the defintion work.

The relationship link insertion window is as follows,

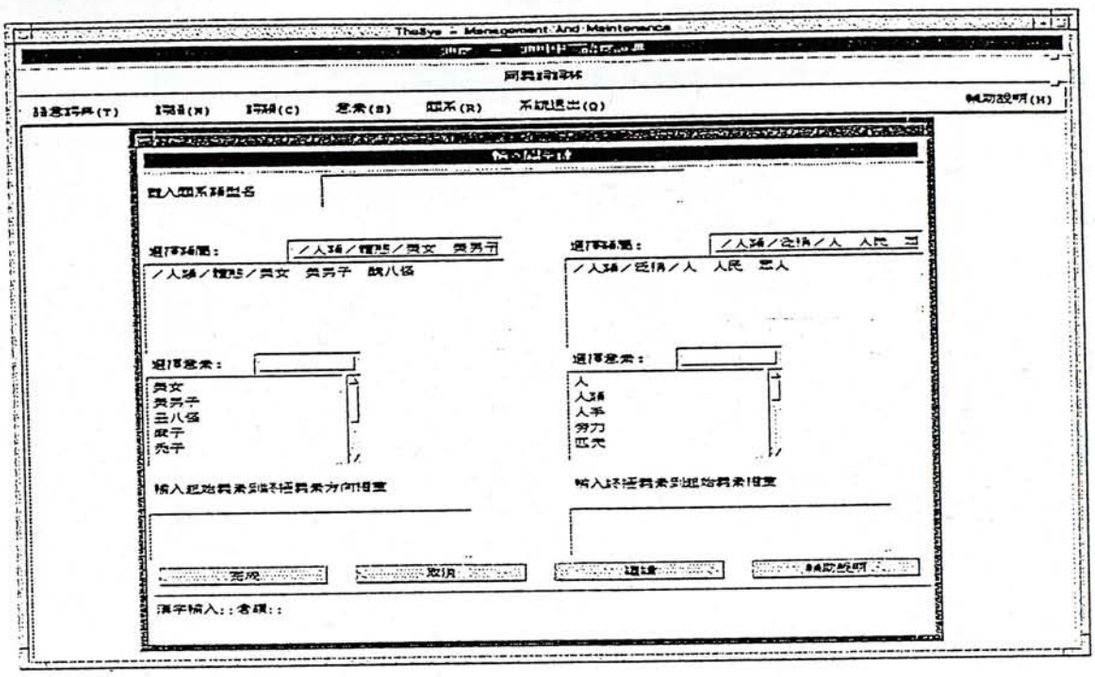

Figure C.14: Add A New Relationship Link

First, users need to enter the relation type name of this relationship link into the toppest frame bar. Second, the owner semanteme and member semanteme of this link need to be selected. For selecting a semanteme, first select the word classification by double clicking the semantic classification. When the classification displayed on the classification frame bar, user can double click the classification so that a list of semanteme will displayed on the semanteme scroll list. Finally, double click the semanteme to make it displayed on the semanteme frame bar. This way, a semanteme is selected. If the weight on relationship link is needed, user can move the mouse cursor to the bottom frame bar which allow user enter the weight value. After filling in these information, user can press the 完成 button. If insertion succeed, this window disappear, else the button sound. The error may be wrong relation type name or wrong assignment of weight value. Users need to redo the insertion work.

The relationship link deletion window is as follows,

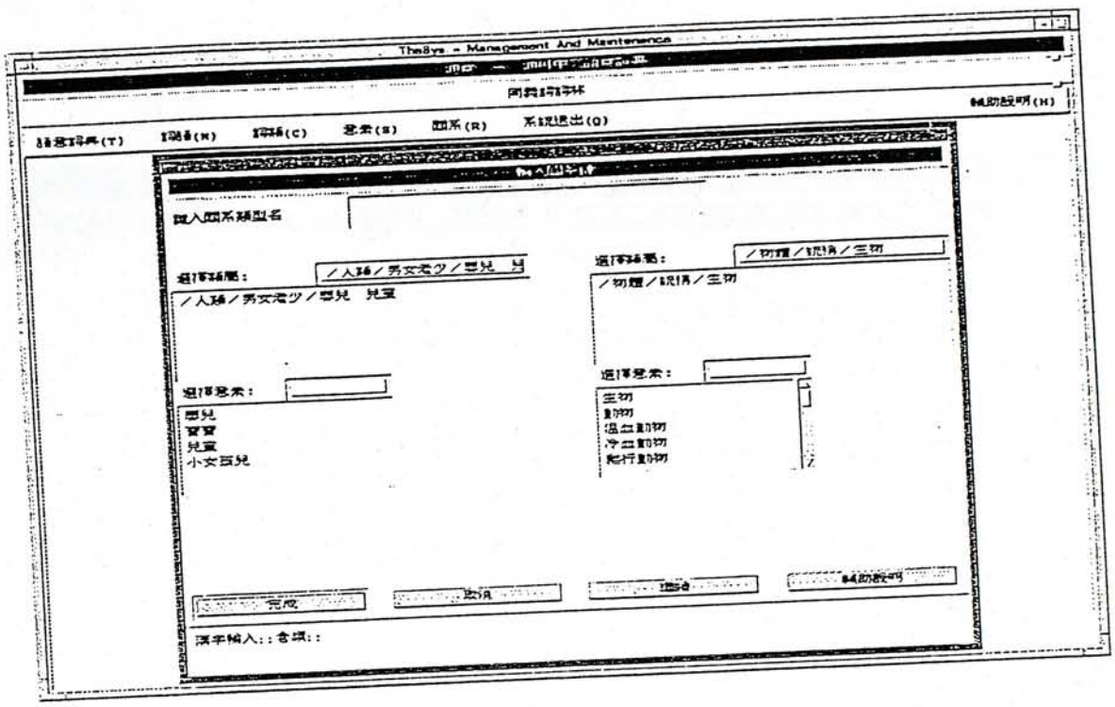

Figure C.15: Delete A Relationship Link

This window is very similar with the relationship link insertion. The only exception is no weight value frame bar. User can follow the above description to enter the relation type name and select the semantemes. It is noted that user must enter the relation type. However, there is no need to select both the owner semanteme and member semanteme. When only one of the semanteme is selected, all the links associated with the semanteme and the relation type will be deleted.

The top window of application user interface is shown as follows.

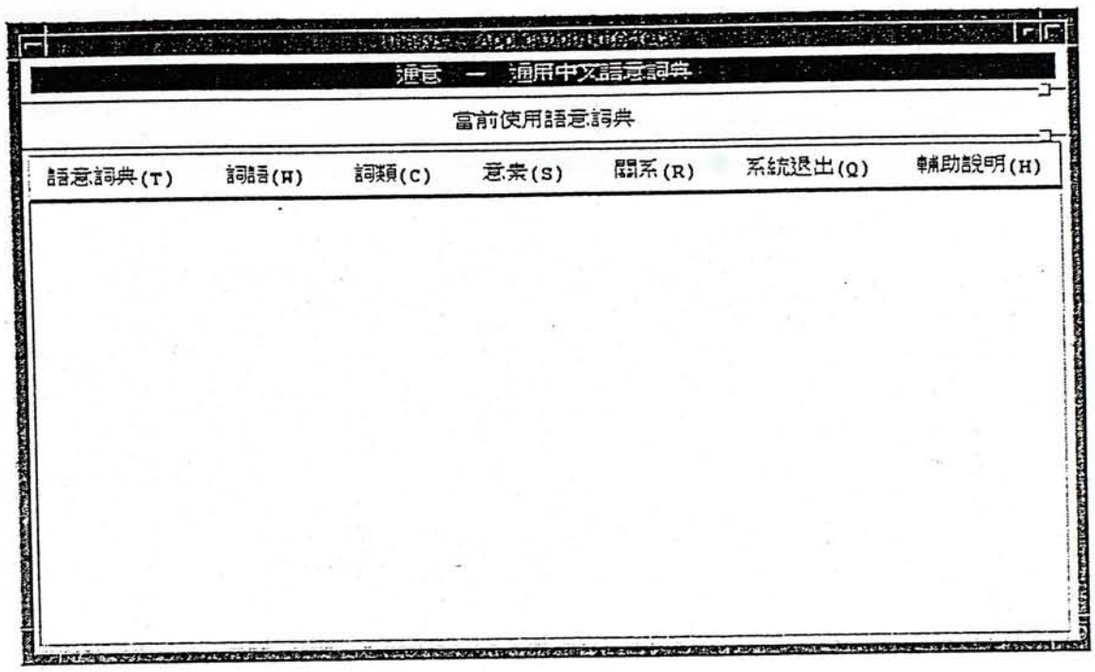

Figure C.16: Application User Interface Top Window

There are seven menus in this window. Under the 語意詞典(Thesaurus) menu, there are two options, 開啓(Open) and 關閉(Close). Under the 詞語 (Word), 詞類 (Word Class) and 意素 (Semanteme) menus, it is samely just one option, 覽 (browse), exists. All the popped up window correspond the above option are the same as those in maintenance module. Under 關系 (Relation), there is also just one option, 相關詞(Related Word). The remain two menus are 系統退出 (Quit) and 輔助說明 (Help).

To select the related word option, we get a window below,

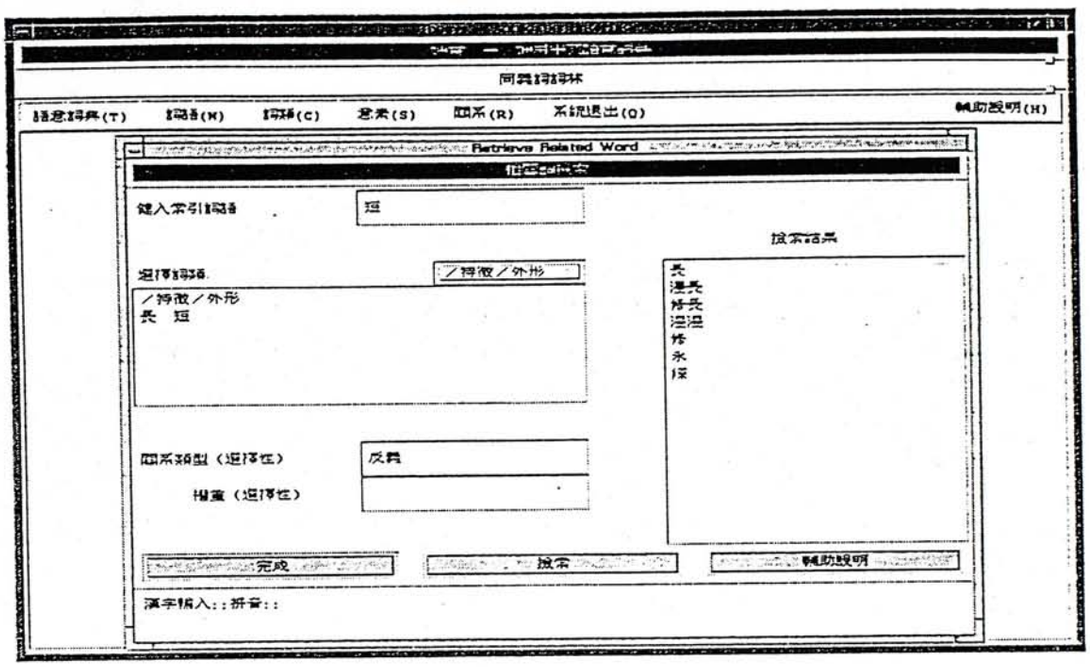

Figure C.17: Retrieve A Related Word

This window is for retrieve the related words of a 索引詞(index word). This index word is defined by the word in the word frame bar and the word classification in the classification frame bar. User need to enter a word on the word frame bar and select the classification of from the scroll list. User can also define the search scale by entering the relation type and distance range. It is noted that the search scale is optional. After filling these information, press the 檢索 button. The search result will be displayed on the left area of this window. If search fail, the area keep empty and the button sound.