ABSTRACT

INTERACTIVE AND BATCH CREATION OF OODB MEDICAL VOCABULARIES

by
Muhammad Arif

Controlled vocabularies are becoming popular for knowledge representation and querying. They are particularly helpful in the medical field since they can unify disparate terminologies and provide information in a compact, comprehensible manner. In this thesis, we present a mechanism to create OODB controlled medical vocabularies from flat-file format. We also describe a tool by which a user can interactively create, edit and browse the vocabulary. For better understanding of the structure of the vocabulary we designed our interface as a graphical editor and browser. The user of this interface will typically be a medical expert who either wants to add new concepts to the vocabulary or create a new vocabulary from scratch. We first describe our approach for creating the vocabulary from an existing flat-file format by batch processing. We then present the software architecture and design of an interactive vocabulary creator (IVC).

INTERACTIVE AND BATCH CREATION OF OODB MEDICAL VOCABULARIES

by
Muhammad Arif

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

October 1997

APPROVAL PAGE

INTERACTIVE AND BATCH CREATION OF OODB MEDICAL VOCABULARIES

Muhammad Arif

Dr. Yehoshua Perl, Thesis Advisor — Date
Full time Professor of Computer and Information Science, NJIT

Dr. James Geller, Thesis Co-advisor — Date
Director of Artificial Intelligence and OODB Laboratory
Associate Professor of Computer and Information Science, NJIT

Dr. Michael Halper, Committee Member — Date
Assistant Professor of Math and Computer Science,
Kean College of New Jersey

BIOGRAPHICAL SKETCH

Author: Muhammad Arif

Degree: Master of Science in Computer Science

Date: October 1997

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1997

- Master in Computer Science,
  Department of Computer Science,
  University of Karachi, Karachi, Pakistan, 1996

Major: Computer Science

This work is dedicated to
my parents, friends and all those who
helped me in my accomplishments

ACKNOWLEDGMENT

I would like to thank Dr. Y. Perl, Dr. J. Geller and Dr. M. Halper for this opportunity to conduct research under their able guidance. Their continuous interest and encouragement have contributed significantly to the work presented in this thesis. It has been an enriching experience for me.

I would also like to thank all my colleagues in the laboratory. A special thanks to the members of my group; without their hard work and dedication this work would not have been possible.

TABLE OF CONTENTS

Chapter Page

Chapter		Page

Chapter | Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A Controlled Vocabulary is an explicit specification of a subject. It is a formal and declarative representation which includes the terms in a subject area and the logical statements that describe what the terms are and how they are related to each other. Vocabularies therefore provide a way of representing and communicating knowledge about some topic which leads to a uniform way of knowledge sharing and reuse. The medical field is one of the most rapidly growing fields in terms of concepts(terms) in one subject. That is why Controlled Medical Vocabularies (CMV) are becoming more and more popular in medicine.

A Semantic network is a tool for modeling vocabularies. Due to the huge number of terms available in a medical vocabulary, the size of semantic networks of CMVs is typically large. That means that we have to organize huge amounts of data in such a way that they can be stored and retrieved efficiently . Choosing a paradigm for a computerized storage of a vocabulary, is a difficult task. As mentioned before, vocabularies are for reuse and sharing of data. Object-Orientation is a paradigm which proved itself as a good tool in terms of re-usability and easy shareablity. We mapped the semantic network of a major medical vocabulary into an Object Oriented Database which we named OOHVR (Object Oriented Health Vocabulary Repository). Chapter 2 deals with modeling details of OOHVR.

After modeling the vocabulary, the most tedious task is to convert a huge semantic network which is initially available in a flat-file format to our OODB. We designed a Schema Generator which reads the semantic network representation as a set of flat files and generates a schema creating code. The preprocessor is a part of the Schema Generator which takes a schema dependent format flat-files and generates common format flat files. These common format flat files are fed to a code generator which generates C++ code for schema generation. Chapter 3 deals with all implementation and design details of preprocessor.

As new ideas and concepts are developed every day, we felt a need for an interactive Creator which enables a user to create vocabularies from scratch. Choosing an interface and development system is an important task which is discussed in sections 4.1 and 4.2 while our proposed system architecture is discussed in section 4.3. Back-end and front-end design issues are discussed in 4.5 and 4.6. We discuss possible future enhancements in IVC in the last chapter.

APPENDIX A contains the CGIs source code for the Creator.

APPENDIX B contains the Java object descriptions and documentation with the source code.

APPENDIX C contains the description of the APIs used in the system.

CHAPTER 2

AN OVERVIEW OF OOHVR

2.1 Need for Health Vocabulary Systems

Effective and efficient delivery of health care requires accurate and relevant clinical information. This is true for the individual doctor caring for the individual patient, as well as the health care organization concerned with measuring outcomes and ensuring cost effective care. Furthermore it is recognized that patient-centered clinical information systems, integrated with decision support and other systems, are the key to high quality clinical information. However developing such systems has been proven difficult and many problems remain. Perhaps the most pervasive and the most important of these problems is that of the clinical terminology or language that is used to represent the information. Advanced clinical systems require advanced terminology systems which must be:

- Comprehensive and sufficiently detailed in content and structure for use in clinical medicine.
- supported across a wide range of natural language communities, both professional and geographical;
- maintainable and extendible, with realistic human effort which the computer must actively support
- well suited to supporting computer-based information systems and hence formally sound.

Clinical terminologies are large, complex, and diverse. For example the details required in a patient's medical record which is used to support the daily care of the patient, are far greater than for an epidemiological study or routine hospital statistics. Furthermore, different users in different clinical settings require different but consistent views of that information. Clinical medicine is inherently large and complex and yet clear separations between medical specialties are not possible. Hence anything we do to represent the detailed record of clinical medicine will also be large and complex in one way or another. HEALTH VOCABULARY SYSTEM's goal is to make this complexity manageable. As the demand has grown for wider coverage and new uses, the traditional techniques of coding and classification have been proven inadequate. They tend to 'explode' in size and become unwieldy, inconsistent and unmanageable.

Advanced clinical systems need more than just terminologies, they need computer systems which can provide a sophisticated and appropriate set of terminological services, allowing applications to be developed to use whatever coding system or natural language, local circumstances demand. Clinical application developers can therefore concentrate on the clinical tasks they must support, knowing that not only are the details of coding and classification abstracted for them but that they have access to a powerful model of clinical information to support their dialogue with clinical users. OOHVR addresses this challenge to develop computer systems that provide powerful terminological services.

2.2 A Medical Vocabulary as a Semantic Network

Semantic networks are a technique for representing knowledge. As with other networks, they consist of nodes with links between them. The nodes in a semantic network represent concepts. A concept is an abstract class, or set, whose members are things that are grouped together because they share common features or properties. The "things" are called instances of the concept. For example, Femur is a concept representing the set of all femurs in the world; John Smith's left femur is an instance of the concept Femur.

Links in the network represent relations between concepts. Links are labeled to indicate which relations they represent. Links are paired to represent a relation and its inverse relation. For example, the concept Femur is related to the concept Upper Leg with the relation has-location. The inverse of has-location is the relation location-of, which relates Upper Leg to Femur.

2.3 Object Oriented Paradigm Choice for Storing Semantic Networks

Object Oriented Databases are good tools for conceptual modeling in information technology. There are a number of reasons why the Object Oriented database paradigm is a good choice for modeling a vocabulary's semantic network. In applications where external agents such as intelligent information locators, decision-support systems, and end-user browsers access the knowledge stored in the vocabulary, transparent and concurrent access to it is necessary[2]. OODB systems provide the traditional access support of Database systems and offer a "low impedance" pathway [14] to the network. As a matter of fact, Object Oriented programming languages are increasingly used in the industry so an OODB can be easily accessed through them. Declarative languages are also available to access the OODB like OSQL in ONTOS case and a path language XSQL [8]. The typical OODB system's repertoire of modeling constructs neatly captures many modeling features of semantic networks used to describe a typical controlled vocabulary [9].

2.4 Modeling InterMed and MED into OOHVR

In this section, we first describe the general structure of a CMV. After that, we go on to present our methodology for modeling such a system as an OODB. Our representation of the InterMED, an existing CMV, as an OODB is called the OOHVR and is currently available in the context of ONTOS.

2.4.1 Structure of a CMV

A common formalism used in building a CMV is the semantic network, each of which nodes in that context is a medical concept. All nodes can exhibit properties which come in two kinds: (1) Attributes whose values are of data types (such as integer or text string), and (2) relationships whose values are references to other concepts in the vocabulary. For a concept V, we will use $P(V)$ to denote the set of all of V's properties.

Each node in a CMV is defined with the attribute *name* that holds the concept's associated *term* (i.e., textual denotation). Note that we distinguish the notions of "concept" and "term." A (medical) concept is a node in the CMV, while a term is simply a string used as the node's name [6]. Sometimes a term is called the printable value of a concept.

In [3, 4], a set of design criteria (sometimes referred to as "Cimino's rules") was proposed that all CMVs should satisfy in order to increase their utility. These criteria are: Domain completeness, non-redundancy, synonymy, non-vagueness, non-ambiguity, multiple classification, consistency of views, and explicit relationships. As an example, non-ambiguity requires that a given medical concept be represented by a unique node even if it has several synonymous names. If multiple nodes representing the same concept exist, then these should be folded into one concept that holds the primary name (i.e., the concept's term) and any secondary names (i.e., synonyms). The related synonymy criterion, in fact, states that any concept must be accessible via its known synonyms, all of which should be stored with the concept. Due to this, each concept in the CMV is assumed to have the property *synonyms* whose value is the entire set of acceptable secondary names for the concept. Let us point out that it is strictly a design decision as to which name is primary and which others are secondary.

The concept subsumption (IS-A) hierarchy is a fundamental aspect of a CMV. Structurally, it is an acyclic collection of IS-A links, each of which connects a subconcept to a related superconcept. The multiple classification criterion requires that the IS-A hierarchy be a directed acyclic graph (DAG). In other words, it must be possible for any concept to have multiple parents.

The IS-A hierarchy plays two important roles in the vocabulary. First, it supports reasoning in the form of subsumption-based inferences. For example, if a user asks if a patient is on antibiotics, then this can be answered in the affirmative by consulting the CMV if we know the patient is taking Tetracycline because the concept **Tetracycline** IS-A **Antibiotic**. The second aspect of the IS-A hierarchy is inheritance: A subconcept inherits all the properties exhibited by its superconcepts (which themselves may have inherited the properties from their ancestors). For example, the concept **Sodium Test** IS-A **Test**, and therefore the set of properties of **Sodium Test** is a superset of the properties of **Test**. If a concept has multiple parents, then it could potentially inherit properties from each of them. Another assumption that we make following [3] is that the CMV satisfies the following rule:

Rule (Uniqueness of Property Introduction): A given property x can only be introduced at one concept in the vocabulary.

Other concepts needing that same property must be defined as descendants of that concept and obtain the property via inheritance. Note that if there is a need to introduce the same property p in several independent nodes, then an "artificial" node can be created to define p, and the other nodes can be made children of this new node [2].

A CMV is also taken to be singly rooted with respect to the IS-A hierarchy. We will refer to the root concept as **Entity**. Of course, there is no loss of generality because **Entity** can be artificially introduced into the vocabulary if need be. Note that **Entity** is defined with the property *name* (that holds a concept's primary term) and *synonyms* (that holds a concept's acceptable secondary names). Via inheritance, all other nodes in the CMV have these properties, too.

2.4.2 Representation of a CMV as an OODB

2.4.2.1 Partitioning the CMV into Areas

Our modeling of a CMV as an OODB is based on a structural abstraction of the vocabulary network. The network is partitioned into groups of concepts such that all the concepts in a single group have the exact same set ofproperties. We refer to such groups as *areas* of the CMV [9]. The partitioning of the CMV into areas closely follows the property introduction and inheritance patterns of the IS-A hierarchy, and in fact can be done automatically in a top-down fashion according to a number of different cases. In the statement of those cases, we will be using the following definitions.

Definition (Property Set of an Area): For an area A, $P(A)$ denotes the set of properties of any (and all) of its constituent concepts.

Definition (Property-introduction Node): A concept at which one or more new properties are introduced into the CMV is called a property-introduction node.

An example of such a concept is the vocabulary's root **Entity** which, among other things, introduces the property *name* that is used to hold the term associated with a given concept. Another example is the concept **Lab Diagnostic Procedure** which introduces the relationship *has-specimen*.

Definition (Root of an Area): A concept V residing in area A is called a root of A if V has no parent in the CMV (i.e., V is the concept **Entity**, the root of the entire CMV) or V's parents all reside in areas other than A.

Definition (Property-Introduction Area): An area with a root that is a property-introduction node is called a property-introduction area.

An example of a property-introduction area is the one to which **Entity** belongs. Recall that **Entity** was defined to introduce the property *name*, among others. Another is the area rooted at **Lab Diagnostic Procedure**.

It can be shown that a property-introduction area always has exactly one root. (We will not prove this result here. Refer to [10] for the details.) The other kind of area, called an *intersection area* (defined below), can have more than one root. If an area has a single root, then the area is named after that concept. The area containing **Entity** as its root is called "Entity Area." The area whose root is **Lab Diagnostic Procedure** is named "Lab Diagnostic Procedure Area."

The partitioning of the network into areas was originally described as a two-step process where the second step was used to overcome a problem introduced by the first step [9]. Below, we present the solution in recursive form, which serves to unify the presentation. To reiterate, the process of identifying areas is top-down starting at the level of the children of **Entity**. The base of the recursion is the special case defining "Entity Area."

For a concept V (not equal to **Entity**), membership in an area is determined by the following two major cases.

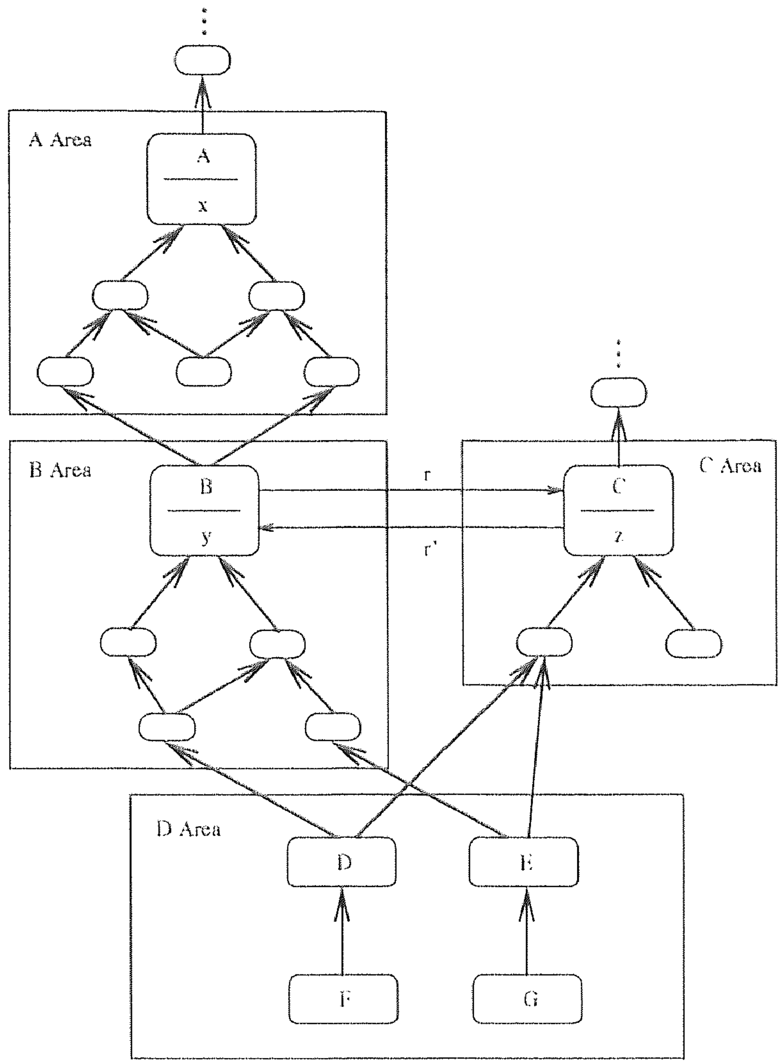

Figure 2.1 Four areas of a CMV

Case 1: *V is a property-introduction node.*

In this case, V belongs to a new area that differs from all areas already identified. In fact, because V is a property-introduction node, the new area is a property-introduction area. As can be shown, V is the one and only root of this new area, so the area is designated "V Area." Three example property-introduction areas, A Area, B Area, and C Area, are shown in Figure 2.1. The concepts in the figure are represented as rectangles with rounded edges, while the IS-A links are drawn as thick, unlabeled arrows directed from the subconcept to the superconcept. The only concepts in those three areas with their names displayed are A, B, and C, the respective roots. The node A introduces the attribute x; B, the attribute y; and C, the attribute z. B also defines the relationship r (drawn as a labeled, thin arrow) that is directed to C, which, on the other hand, introduces relationship r', the converse of r.

Case 2: *V is* not *a property-introduction node.*

Here, there are two major sub-cases.

Case 2.1: *V has a single superconcept W.*

In this situation, V is in the same area as W. Recall that the CMV was defined to be singly rooted with respect to the IS-A hierarchy. Therefore, every concept (except for **Entity**) has at least one superconcept.

Case 2.2: *V has multiple superconcepts $W_1, W_2, \ldots, W_n$ $(n > 1)$.*

Here, again, there are two additional sub-cases. Before stating these, we will need the following definition.

Definition (Intersection Node): Let V be a concept having multiple superconcepts $W_1, W_2, \ldots, W_n$ $(n > 1)$. V is called an intersection node if the following condition holds: $\forall i\colon 1 \leq i \leq n$, $P(V) \neq P(W_i)$. That is, the set of properties of V differs from all of its parents' sets of properties.

We use the designation *intersection node* because V lies at the junction of (at least) two independent inheritance paths. With this new kind of node, we also have:

Definition (Intersection Area): An area with a root that is an intersection node is called an intersection area.

Case 2.2.1: *V is* not *an intersection node. That is, V has the exact same set of properties as at least one of its parents, say, W_i.*[1]

[1]Formally, we can state this condition as: $\exists i\colon 1 \leq i \leq n$ such that $P(V) = P(W_i)$.

In this case, V is in the same area as W_i. Note that all other parents with the same set of properties as W_i are in W_i's area as well.

Case 2.2.2: *V is an intersection node.*

Then V belongs to an area that differs from all the areas of its parents. By definition, V is a root of its area, and hence the area is an intersection area. It is possible that this intersection area might already have been identified by a previous application of Case 2.2.2, so it is necessary to scan all existing (intersection) areas to determine V's membership. If $P(V) = P(A)$ for some area A already identified, then V is a member of A. Otherwise, V defines a new intersection area that differs from all known areas. Since V is the first concept in this new area, it is named "V Area."

As we mentioned, unlike a property-introduction area, an intersection area can have more than one root. This is demonstrated in Figure 2.1 by the intersection area called "D Area" which contains four concepts, D, E, F, and G. Its roots are D and E, both of which have two parents, one residing in B Area and the other in C Area. We have assumed that the concept D was identified as a member of this area first, and hence the area was named D Area. If E had been examined before D, then the area would have been designated E Area. The concepts F and G are members of D Area by virtue of the fact that they are children of D and E, respectively. F and G are not roots of D Area. It will be noted that none of the concepts in D Area has any intrinsic properties. All properties are inherited. It is not possible for an intersection area to have as one of its nodes a property-introduction node since such a node would define a new area with new properties.

Before continuing, let us summarize, without proof, a few important properties that hold for areas.

1. An area is either a property-introduction area or an intersection area. That is, there are no other kinds of areas.

2. All areas have at least one root.

3. A property-introduction area has exactly one root.

4. An intersection area can have multiple roots.

5. An intersection area cannot contain a property-introduction node.

2.4.2.2 OOHVR Schema

In the OODB-version of the vocabulary, which we refer to as the OOHVR, each concept is represented by a unique object. The OOHVR's schema is constructed automatically after the identification of all areas. There is a one-to-one correspondence between the areas in the CMV and the classes in the OOHVR's schema. That is, one class is defined to represent one area. The extension of a given class (i.e., its entire set of instances) is identical to the set of concepts in the corresponding area in the CMV. Due to this, we refer to the classes in the OOHVR schema as *area classes*. If the area happens to be a property-introduction area, then we have a *property-introduction class*. Likewise, for an intersection area, there is an *intersection class*. Let us point out that, in an OODB schema, classes are defined for the purpose of describing a set of objects whose structure and behavior are the same. This is indeed what is done in our mapping. The instances of one class are exactly all those concepts which reside in a single area which, by definition, contains all concepts exhibiting identical properties.

The intrinsic properties of a property-introduction class are defined to be exactly those introduced by the root concept of its corresponding area. In addition, all the concepts in a property-introduction area must have the properties inherited

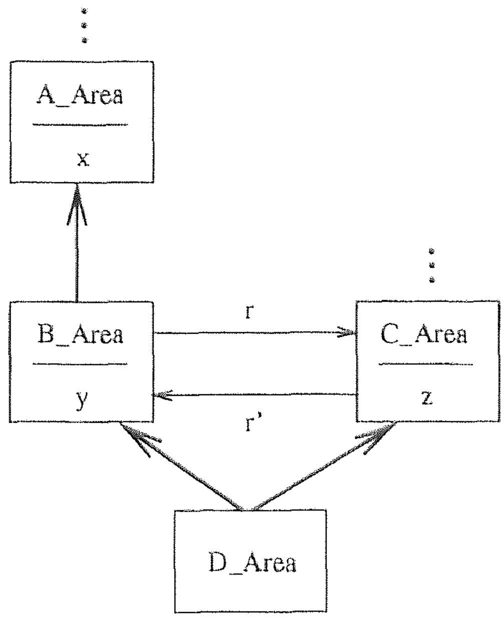

Figure 2.2 The area classes for the areas in Figure 2.1

by the root from its parent(s) in the CMV. To capture this situation, the property-introduction class is placed in subclass relationships with those other area classes to which the parents of the root belong. In this way, the property-introduction class obtains all necessary properties: Some are defined intrinsically, while the others are inherited from other classes. It should be noted that even though the root of a property-introduction area contributes both its name (via the area name) and its intrinsic properties to the area class definition—and in this sense the class itself denotes that root concept—there still exists an object that directly represents the root in the extension of the class.

In Figure 2.2, we illustrate the above by showing the classes, *A_Area*, *B_Area*, and *C_Area*, that respectively represent the corresponding areas from Figure 2.1. The classes are boxes with their names and attributes written inside. The labeled arrows are the ordinary relationships. The subclass relationship is drawn as a thick arrow pointing from the subclass to the superclass. The ellipses indicate the omission of the subclass relationships that *A_Area* and *C_Area* would have in an expanded drawing. All property-introduction classes—and, indeed, all intersection classes—have at least one subclass relationship. The only exception to this is the class *Entity_Area* which is the root of the OOHVR schema.

Since an intersection area, by definition, does not contain any property-introduction nodes, and, in fact, all properties of its concepts are obtained via inheritance, an intersection class does *not* introduce any properties of its own. Instead, it is defined to be a subclass of all other area classes (potentially intersection classes themselves) which contain one or more parents of its root(s). Again, an intersection area may have more than one root. Let us also note that an intersection class *always* exhibits multiple inheritance, i.e., it inherits from two or more superclasses.

Referring to Figure 2.2 again, we see the intersection class *D_Area* representing D Area. *D_Area* is a subclass of both *B_Area* and *C_Area* because its roots (D and E) have parents residing in both those respective areas. As can be seen, *D_Area* has no intrinsic properties defined for it.

The final aspect of the mapping which deserves special care is the IS-A hierarchy of the CMV. It is appropriate to view the IS-A link as a generic property, one featured by all concepts, aside from the ordinary attributes and relationships. Indeed, all concepts can—and indeed *must*—have some IS-A connections to other concepts (except for **Entity**). Therefore, in the original network, the root concept **Entity** can be considered to be endowed with the multivalued relationship "IS-A" that provides all nodes with the capability of making superconcept connections to other concepts. In the mapping, this translates to the inclusion of the multivalued, reflexive relationship *IS-A* in the definition of the class at the top of the schema, namely, the class *Entity_Area*. In this way, all concepts (objects) in the OOHVR can have their required IS-A connections, too.

It is important to note that the CMV's IS-A hierarchy is different from the subclass hierarchy of the OODB schema, though, to be sure, the latter is derived from the former. An IS-A link between two concepts in the CMV indicates that one is a subconcept (or, vice versa, a superconcept) of the other. A subclass connection between a pair of area classes in the schema denotes the fact that the set of properties exhibited by the concepts of one area is a superset of the properties exhibited by the concepts in the other. Of course, as we have just discussed, the CMV's IS-A hierarchy does appear in its entirety at the instance-level of the OOHVR with respect to the relationship *IS-A* appearing at *Entity_Area*.

In Figure 2.3, we show the entire OOHVR schema which comprises 39 classes (29 property-introduction classes, 10 intersection classes) and 50 subclass relationships. The schema was generated automatically by software described in [10]. Overall, it provides a structural abstraction of the underlying network of the CMV. Concepts with like properties are grouped into areas which in turn are modeled as object classes; the concepts themselves become the objects of the OODB. We refer to this kind of schema as a *network abstraction schema* [9]. It is important to point out that this schema represents a substantial reduction in size from the original CMV. The InterMED contains about 3,000 concepts, while the OOHVR schema has merely 39 area classes—approximately a 75-to-1 reduction. This ratio is high since, by the "Uniqueness of Property Introduction" Rule, each property can be introduced only at one node in the CMV. Thus, the number of different properties is an upper-bound on the number of property-introduction areas and their corresponding classes in the schema.

The schema can aid in the comprehension of the vocabulary and help a vocabulary administrator uncover problems in the modeling [7]. The same methodology was also carried out with respect to the entire MED which contains

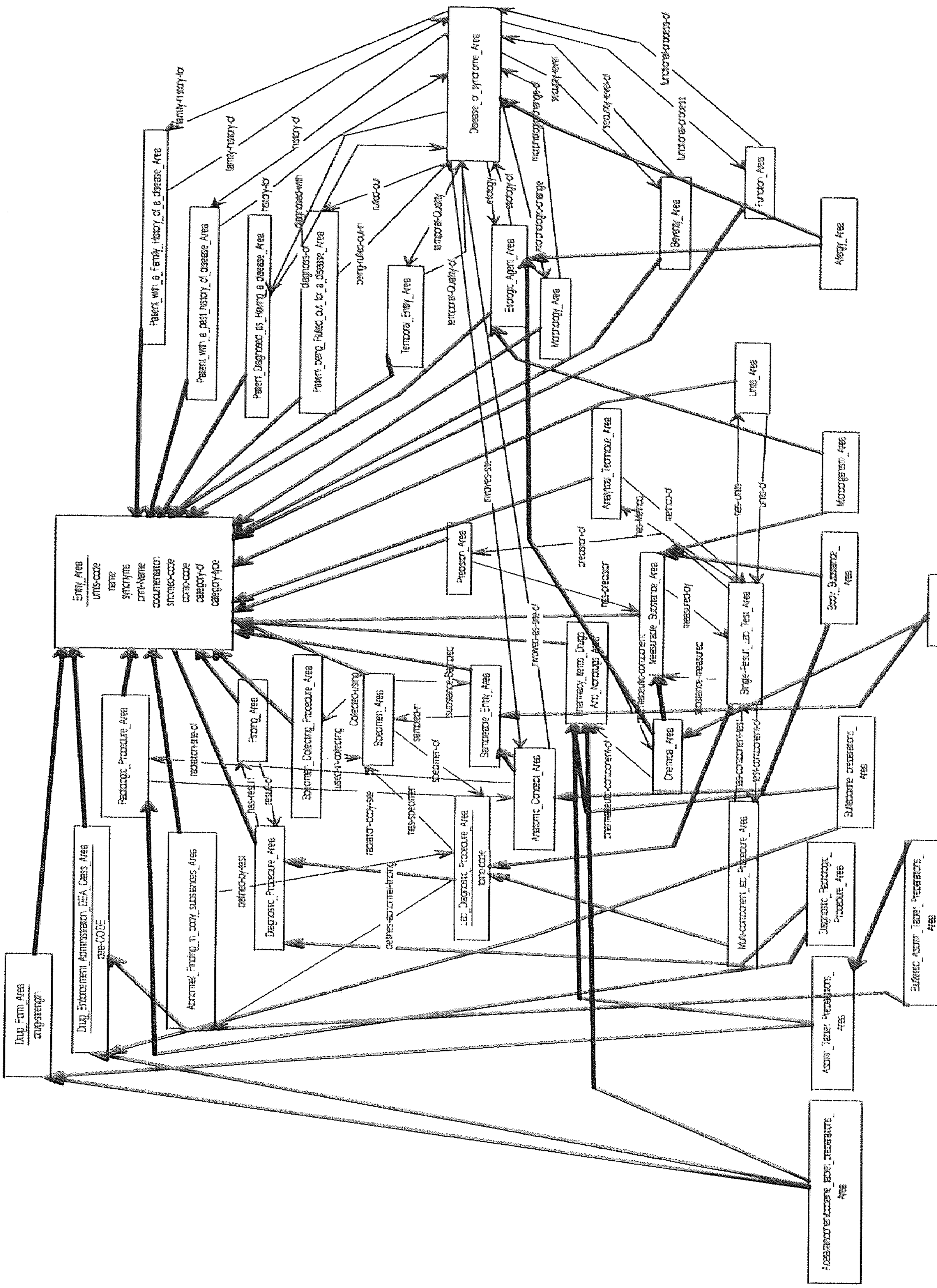

Figure 2.3 OOHVR Schema

approximately 46,000 concepts. There, the schema comprised about 90 classes, and the ratio was about 500-to-1.

2.5 Modeling CHREF-I and CHREF-II into OOHVR

2.5.1 CHREF-I

The National Drug Code system was established as an essential part of an out-of-hospital drug reimbursement program under Medicare. The purpose of NDC was to provide a universal product identifier for prescription drugs. It contains information about most frequently prescribed drugs.

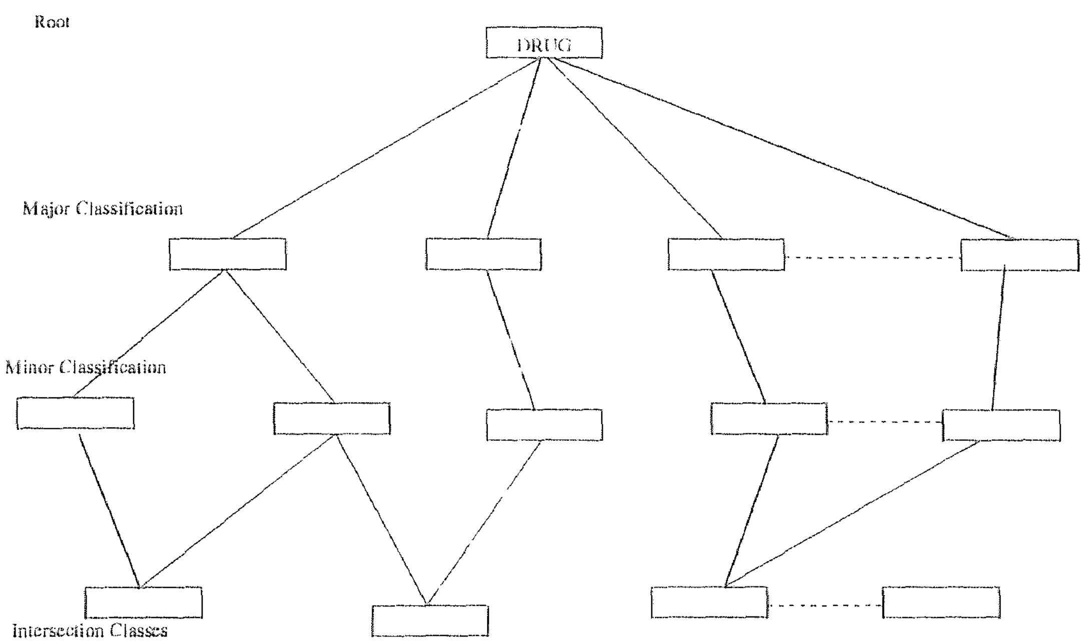

Figure 2.4 CHREF-I schema structure

The Directory is originally composed of four different sections indexed by different keys. We analyzed the directory data taken from a hospital. The Drug Classification provided the basis to build the schema. The classification places the drugs and their NDC codes into a hierarchical structure. We took that structure as our structural schema, and name the database CHREF-I. The schema contains a four level hierarchy, the root of which is Drug and the second level is major drug classes, which places drugs in therapeutic or pharmacological classifications. The major classification is further divided into minor classes which contain actual instances of drugs. In the fourth level of the hierarchical structure, classes exist which actually are inheriting from more than one class in the third level. In Figure 2.4 the general sketch of the schema is shown.

2.5.2 CHREF-II

After analyzing the data in the NDDF that we recieved from First Data Bank for NDC in the form of a relational database, we found that drugs can also be classified by their HIC (Hierarchical Ingredient Code) which actually maintains the Drug classification according to their ingredients.

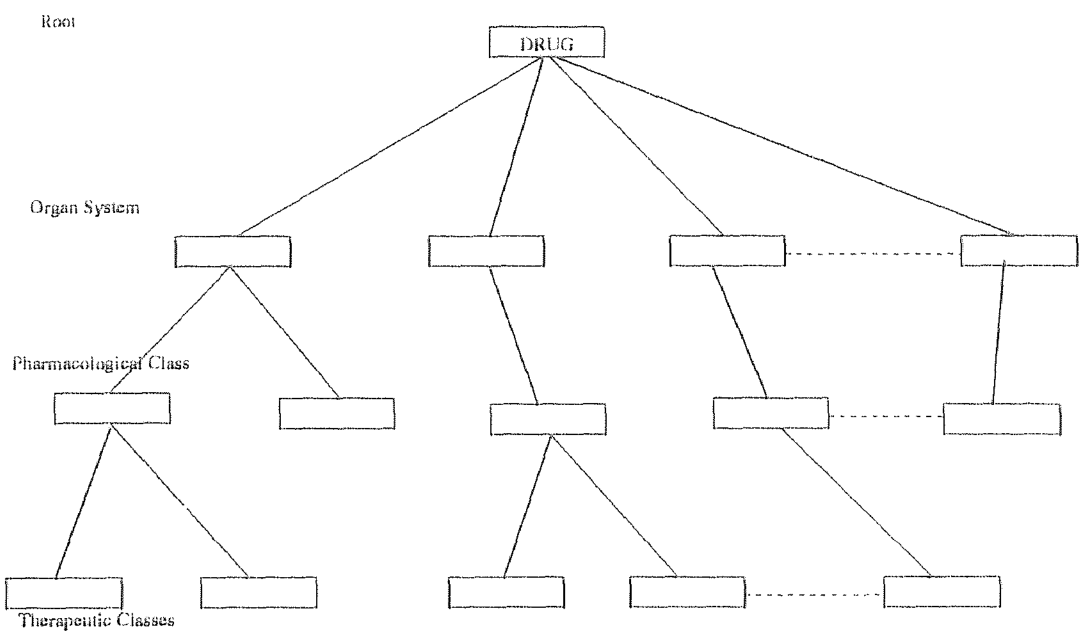

Figure 2.5 CHREF-II schema structure

This hierarchy was also maintained as a three level tree. To map the classification to OOHVR we built a four level schema, starting from DRUG as the root. The second level represents the Organ system, the third level is the pharmacological class of the drug and the fourth level represents the therapeutic class. No intersection classes were found in this hierarchy since it is a tree. We built the schema under the name CHREF-II.

The next chapter will present detailed discussions about the design of pre-processors for all of the above mentioned Schemas.

CHAPTER 3

PREPROCESSOR DESIGN AND IMPLEMENTATION

3.1 Practical Realization of the OOHVR Implementation

Our method of mapping a controlled vocabulary onto an object-oriented database can be applied not only to a medical vocabulary, but to any semantic network-based vocabulary, as long as the "uniqueness of property introduction" rule is satisfied. For the medical domain we used in our research, OOHVR can be built from scratch. Existing vocabulary sets, e.g. InterMED or MED, can be loaded into the OOHVR as well.

3.1.1 The Need for the OOHVR Generator

For loading existing vocabularies which are usually stored in different formats, the preferred approach would be to design a universal loader. Otherwise, for each vocabulary format, we would have to write a corresponding program to load it into the OOHVR. We approximated the universal loader, which we call the *OOHVR generator*, by modular design. For loading a different format vocabulary, only one program component, the *Preprocessor*, in the *OOHVR generator* needs to be rewritten and the rest of the program modules can be reused.

The MED and InterMED are too large, and the OOHVR schema is too complex, to consider creating the schema and the data definition language (DDL) statements for generating it by hand. Rather, it is necessary to use a program that transforms the MED or InterMED into an appropriate set of DDL statements. Even if one would consider creating the schema manually, it is expected that the database and even the schema will change on a regular basis, as the MED and InterMED are constantly growing. In addition, the task of dealing with the schema is made more difficult by the length of many of the class names. Currently, the longest class name has 47

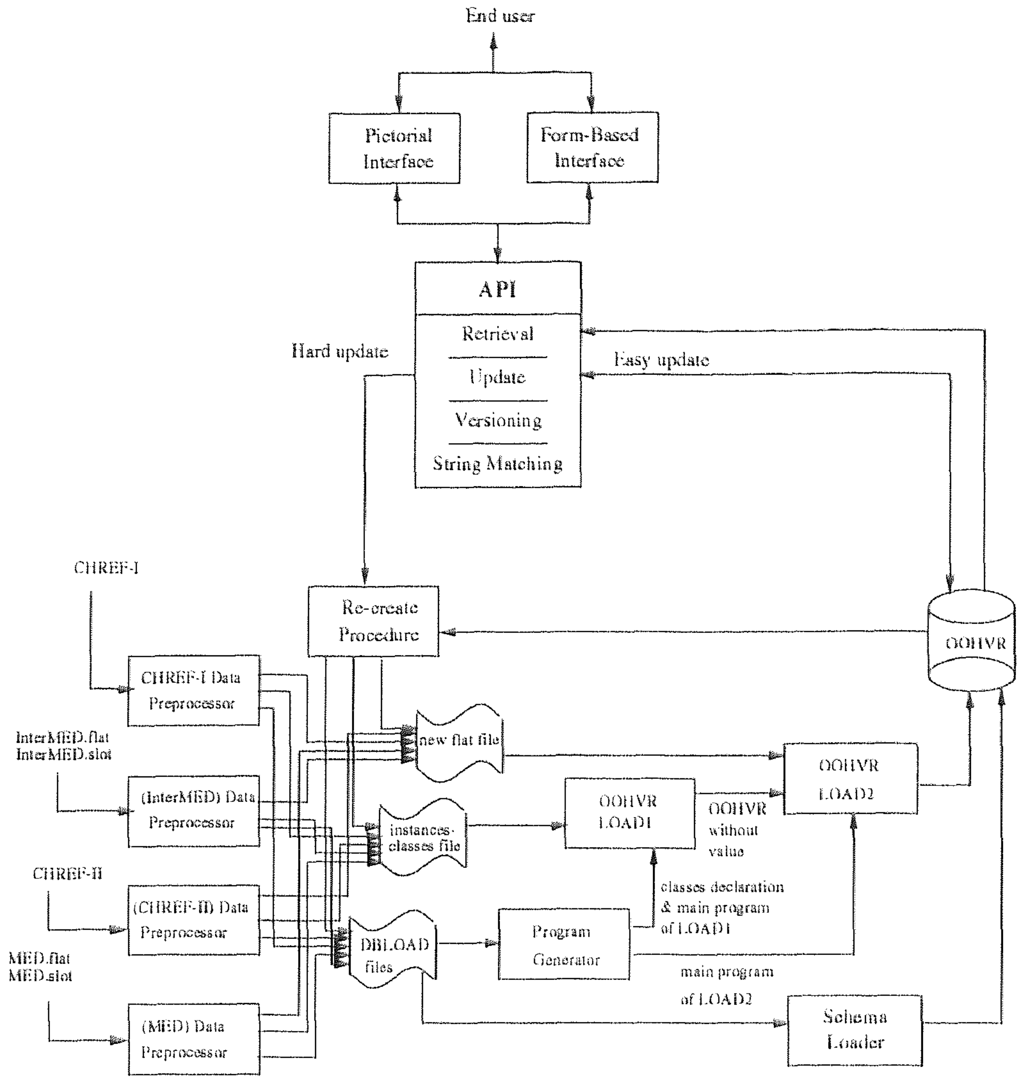

Figure 3.1 Schematic Figure of the whole system

characters and as such is not easily retyped. Moreover those class names contain complex medical terms which again are not easily retyped.

Some of the concept names of the InterMED contain special characters such as "/", "(", ",", etc. which are not permitted in C++ class names. Dealing consistently with those is much easier for a generator than for a human programmer. This adds another argument for the need for the *OOHVR generator*. In the next subsection, we will describe the format of the input data. Then we will advance to the OOHVR generator functionality.

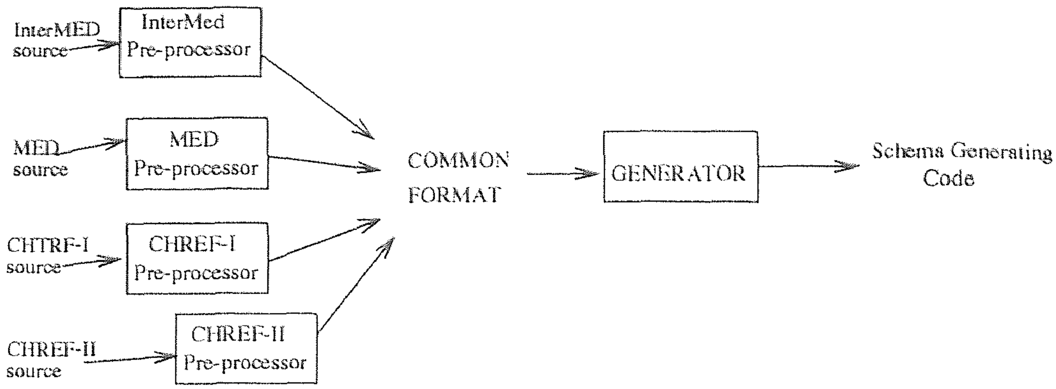

Figure 3.2 Schematic Figure of the Pre-processor

3.1.2 Preprocessor Description

The preprocessor is the part of the *OOHVR generator* which Pre-process the data in different formates and converts it into a common format from where the *OOHVR generator* can generate code for schema loading. Preprocessor takes schema dependent files as input and generates three files which will be used for generating schema code by the OOHVR generator. Figure 3.2 shows the process. Last section summarizes the layout of those files. Every preprocessor also generates three files which are used for creating instances of objects.

3.2 Processing Details

3.2.1 Implementation of Intersection Classes in ONTOS

If one draws the OOHVR as a graph then this graph has many nodes with several parents. An example of a class with two parents is the Chemical_Area. The preprocessor is complicated by the occurrence of a class with several parents. The actual difficulty is not caused by the two parents, but by any common ancestor of those two parents. However, this always happens in the OOHVR schema due to the existence of the class *Entity_Area* as its unique root. Since *Entity_Area* is a

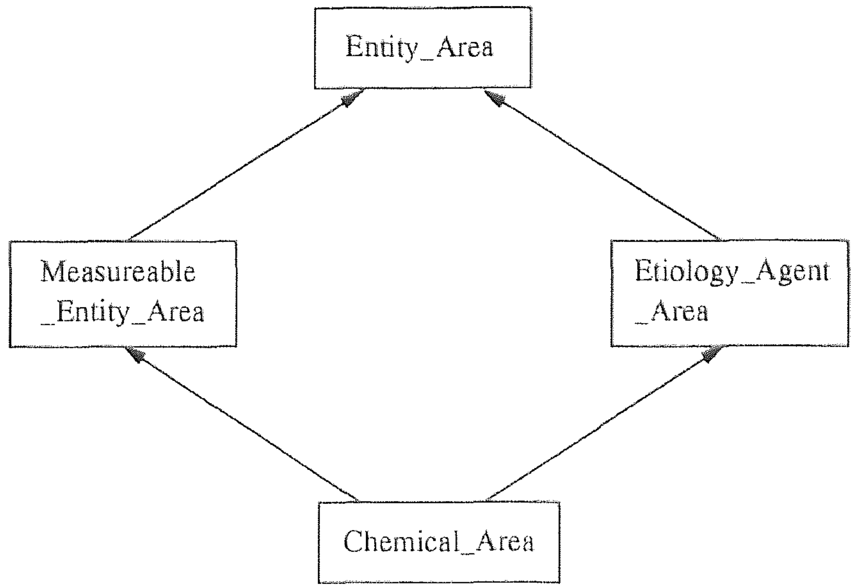

Figure 3.3 An example of multiple-inheritance in the InterMED

persistent class, all its descendants are also persistent classes which we want because ONTOS uses inheritance to make classes persistent. In Figure 3.3, *Chemical_Area* has two parents which have a common parent *Entity_Area*. C++ permits two solutions for such cases. In one solution, the structures corresponding to the common ancestor are duplicated. This leads to considerable waste of memory. For InterMed, we estimated that the waste of memory is as high as 60% for the *Acetaminophen_Codeine_Tablet_Preparation_Area*. The other solution makes use of the C++ virtual superclass construct. Unfortunately, by the syntax of C++, we cannot instantiate any virtual class, which means *Entity_Area* cannot have any instances and this is not true in the OOHVR.

To deal with this problem, we have constructed a "diamond cutting algorithm" which eliminates paths that have common ancestors. The details of the algorithm are given in the next section.

Another problem that we had to solve is as follows. Every relationship is introduced in a certain class and points to a particular class which is called the *target*

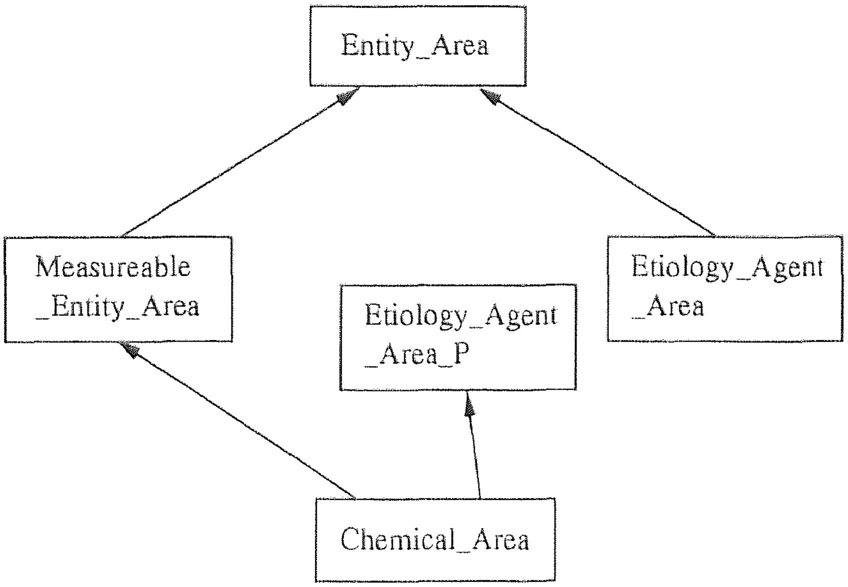

Figure 3.4 The result of applying the diamond cutting algorithm to Figure 3.3

area. If we have a relationship, *R*, which uses *Etiology_Agent_Area* as the target class then the extension of *Chemical_Area* should contain the candidate targets. However, *Chemical_Area* is not pointing to *Etiology_Agent_Area* from the DBMS's point of view. Any references to the instances of *Chemical_Area* will be considered an error. Our solution for this problem is to change all the relationship target areas to the root class, *Entity_Area*. Luckily we can store this information in the *shadow meta-schema*. Any setup of relationships will have no problems from the database management system's point of view. The correctness of the target areas for relationships is ascertained by checking the *shadow meta-schema* information.

3.2.2 Diamond Cutting Algorithm

As discussed earlier the algorithm basically eliminates paths that have common ancestors. The information loss which may happen is avoided by creating a copy of the class that has become unreachable by this operation. We call this copy "primed class" or "shadow class" of a node. For example, in Figure 3.3, Etiology_Agent_Area is initially a parent of Chemical_Area. After applying the algorithm, Etiology_Agent_Area_P becomes the shadow class of Etiology_Agent_Area and a "primed parent" of Chemical_Area.

The shadow class has no connection to the persistent superclass, and therefore the original problem is eliminated. It also has all the properties of the node it is copied from, so that the node with multiple parents is till inheriting the right set of properties as shown in Figure 3.4. As the shadow class is never instantiated, it does not need to be persistent.

Computationally, whenever the edge between an area and one of it's parent is cut, we transformed that parent from the parent-set of the class to its primed-set. We also add ancestors (persistent as well as non-persistent) of that primed parent to the primed-set of the intersection class excluding the root of the diamond.

For instance, in Figure 3.5, when we cut the edge between G and F, to prevent G from losing properties that were introduced in F we make a copy of the class F called F' and make it a parent of G. By cutting the edge we lost the properties of E and A. So, we make a copy of E too and make it a parent of G. Since A is root of the diamond, it's properties will be inherited via the other path through D and so we don't need to duplicate it.

The same solution is used whereever intersection classes occurs in any of the schemas. As mentioned in the last chapter we come across with intersection classes in InterMED, MED and CHREF-I. Hence the Diamond cutting algorithm was used in their pre-processor.

3.2.3 The InterMED and The MED

Due to similarities in the source of MED and InterMED we have decided to put the pre-processor information for them together. The InterMED and MED have the same disk-resident format consisting of two files: *slot file* and *flat file*. The MED

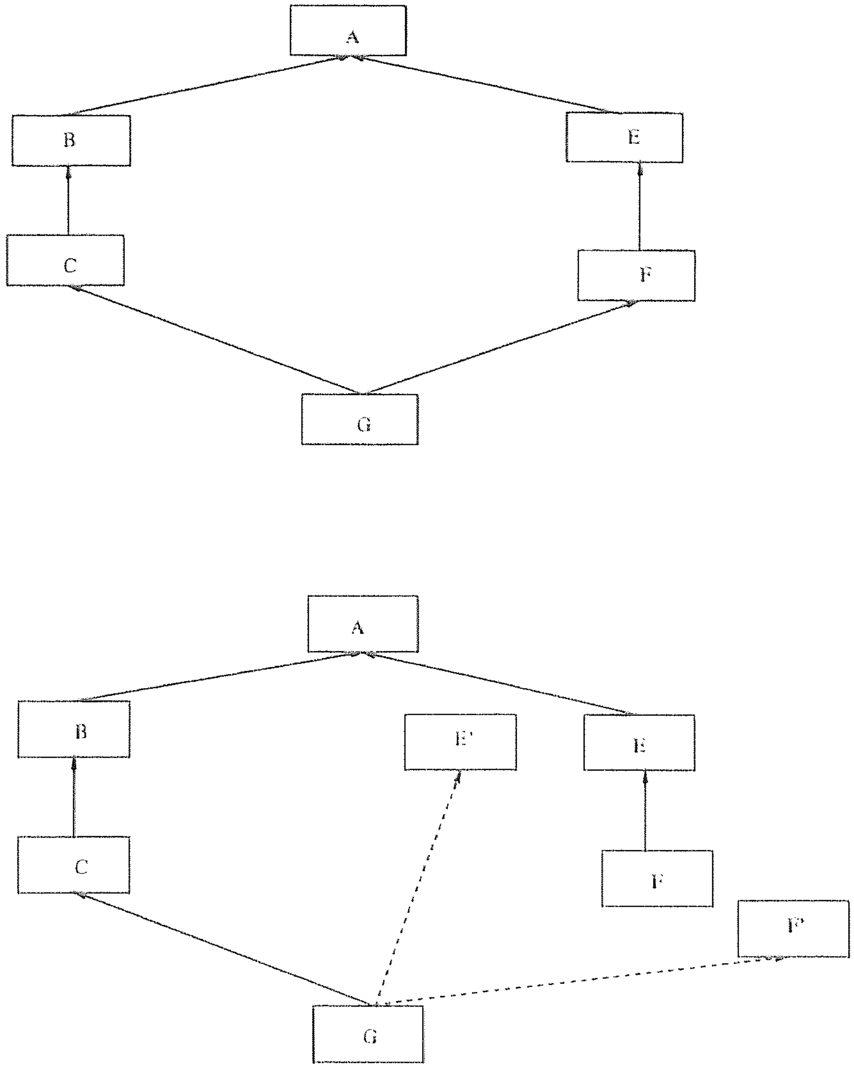

Figure 3.5 Advanced diamond cutting

is much larger in both slot file and flat file because the MED has over 16 times as many concepts as the InterMED. In this section we simply use the smaller one, the InterMED, as the example to discuss the process of generating the OOHVR. The first file, the *slot file*, describes all the attributes and relationships types of the InterMED. Every attribute (or relationship type) is described by one line in the slot file. As of this writing, there are 52 lines in the slot file. Figure 3.13 (a) shows the first couple of lines of the slot file. The fields in the *slot file* are separated by commas. The first field is the slot number. The second field is the slot name and the third field is the concept number which introduces this property. Attributes have a string in the last field while relationship types leave that field empty. The remaining four fields are irrelevant to this discussion.

```
0,"MED-CODE",1,-1,0,,"IDENTIFIER"
1,"UMLS-CODE",1,-1,0,,"IDENTIFIER"
2,"NAME",1,-1,0,,"SYNONYM"
3,"DESCENDANT-OF",1,0,0,-2,
4,"SUBCLASS-OF",1,0,0,-1,
5,"SYNONYMS",1,-1,0,,"SYNONYM"
6,"PRINT-NAME",1,-1,0,,"SYNONYM"
7,"DOCUMENTATION",1,-1,0,,"LONG_STRING"
8,"SNOMED-CODE",1,-1,0,,"IDENTIFIER"
9,"HAS-RESULT",3,1,1,10,
10,"RESULT-OF",28,0,1,9,
11,"HAS-SPECIMEN",4,1,1,12,
12,"SPECIMEN-OF",29,0,1,11,
13,"SUBSTANCE-MEASURED",5,1,1,14,
14,"MEASURED-BY",30,0,1,13,
15,"HAS-PRECISION",5,1,1,16,
```

(a) InterMED.slot file

```
1,1,"T071"
1,2,"ENTITY"
1,4,
1,5,"MEDICAL ENTITY"
1,6,"Entity"
1,7,"The class of all concepts
the collaborative vocabulary knowledge"
1,8,""
1,23,"1"
1,49,""
2,1,""
2,2,"PROCEDURE"
2,3,1
2,4,1
2,5,""
2,6,"Procedure"
```

(b) InterMED.flat file

Figure 3.6 The InterMED source files

The second file, the *flat file*, describes all the details of the data in the InterMED and currently contains over 43,000 lines. Figure 3.13 (b) shows the first couple of lines of the flat file. Essentially, an entry in the flat file consists of three elements. The first element is a concept number, a number representing one of the concepts in the semantic network. The second number is a slot number which stands for one of the relationship types or attributes and is therefore an index into the slot file. The third element may be another number (for a different concept) denoting the referent of a relationship. For an attribute, the third element is a primitive value, represented as a string type. For instance, the line 2,2,"PROCEDURE" means that the concept 2 is named "PROCEDURE" and the line 2,4,1 means that the concept 2, PROCEDURE, has the "SUBCLASS-OF" relationship (4) to concept 1, ENTITY. The MED has 160 lines in its slot file and around 950,000 lines in its flat file, and is constantly growing. More details are irrelevant for this paper and will be omitted.

The *Preprocessor* uses these two files to generate three types of intermediate files: *DBLOAD files*, *instances_classes file*, and *new flat files*. There are four *DBLOAD files*, one *instances_classes file*, and two *new flat files*. The *DBLOAD files* contain all the information necessary for generating class declarations for the area classes. They are sent to the *Program Generator* which generates the necessary DDL statements for generating OOHVR, as well as *Concept Creator*, and *Property Loader*. The *instances_classes file* contains the information for instantiating all concepts and is used by the *Concept Creator*. The *new flat files* contain the information for loading the property values and are used by *Property Loader* after *Concept Creator* instantiates all concepts.

The spirit of the Object Oriented paradigm has been adopted even while pre-processing the data files. Two major entities in our database are terms and areas so they have been defined as classes in C++ program. The area class is made a subclass of the term class because it is a superset of term class. A set class was used to maintain the set of parents and children of a term and area for that mySet.h. The library used was previously developed by R. Singh for developing the InterMED pre-processor. Using a set as a container of parents, slots, ancestors and primed-sets etc. provides an ability to apply powerful mathematical set operations like Union, Intersection and Difference etc.

We have differentiated between Area and Intersection Area in the last chapter so the first task of a pre-processor is to identify the list of Areas and Intersection Areas. The list of Areas can easily be generated from MED.slot or InterMED.slot since it contains the information about the term where each attribute or relationship was first time introduced. The third column of slots contains the term number where the property was first time introduced. The function Create_Areas_From_Slotsfiles in the program goes through each and every slot in the slot file and generates the list of Areas as an array of Area Objects and returns the count of total areas found.

Finding Intersection Areas is a difficult task, since the source files doesn't contain the information directly. The fact that there can be multiple intersection areas having exactly same set of slots lead us to maintain another file called intersection file. The file contains the root of the intersection areas and their parents.

```
   2,  BODY SUBSTANCE, 2672, 50
  43, CHEMICAL, 50, 135
1080,  WHITE PIEDRA, 1067, 2691
1179,  CHARACTER STRING RESULT, 1178,  32431
1712,  ALLEN SERUM AMYLASE MEASUREMENT,  144,  2248
2315,  ELECTROCARDIOGRAM, 2314, 24466
2548,  HEART DISEASE,  10016,  1178
2672,  PHYSICAL ANATOMIC ENTITY, 14, 32291
2691,  MICROORGANISM, 315,  135, 50
10014,  PULMONARY COLLAPSE, 21878 ,  10016
        10046,  PYOPNEUMOTHORAX, 1067, 10014
10055,  CALCIFICATION OF PLEURA, 10016, 35232
```

Figure 3.7 intersection_info a snapshot

The function Add_InterAreas_FromInterfile reads the intersection file and adds the intersection areas in the array of Areas. The next step is to maintain the hierarchy of Areas from the Area Array. The function Create_Hierarchy takes the array of areas and generates the set of parent and children areas for each area i.e. makes the whole hierarchy. If an Area X contains all the properties of area Y plus the properties introduced by itself, then Area X is a child of Area Y. This logic is used to generate the set of parent and child areas of an area while for the intersection Areas the information form the intersection file is used for making the hierarchy.

After making the hierarchy of Areas in memory as an array of area objects, the output files can be generated. Each area object contains the set of parents, children, and properties etc. Create_DBLOADx (where x can be 0,1,2 and 3) functions generates corresponding DBLOAD files. A function INST is designed for generating instances of the Areas.

3.2.4 Intermediate File

Many intermediate files are generated for MED, InterMED preprocessor use. Following is the information and format of those files.

```
The names of files are

         1) NewInterclasses.attrib
         2) inter_info
         3) MED.names
         4) Term.names
         5) MED.slots

Description of files:

    1) NewInterclasses.attrib
              This file contains the first two fields of the actual flat
              file only for Intersection Areas.
       Format :

                 medcode  slot_id

              Example :

 1  1
 1  2
1  4
1  5
1  6
1  7
1  8
1  50
1  51
```

2 1

How to create the file:

The file can be created by using the program gen_interinfo.cpp
Which takes the area hierarchy and NEWMED from the MED.latest
directory as input File and generates this file.

2) inter_info :

The file contains the area information.

Format :

```
med_code, Name , parent1, parent2, parent3, ..
```

Example:

```
 2, BODY SUBSTANCE ,50,2672
14, ANATOMIC ENTITY ,1
43, CHEMICAL ,50,135
49, SPECIMEN ,1
50, MEASUREABLE ENTITY ,1
75, MENTAL OR BEHAVIORAL DYSFUNCTION ,76,21762
76, DISEASE OR SYNDROME ,1
83, LABORATORY OR TEST RESULT ,1
93, LABORATORY DIAGNOSTIC PROCEDURE ,94
94, DIAGNOSTIC PROCEDURE ,1
135, ETIOLOGIC AGENT ,1
144, CPMC LABORATORY DIAGNOSTIC PROCEDURES ,93
315, CULTURE RESULT ,35878
1035, CULTURE PREFIX RESULT ,315
```

TO CREATE THE FILE :

The file can be created by using the program gen_interinfo.cpp
Which takes the area hirarchy and NEWMED from the MED.latest
directory as input File and generates this file.

3) MED.names & 4) Term.names

Files contains names of all terms and Areas. Only difference is in MED.names all "(",")","-" etc. are converted to "_".

FORMAT:

med_code   names

Example:

```
1 MEDICAL ENTITY
2 BODY SUBSTANCE
3 BODY SPACE OR JUNCTION
4 EMBRYONIC STRUCTURE
5 CONGENITAL ABNORMALITY
6 ACQUIRED ABNORMALITY
7 ANATOMIC SYSTEM
8 BODY PART, ORGAN, OR ORGAN COMPONENT
9 TISSUE
10 CELL
```

TO CREATE THE FILES :

gen_names.cpp program can be used which takes NEWMED as input and generates both files.

5) MED.slots

It is the slot file from which all "," and " (QUOTS) are deleted.

FORMAT:

slot_code   slot_name   term#   unknown Attrib-Relation

Example:

```
0 MED-CODE 1 -1 0  IDENTIFIER
1 UMLS-CODE 1 -1 0  IDENTIFIER
2 NAME 1 -1 0  SYNONYM
3 DESCENDANT-OF 1 0 1 -2
4 SUBCLASS-OF 1 0 0 -1
5 SYNONYMS 1 -1 0  SYNONYM
6 PRINT-NAME 1 -1 0  SYNONYM
7 HAS-PARTS 1 1 1 8
8 PART-OF 1 0 1 7
9 CPMC-LAB-PROC-CODE 144 -1 0  IDENTIFIER
10 SERVICE-CODE 144 -1 0  IDENTIFIER
11 CPMC-UNIT-NAMES 144 -1 0  NAME
12 CPMC-LAB-TEST-NAMES 2248 -1 0  NAME
13 SPECIMEN-OF 49 0 1 14
14 SPECIMEN 93 1 1 13
```

CODE	CLASSIFICATION
0100	ANESTHETICS/ADJUNCTS
0117	ANESTHETICS, LOCAL (INJECTABLE)
0118	ANESTHETICS, GENERAL
0119	ANESTHESIA, ADJUNCTS TO
0120	MEDICINAL GASES
0121	ANESTHETICS, TOPICAL
0122	ANESTHETICS, OPHTHALMIC
0123	ANESTHETICS, RECTAL
0200	ANTIDOTES
0281	ANTIDOTES, SPECIFIC
0283	ANTIDOTES, GENERAL
0285	ANTITOXINS/ANTIVENINS
0286	ANAPHYLAXIS TREATMENT KIT
0300	ANTIMICROBIALS
0346	PENICILLINS

Figure 3.8 A drugclass_table Snapshot

```
TO CREATE FILE:

 gen_medslots.cpp program can be used to get this file.
 The program takes NEWSLOTS as input.
```

3.2.5 CHREF-I

To maintain similarities in preprocessor code we designed same type of function for CHREF-I as well. Writing a pre-processor for CHREF-I is relatively a simple task due to lower complexity of it's schema. The drugclass_table contains the information about each drug. The program reads the drugs information from this file and maintains an array of Areas. An Area with UNKNOWN_NDC_CLASSIFICATION is created for the NDC-codes which do not fall into any classification.

```
1032 1265
1032 1479
1032 1568
1032 1724
1032 1940 1941
1032 1941
1032 1947
1033 1479
1034 1040
1034 1041
1034 1265
1034 1479
1035 1041
1035 1041 1371
```

Figure 3.9 intersection_classes snapshot

We maintain an "intersection_classes" file for the intersection classes. The next step of the program reads the information about the intersection classes from the file and adds them in the array of Area Objects. Figure 3.9 shows one intersection class per line. The file doesn't give names to the classes so we build the names of intersection classes by concatenating the names of all parents classes of an intersection class. The MakeHierarchy function makes the hierarchy by adding the values in the parents and children sets. The hierarchy is built on the biases of the code. E.g. if the last two digits of the code are 00 that means the class represents the major drug class while if the last two digits are not 00, the class belongs to the third level of the schema hierarchy. In Figure 3.8 code 0100, 0200 and 0300 represent major classes while all other not having the last two digits 00 belong to the minor classification. The fourth level of the schema hierarchy is the intersection classes. GenerateDbloads generates all the output files while INST generates the files for creating instances.

```
1 A
2 A1
3 A1A
4 A1B
5 A1C
6 A1D
7 A2
8 A2A
9 A4
```

Figure 3.10 hic_class snapshot

3.2.6 CHREF-II

CHREF-II preprocessor is simpler than other preprocessor. Since are no intersection classes in CHREF-II schema we no don't have to apply the diamond cutting algorithm. It just takes the HIC hierarchy i.e. represented by the code of the classes. If a class has only one character code than it belongs to major classes i.e the second level classes. The hic_class file contains the list of all codes and their class numbers. The file tblHic.txt contains the names of those drug classes. The program reads the hierarchy from the hic_class file and the name of the drug classes from the tblHic.txt file and generates the DBLOAD_X files. For instances the INST function takes attributes files input and generates inst.out, o12 and o3 files.

3.3 Output Files Format

3.3.1 DBLOAD_X

There are four DBLOAD files generated by the preprocessor. All files have the same structure. All concept has their attributes and relationships, in the DBLOAD files in the following format.

```
MEDICAL_ENTITY_AREA
0
7
MED_CODE
UMLS_CODE
NAME
SYNONYMS
PRINT_NAME
MAIN_MESH
SUPPLEMENTARY_MESH
5
DESCENDANT_OF MEDICAL_ENTITY_AREA
SUBCLASS_OF MEDICAL_ENTITY_AREA
HAS_PARTS MEDICAL_ENTITY_AREA
PART_OF MEDICAL_ENTITY_AREA
SUPERCLASS_OF
```

Figure 3.11 DBLOAD_X snapshot

```
CONCEPT NAME
Number of Subclasses
Names of subclasses each on different line.
Number of Attributes
Name of Attributes
Number of Relationships
Name of relationships
```

Above entry shows that ENTITY_AREA has 0 number of parents, 7 attributes which are listed on next 7 lines, has 5 relationships listed with target classes.

The DBLOAD_0 file contains the area list before applying the diamond cutting algorithm, while DBLOAD_1 contains the area list after the diamond cutting algorithm. DBLOAD_2 contains list of shadow areas. DBLOAD_3 contains all the areas in DBLOAD_0 but areas also contain the properties they inherit from their parents.

```
SINGLE_RESULT_LABORATORY_TEST_AREA      CHEMISTRY TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      INTRAVASCULAR CHEMISTRY TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      WHOLE BLOOD CHEMISTRY TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      PLASMA CHEMISTRY TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      SERUM CHEMISTRY TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      INTRAVASCULAR SODIUM ION TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      WHOLE BLOOD SODIUM ION TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      PLASMA SODIUM ION TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      SERUM SODIUM ION TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      IMMUNOLOGY TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      COAGULATION TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      CELL AND ARTIFACT TEST
SINGLE_RESULT_LABORATORY_TEST_AREA      MICROBIOLOGY TEST
```

Figure 3.12 inst.out snapshot

3.3.2 inst.out, o12 and o3

These files are used to generate instances of the areas. inst.out contains the name of the area an object belongs to and the value of its key field. Each line of inst.out represent an object. The format of the line is

```
<AREA NAME>            <KEY FIELD VALUE>
```

In Med and Intermed the keyfield is the name of the concept while in CHREF-I and CHREF-II it is the NDC-CODE of the drug.

o12 and o3 contain the values of other attributes of the objects. o12 contains the name of the attributes and Object identifier.

```
<NAME OF AREA>   <Object Identifier>
```

Each line in o12 has a corresponding line in o3 which contains the value of the attribute.

```
<Value of the attribute listed in o12 file>
```

UMLS_CODE ENTITY	T071
NAME ENTITY	ENTITY
SUBCLASS_OF ENTITY	
SYNONYMS ENTITY	MEDICAL ENTITY
PRINT_NAME ENTITY	Entity
DOCUMENTATION ENTITY	The class of all concepts
SNOMED_CODE ENTITY	1
CPMC_CODE ENTITY	0
CATEGORY_OF ENTITY	
CATEGORY_TYPE ENTITY	
SUPERCLASS_OF PROCEDURE	SPECIMEN COLLECTION PROCEDURE

Figure 3.13 The InterMED source files

CHAPTER 4

DESIGNING A VOCABULARY CREATOR

4.1 Introduction

Development and research in the medical industry produces new concepts and terms in the subject. An interactive tool is necessary to allows a medical expert to add new concepts and relationships to an existing vocabulary as well as to create a new vocabulary from scratch.

When a user starts creating a vocabulary he/she starts by defining a set of terms/concepts. Then he/she adds relationships and attributes to the concepts, which leads him to a semantic model or a conceptual representation of the subject knowledge. In the process a user has to face a number of challenges. He has to create and organize a large amount of concepts. He has to assure that each concept contains the attributes and relationships necessary to represent that concept in the subject field. To accomplish this the user must have a solid grasp of the overall structure of the vocabulary.

A Vocabulary usually contains at least hundreds of concepts and can grow up to tenths of thousands. Remembering the name of just a few dozen of these and the relationships between them may be troublesome. With the increase of vocabulary size, comprehending the structure becomes almost impossible. A graphical view of the vocabulary can provide an easy to visualize tool for the vocabulary creator. That is why we decided to provide a graphical user interface for our IVC (Interactive Vocabulary Creator).

As discussed earlier the vocabularies should provide an easy to access knowledge. The internet is a way which can provide easy and world wide access to knowledge. We decided to use WWW as a medium of distribution for our IVC. Section 4.2 provides background of WWW systems. Section 4.3 gives a brief idea about which are the essential features of IVC. Sections 4.4, 4.5 and 4.6 deals with design issues in details.

4.2 Background of WWW

The World Wide Web architecture was developed by Berners-Lee [13] and is based on a generic object-oriented protocol, the Hypertext Transfer Protocol (HTTP). This protocol manages requests in the form of a Uniform Resource Locator (URL) and delivery of information as Multipurpose Internet Mail Extension (MIME) objects. The most common objects delivered by the HTTP protocol are documents written in the Hypertext Markup Language (HTML), a subset of the more general Standard Generalized Markup Language (SGML) [1]. HTML adds structure to ASCII text documents, and WWW browsers (such as Mosaic or Netscape) use this structure to display the text in a graphical manner. Beyond designating the structure of the documents, HTML provides a syntax for embedding graphics, images, sounds, and video, as well as hyperlinks to other documents [13].

One of the main tenants of the HTTP protocol is that it is stateless: after the HTTP server returns the requested information, the session is terminated. No information about the state of the user is maintained. Because many interactive processes require maintenance of state information, developers have maintained state information in hidden fields of HTML forms or in the databases resident on the server [5].

To support the processing of user input, the Common Gateway Interface (CGI) standard was developed. This standard assures that WWW browsers, HTTP servers, and external processes communicate using a standard set of parameters. When a hyperlink or HTML form is used to initiate a CGI process, the HTTP server receives the request, starts the CGI process with the parameters submitted by the user, waits for the output of the CGI, and delivers the output to the browser. The CGI application can use the supplied parameters to perform almost any task: make a database query, annotate a document, or send an electronic mail message.

Another drawback of HTML is, one can not design user interfaces for which complex client side computation and display is required. Java provides the solution to this limitation. Java has the capability of virtually handling any kind of complex interface. We decided to use Java based client, for the Creator. To make the data persistent Java still needs some interface to talk to the server. We used CGIs as our interface. Other possible interfaces are discussed in chapter 5.

4.3 Essential Features of a Vocabulary Creator

A vocabulary creator should have the following features as a minimum.

- Add/Delete/Edit a Concept to the vocabulary.
- Add/Remove/Edit an attribute of a concept.
- Add/Remove/Edit values of the attribute.
- Add/Remove/Edit Relationships in a concept.
- Add/Remove/Edit Relationships values.

Other features includes a good search mechanism, which allows a user to retrieve information from the vocabulary.

We started to develop a creator initially with the essential requirements.

4.4 System Architecture

The architecture of the IVC is composed of five components

1) A Java Based WWW Browser

2) An HTTP Server

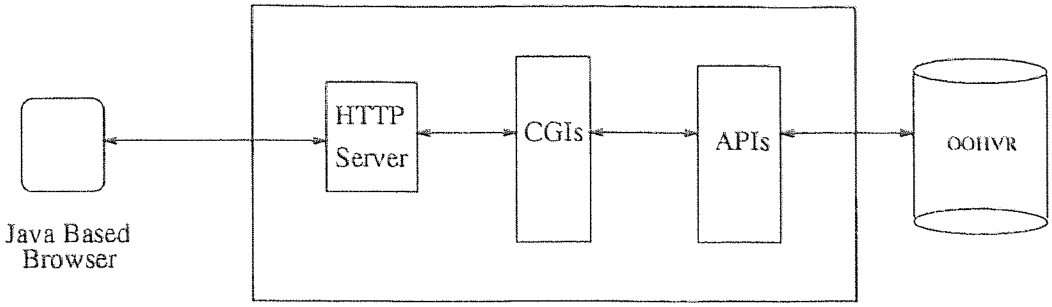

Figure 4.1 Architecture of IVC

3) CGI Mediator

4) APIs

5) OOHVR

The following steps are followed in response to a user's request. A user activates a request e.g. a new Concept addition, from the Java based browser. The browser activates the CGI mediator on the HTTP server by using the HTTP protocol. CGI Mediator which calls the corresponding APIs for the request and these APIs actually communicate with the ONTOS database to make changes accordingly. If the request is a query from a user than the resulting data is passed back to the CGI program, which delivers the data to the HTTP server, and it is sent back to the Client. Detailed information about CGI, APIs and OOHVR is given in the next section. A Java based browser design is discussed in section 4.6.

4.5 Back-End Design

4.5.1 OODB Schema for a General Vocabulary

By using the Creator one can build any kind of vocabulary. We designed a schema which can allow editing of a general semantic network. We decided to use a relatively simple schema, which is shown in Figure 4.2. Any concept would be an instance of

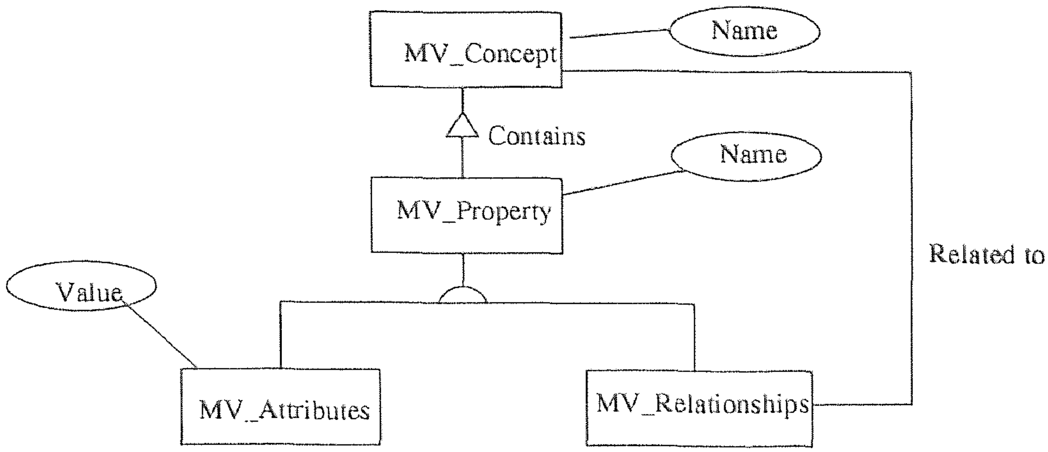

Figure 4.2 Schema for a general Vocabulary for IVC

MV_Concept which has Name as an attribute. It contains a set of MV_Property. The MV_Property is a base class for MV_Attribute and MV_Relationships, it contains the Name of the property. MV_Attribute has attribute value which actually saves the value of attribute. MV_Relationships has a reference to a target concept for the relationship. Since MV_Concept contains a set of Properties it can have more than one Attribute and Relationship.

4.5.2 API Design

Application programmers Interfaces (API) are functions developed as a library. They provide an easy access to the database for an Application programmer. We have designed a set of APIs for all possible editing or browsing requests. These APIs call the ONTOS database to provide a desired functionality. A list of the APIs is given in APPENDIX C.

4.5.3 Common Gateway Interfaces

CGIs are a main component of the Creator. They provide an interface between the APIs and HTTP server. As discussed before, the CGIs are the programs which provide dynamic data to the Web. Our CGIs has to perform the requested APIs. To accomplish that, we designed a protocol for client and server synchronization. First we'll provide the general details about the CGIs. Then we'll discuss the Protocol we have adopted for the client and server synchronization and error checking.

CGIs are executable programs which are executed by the HTTP server upon request of the client. They can take data in two different ways. They can give back results in only one way i.e writing to the standard output. Data input to CGIs can be given by Standard input (POST Method) or by environment variables (GET Mthod) . Environment variables are good for the cases when one has to transfer a smaller amount of data to the CGIs. Because of our data size we decided to use Standard Input (POST Method) for providing data to CGIs.

The client actually prepares for a connect string i.e. a (Universal Resource Locator) URL. A URL contains the type of protocol to be used, the server to be connected, name and path of CGI program and a list of parameters to be passed to the CGI program. We used the HTTP protocol as our connection protocol.

For example a server name is object.njit.edu:2000 (where 2000 is a port number of the HTTP server), and the CGI program is oohvr/oohvr.cgi and we have to pass variables NAME=Text and EMAIL=xyz@homer.njit.edu to CGI. Then the URL would be

http://object.njit.edu:2000/õohvr/oohvr.cgi?NAME=Text&
EMAIL=xyz@homer.njit.edu

More than one parameter can be passed by "&" separated strings. The Text here is the value of the parameter, which should be in an encoded form. The encoded form is to change all spaces to "+" and special characters in hexadecimal number representing ASCII code of the character. This whole creation of URL is done on the client side. The above URL would instruct the HTTP server to run the program oohvr.cgi and pass NAME=Text&EMAIL=xyz@homer.njit.edu to the program, which reads this string from the standard input. After performing the desired request, the is output on standard Output.

To call the APIs from a remote side, we take a variable name FUNCTION. The value of the variable would tell the CGI program to call a specific API. Let us suppose to run an API ListAllConcepts, the URL would be

http://object.njit.edu:2000/õohvr/oohvr.cgi?FUNCTION=ListAllConceptes.

The CGI program in the begining will see the name of the function from FUNCTION variable and would call MV_List_All_Concepts. The result of the API is sent to the Standard Output which would be redirected to client by the HTTP server. For the APIs which need some parameters to be passed to, they are passed by using the Name=Value. E.g. to call ListAllChilern the API need the name of the parent concept so the URL would be constructed as following

http://object.njit.edu:2000/õohvr/oohvr.cgi?FUNCTION=ListAllChildern &PARENT=ENTITY

If an API requires more than one parameters the parameter are sent by concatenating desired parameter names and their values at the end of URL proceeded by "&" sign.

The CGI returns OK in first line of output to represent that the request has been fulfilled . If the first line doesn't contain OK, then that means there is some error occurred. In this case the first line represents the Error message. In the case of a successful query the lines followed by OK contains the results of the query. If there is nothing after the OK line, that means the data is not available or the request was just an edit operation to be performed.

4.6 Front-End Design

4.6.1 The Notion of Neighborhoods

One of the problems that we had to face was how to display a vocabulary. Our initial choice was to use a graphical display of the vocabulary network which shows whole vocabulary on the screen. We have built an experimental layout algorithm and fed it the InterMED hierarchy, i.e., all the nodes and the IS-A connections between them, but no attributes and no relationships. The result was a picture that was too overwhelming to be of any use [11]. It can be described as having a center that is entirely black with no recognizable features whatsoever. In addition, the layout algorithm was intensive in computational time.

The general problems we face in viewing a graphical diagram of a vocabulary are large scale and high complexity, particularly in comparison with the limited size of display media (e.g., a computer screen) and limited human-comprehension capacity. The complexity issue was previously discussed in [12], where the ratio of edges to nodes was proposed as a quantitative measure of diagram complexity. Even if we can display a diagram of, say, 50 concepts and 200 connecting relationships (lines) on a single page of paper or on a monitor, such a diagram is overwhelming to most users. And 50 concepts usually represents a small fraction of a CMV.

To cope with these difficulties, we define the notion of various forms of *concept neighborhoods* (or *neighborhoods*, for short) in CMV diagrams.

Definition (Neighborhood): The neighborhood of a concept V in a CMV diagram contains V and V's children and parents (with respect to the IS-A hierarchy) as well as any concepts related to V via non-hierarchical relationships.

Definition (Two-level Neighborhood): The two-level neighborhood of a concept V in a CMV diagram contains V's neighborhood and V's siblings, grandparents, and grandchildren (with respect to the IS-A hierarchy).

Definition (Indirect Ancestors): The indirect ancestors of a concept V are the ancestors of V excluding V's parents.

Definition (Indirect Descendants): The indirect descendants of a concept V are the descendants of V excluding V's children.

Definition (Extended Neighborhood): The extended neighborhood of a concept V contains V's neighborhood and V's siblings, indirect ancestors, and indirect descendants.

Note that these definitions are valid both for the concept diagram and the area class diagram of a CMV when it is modeled as an OODB, as discussed in the previous section. We refer to the respective neighborhoods as concept neighborhoods and (area) class neighborhoods when it is necessary to draw a distinction.

A neighborhood diagram displays only a portion of limited size and complexity of the entire CMV diagram, and thus affords a user a much more comprehensible view. As we will explain later, navigating through a CMV network via successive "center" shifts from a concept to another concept in its neighborhood facilitates search traversals.

To use an analogy, navigation is like looking at the night sky with a telescope that magnifies a small portion. By moving the telescope slowly, the astronomer achieves a "sliding" clear view of a substantial portion of the sky. He uses his limited view to obtain an overall view and focus in on objects of interest.

The various kinds of neighborhoods give the user of the vocabulary flexibility in the choice of "focus." At each stage, a user can select a suitable view based on the size of the neighborhood and the desired information. For example, if a concept's neighborhood contains only six nodes, the user might choose the two-level neighborhood or even the extended neighborhood. On the other hand, for a large neighborhood, the ordinary neighborhood display might be more appropriate.

There are, however, problems in creating such an interface to the CMV:

1. The number of children of a concept might be so large as to not fit on the screen.

2. The neighborhood layout may be time consuming to generate on demand as it could differ considerably from the layout of the same concepts in the context of the whole vocabulary diagram.

It is clear from the above discussion that nothing is perfect in terms of a good view for the user. So we decided to give the user a choice of more than one type of views. A user in the start of the program will see only the root and the first level of the vocabulary. That means only,first level neighbor will be shown while the root is the focused element. After that a user can expand any child of Root to see the second level neighbor i.e. one can view the vocabulary as needed.

To decrease the complexity some other views are provided to a user in which 1) First Level Neighbor, second level neighbor, ancestorial view and descendent view. The ancestor view starts from the bottom and goes up to the root of the vocabulary. This would be a somewhat simple view because usually a concept doesn't have many multiple parents. Descendent view starts from the given concept and goes all the way to the bottom. We can say it is a top to bottom view. We are planning to have n-level neighborhood view in which a user can give n as a parameter and the program generates the view.

4.6.2 Programming Details

The skeleton of the program consists of three main parts.

API Calling Component.

Layout Manager.

User Interface.

We will discuss them separately.

4.6.2.1 API Calling Component

This component basically provides an interface to the CGIs discussed in the previous section. The component is designed as a separate Java class OOhvrCGI. It provides all the APIs to other components. Other components of the program would call those APIs just like they are calling APIs from a local system. It provides a transparent interface to Server APIs. Due to this feature it can be changed with any other interface, like RMIs or CORBA and we don't have to change other components of the program.

The OOhvrCGI contains all available APIs declarations with the same number and types of parameters as they are for APIs. Those API functions basically build the URL string discussed in the last section and calls CallServer a function of OOhvrCGI. That function calls the URL and gives the results back. If there is an error, a public boolean variable of the class would give the indication to the calling object.

4.6.2.2 Layout Manager

If we take concepts as nodes and relationships between them as edges then a Vocabulary is nothing but its a directed Graph. The layout manager keeps track of the graph and has the ability to apply an algorithm on the graph for the layout of the concepts on the screen.

4.6.2.3 User Interface

The user Interface is the central part of IVC. It provides a graphical view, dialog boxes and frames to user. The user interface calls other components of the program when needed. It gets data from the server by using the OOhvrCGI class and pass it to Layout manager which apples the layout algorithm on the graph. Then user interface shows the vocabulary. The user can select a particular concept and can apply the different functions. Like one can select a concept and press the Properties button to see and edit all the properties of the concept. The property editor is a dialog box which provides the Attributes and Relationships with their values in separate list boxes. All essential features discussed in 4.3 Section are included in the system.

CHAPTER 5

FUTURE WORK

The current architecture of IVC uses CGIs for user for client server communication. In the following we'll discuss some other ways of client server communication and try to find out the differences in them in terms of query executation time.

5.1 Performance Criteria

The following times are considerable for giving a quick response to a user on his requests

5.1.1 Connection Establishment Time

Our network is based on the TCP/IP protocol. The establishment time is to translate the server name to an IP address from a DNS server. Then finding out the route to the destination for the virtual circuit, and actually setting up the virtual circuit.

5.1.2 Request Placement Time

Request placing time depends upon the amount of data transfer for the request since some APIs need more data as arguments.

5.1.3 Database Access Time

This time is actually the setting up time for a query. A process is forked to access the database by HTTP server and establishes connection to databases.

5.1.4 Data Retrieval Time

This time totally depends upon the way the APIs retrieve data from a database and can be decreased by any optimization to the APIs if possible.

5.1.5 Data Transfer Time

This time is dependent upon the following factors:

Current Traffic size on the network.

Amount of data to transfer.

Distance data has to travel.

5.1.6 Presentation Time

This is the layout and client side calculation part on data which may be required. In the case of the Creator this time is very important. The Creator has to apply a layout algorithm on the schema graph.

5.2 Different Options Available for Client Server Communication

5.2.1 Common Gateway Interfaces

A way to transfer dynamically generated data from the HTTP server side upon client request. Upon request, the data is retrieved from the API's and is sent to the client where the client side program can present it. For each and every CGI request, the client has to establish the connection, i.e. our time consideration Connection Establishment time is always delaying the response time. For the database accessing, every time a process has to be forked by the HTTP server. It needs setting up time by the operating system and for each request database opening time which is also considerable time. All these are acutely performance drawbacks of CGIs. But on the other hand CGIs are easy to develop and setup and no special client is always necessary to access data. An HTTP browser can be used directly to browse the data.

5.2.2 TCP/IP Based API Server

This approach can provide better performance than CGIs. We can design a TCP/IP based server which can continuously listen on an assigned port of the server for an API request. The server, at start time, can open the database. This eliminates the need of forking a process on each request. The client can set up a TCP/IP virtual circuit in the start of the program. Then only an API call is needed. I.e. setting up time will be once only while database access time would also be improved. Minor changes to the APIs may be needed for a better database access time.

5.2.3 Java RMIs

Java Remote method invocation can be a good alternative to TCP/IP based server, because developing a TCP/IP based server is not an easy task. RMI server can approximately give the same performance as a TCP/IP server can give. RMIs are the way of calling methods of Java object on remote machines which actually runs on the server side and returns data on client side transparently. RMI gives a better performance as compared to CGIs since (Start of client program) setup time is required once. The main problem is that our system has to call the APIs written in C++. I.e. we need an Interface between Java and C/C++. JNI (Java Native Interface) can be used to call APIs written in C++.

5.2.4 Using CORBA

Common Object Request Broker Architecture is a standard for remote object binding. CORBA is similar to RMIs except that it is a language independent Architecture. In RMIs we have to use JNIs for Java to C++ interface. CORBA can be used for direct binding of server's C++ objects to client's Java objects.

5.3 Conclusion

- We need a change in the architecture for better performance.
- TCP/IP based API server is the best option available but at the same time it is hard to develop. Since we only needed to provide an interface between Java and C++ at this stage, RMIs of Java is second best option. CORBA can widen our future work, which can allow us to create client other than Java language.

APPENDIX A

CGIs CODE

The code for the CGIs is presented here.

```
#include"api/include/MV_ut.h"
#include"api/include/MV_api.h"
#include<iostream.h>


#include <stdlib.h>
#include <stream.h>
#include <string.h>
#include "cgic.h"

#define PARENT   1
#define CHILDREN 2

/*   LIST OF FUNCTIONS AVAILABLE FROM CGI  */

#define OKFLAG "OK\n"
#define NO_FUNCTION "NO\n"
#define ERROR_FLAG  "ERROR\n"
#define CONTINUE "CONTINUE\n"

#define fnCHILDREN "CHILDREN"
                    // PARENT=" "  would be parameter
#define fnPARENT   "PARENTS"
                   // CHILD=" "   would be parameter
#define fnAttribValue            "AttribValue"
#define fnAllAttribValue         "AllAttribValue"
#define fnAllRelationValue         "AllRelationValue"
#define fnRelationShipValue      "RelationValue"
#define fnChangeAttribVal        "ChangeAttribValue"
#define fnChangeRelationVal      "ChangeRelationValue"
#define fnAddChild               "AddChild"
#define fnAddProperty            "AddProperty"
#define fnAllConcepts            "AllConcepts"
#define fnAddRelationshipValue "AddRelationshipValue"
#define fnCreateNewRelation      "CreateNewRelation"
#define fnDeleteAttribute        "DeleteAttribute"
```

```
#define fnDeleteRelation        "DeleteRelation"
#define fnRemoveAttributeValue "RemoveAttributeValue"
#define fnRemoveRelationValue  "RemoveRelationValue"
#define fnDeleteConcept         "DeleteConcept"

void Child_Parent(char *,int);
void AttribValue(char *CName,char *PName=0,char FLAG=1);
void List_All_Concepts();

cgiMain(void)
{
char Function[30];

        char hst[100];
        cgiFormStringNoNewlines("FUNCTION", Function, 30);
        cgiHeaderContentType("text/html");
        if(!strcmp(Function,fnCHILDREN)){
            char parent[100];
            cgiFormStringNoNewlines("PARENT", parent, 100);
            Child_Parent(parent,CHILDREN);
        }else if(!strcmp(Function,fnAllConcepts)) {
             List_All_Concepts();
        }else if(!strcmp(Function,fnPARENT)){
             char child[100];
             cgiFormStringNoNewlines("CHILD",child,100);
             Child_Parent(child,PARENT);
        }else if(!strcmp(Function,fnAttribValue)){
            char concept[100];
            char attr[100];
            cgiFormStringNoNewlines("CONCEPT",concept,100);
            cgiFormStringNoNewlines("ATTRIBUTE",attr,100);
            AttribValue(concept,attr,0);
        }else if(!strcmp(Function,fnAddProperty)){
            char concept[100];
            char prop[100];
            cgiFormStringNoNewlines("CONCEPT",concept,100);
            cgiFormStringNoNewlines("PROPERTY",prop,100);
            MV_Create_Attribute(concept,prop);
            fprintf(cgiOut,OKFLAG);
        }
        else if(!strcmp(Function,fnAllAttribValue)){
                char concept[100];
                cgiFormStringNoNewlines("CONCEPT",concept,100);
                AttribValue(concept);
```

```
}
else if(!strcmp(Function,fnDeleteConcept)){
        char concept[100];
        cgiFormStringNoNewlines("CONCEPT",concept,100);
        MV_Delete_Concept(concept);
        fprintf(cgiOut,OKFLAG);
}
else if(!strcmp(Function,fnAllRelationValue)){
        char concept[100];
        cgiFormStringNoNewlines("CONCEPT",concept,100);
        AttribValue(concept,NULL,3);
}
else if(!strcmp(Function,fnAddRelationshipValue)){
        char concept[100],target[100],Relation[100],*RRelation;
        cgiFormStringNoNewlines("CONCEPT",concept,100);
        cgiFormStringNoNewlines("Relation",Relation,100);
        cgiFormStringNoNewlines("Target",target,100);

     if(MV_Show_Reverse_Relationship(Relation,RRelation))
     {
        MV_Add_Relationship_Value(concept,Relation,target);
        MV_Add_Relationship_Value(target,RRelation,concept);
        fprintf(cgiOut,OKFLAG);
        delete RRelation;
     } else{
        fprintf(cgiOut,"NO REV. RELATION");
     }
}
else if(!strcmp(Function,fnCreateNewRelation)){
        char concept[100],target[100],Relation[100],revRelation[100];
        cgiFormStringNoNewlines("CONCEPT",concept,100);
        cgiFormStringNoNewlines("RELATION",Relation,100);
        cgiFormStringNoNewlines("RevRELATION",revRelation,100);
        cgiFormStringNoNewlines("Target",target,100);
        MV_Create_Relationship(concept,Relation,target,revRelation);
        fprintf(cgiOut,OKFLAG);
}
else if(!strcmp(Function,fnRelationShipValue)){
      char concept[100];
      char attr[100];
      cgiFormStringNoNewlines("CONCEPT",concept,100);
      cgiFormStringNoNewlines("ATTRIBUTE",attr,100);
      AttribValue(concept,attr,2);
}else if(!strcmp(Function,fnChangeAttribVal)){
```

```
        char CName[100];
        char PName[100],oValue[100],Value[100];
        cgiFormStringNoNewlines("CONCEPT",CName,100);
        cgiFormStringNoNewlines("ATTRIBUTE",PName,100);
        cgiFormStringNoNewlines("OLDVAL",oValue,100);
        cgiFormStringNoNewlines("VALUE",Value,100);
        MV_Change_Attribute_Value(CName,PName,oValue,Value);
        fprintf(cgiOut,OKFLAG);
}else if(!strcmp(Function,fnChangeRelationVal)){
        char CName[100];
        char PName[100],oValue[100],Value[100];
        cgiFormStringNoNewlines("CONCEPT",CName,100);
        cgiFormStringNoNewlines("ATTRIBUTE",PName,100);
        cgiFormStringNoNewlines("OLDVAL",oValue,100);
        cgiFormStringNoNewlines("VALUE",Value,100);
        MV_Change_Relationship_Value(CName,PName,oValue,Value);
        fprintf(cgiOut,OKFLAG);
}else if(!strcmp(Function,fnAddChild)) {
          char Parent[100],Child[100];
          cgiFormStringNoNewlines("PARENT",Parent,100);
          cgiFormStringNoNewlines("CHILD",Child,100);
          MV_Create_Concept(Child,Parent);
          fprintf(cgiOut,OKFLAG);
}else if(!strcmp(Function,fnDeleteAttribute)) {
          char CName[100];
          char PName[100];
          cgiFormStringNoNewlines("CONCEPT",CName,100);
          cgiFormStringNoNewlines("ATTRIBUTE",PName,100);
   MV_Delete_Attribute(CName,PName);
          fprintf(cgiOut,OKFLAG);
}else if(!strcmp(Function,fnDeleteRelation))  {
          char concept[100],target[100],Relation[100],revRelation[100];
          cgiFormStringNoNewlines("CONCEPT",concept,100);
          cgiFormStringNoNewlines("RELATION",Relation,100);
          cgiFormStringNoNewlines("RevRELATION",revRelation,100);
          cgiFormStringNoNewlines("Target",target,100);
          MV_Delete_Relationship(concept,Relation,target,revRelation);
          fprintf(cgiOut,OKFLAG);
}else if(!strcmp(Function,fnRemoveAttributeValue)) {
          char CName[100];
          char PName[100],Value[100];
          cgiFormStringNoNewlines("CONCEPT",CName,100);
          cgiFormStringNoNewlines("ATTRIBUTE",PName,100);
          cgiFormStringNoNewlines("VALUE",Value,100);
```

```
                    MV_Remove_Attribute_Value(CName,PName,Value);
                    fprintf(cgiOut,OKFLAG);
        }else if(!strcmp(Function,fnRemoveRelationValue))  {
                    char CName[100];
                    char PName[100],Value[100];
                    cgiFormStringNoNewlines("CONCEPT",CName,100);
                    cgiFormStringNoNewlines("RELATION",PName,100);
                    cgiFormStringNoNewlines("VALUE",Value,100);
                    MV_Remove_Attribute_Value(CName,PName,Value);
                    fprintf(cgiOut,OKFLAG);
        }else
        {
                   fprintf(cgiOut,"Sorry: Function %s is not Implimented",Functio
        }
        return 1;
}


void List_All_Concepts()
{
       int number=0;
        char **list;
        char CName[200];
        MV_List_All_Concept(number,list);
        if(number == 0) {
       fprintf(cgiOut," Vocabulary doesn't have Any Concept\n");
        } else {
                    fprintf(cgiOut,OKFLAG);
   for(int i=0;i<number;i++)
                             {
                                if(list[i])
                                    fprintf(cgiOut,"%s\n",list[i]);
                                else
                                    fprintf(cgiOut,"\n");
                             }
   mv_free(number,list);
                 }
}

void Child_Parent(char *parent,int FLAG)
{
int number=0;
char **list;
```

```
            switch(FLAG)
            {
               case CHILDREN:
MV_List_Children(parent,number,list);
                 break;
               case PARENT:
                 MV_List_Parents(parent,number,list);
                 break;
            }
            if(number == 0)
                 {
                    fprintf(cgiOut,CONTINUE);
   fprintf(cgiOut,
                      " Concept %s don't have Children/Parent\n",parent);
}
                  else
                 {
                    fprintf(cgiOut,OKFLAG);
   for(int i=0;i<number;i++)
                             {
                                if(list[i])
                                    fprintf(cgiOut,"%s\n",list[i]);
                                else
                                    fprintf(cgiOut,"\n");
                             }
   mv_free(number,list);
}
}

void AttribValue(char *CName,char *PName,char FLAG)
{
        int number=0;
        char **list;
          switch(FLAG)
          {
             case 0:
                MV_Show_Attribute_Value(CName,PName,number,list);
                break;
             case 1:
                 MV_List_All_Attribute_Value(CName,number,list);
                 break;
             case 2:
                 MV_Show_Relationship_Value(CName,PName,number,list);
                 break;
```

```
            case 3:
                MV_List_All_Relationship_Value(CName,number,list);
                break;
        }
          if(number == 0) {
                       fprintf(cgiOut,"%s\n","Dose'nt have value");
              } else {
                 fprintf(cgiOut,OKFLAG);
                 for(int i=0;i<number;i++)
                     if(list[i])
                        fprintf(cgiOut,"%s\n",list[i]);
                     else
                        fprintf(cgiOut,"\n");
                 mv_free(number,list);
              }
}
```

APPENDIX B

JAVA CLASSES AND THEIR CODE

The code for the IVC is presented here.

```
/**
*   File: newoohvr.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import EDU.auburn.VGJ.gui.*;
import EDU.auburn.VGJ.graph.*;
import EDU.auburn.VGJ.algorithm.tree.*;
import EDU.auburn.VGJ.algorithm.GraphAlgorithm;
import java.util.Stack;

/**
*    Main Applet class which activates the Schema Frame. And Initializes
*    the CGI class
*/

public class newoohvr extends Applet
{
    OohvrSchema mainSchema;
    GraphWindow gwin;
    int nodeno;
    NewFrame SchemaFrame;
    SchemaCanvas gCnv;
    double view;
    /**
    *       CGI variable
    */
    OOhvrCGI cgi;
    final int OVAL=1;
    final int RECTANGEL=2;
```

```
    /**
    *   Main initialization function which actvates the CGI and Schema frame
    */
    public void init()
    {
      nodeno=0;
      cgi=new OOhvrCGI("http://object.njit.edu:2000","~arif/oohvrC.cgi",this);
                             // Creating an Instance of cgi whith http
                             // server and oohvrC.cgi as CGI prog. name.
      view=20;              //  Initial value of view
      String root=cgi.GetRoot(); // Getting the root of Vocabulary
      mainSchema=new OohvrSchema(root,RECTANGEL,"IN");
// Creating Schema as Directed graph
      int rootId=mainSchema.getRootId();
      String str[]=cgi.GetChild(root);

      int k=0;
      int no_of_children = cgi.ReadTotal();

      for(int i=0;i<no_of_children;i++)
      {
            k++;
            int id=mainSchema.AddChild(root,str[i]);
            System.out.println(k+":"+str[i]);

      }

      System.out.println("Total  : "+k+" Generated");
      SchemaFrame=new NewFrame(mainSchema,cgi);
      SchemaFrame.resize(600,600);
      gCnv=new SchemaCanvas(mainSchema,SchemaFrame);
      SchemaFrame.UpdateCanvas(gCnv);  // just updates Schemacanvas variable
      ScrolledPanel vPanel=new ScrolledPanel(gCnv);
      SchemaFrame.add("Center",vPanel);
      gCnv.setMouseMode(gCnv.SELECT_NODES);
      SchemaFrame.pack();
      SchemaFrame.show();
      SchemaFrame.Refreash();

    }

}
```

```java
/**
*   File: OohvrSchema.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import EDU.auburn.VGJ.gui.*;
import EDU.auburn.VGJ.graph.*;
import EDU.auburn.VGJ.algorithm.tree.*;
import EDU.auburn.VGJ.algorithm.GraphAlgorithm;
import java.util.Stack;

/**
*   The schema is a Directed graph b/w diffrent concepts. Showing the
*   relationship b/w them.
*/

public class OohvrSchema extends Graph
{
  private int CShape;
  private String CLabelPosition;
  private int RootId;
  Hashtable NameToIndex;
  /*
  * Schema Constructor which takes Root of the schema and shape type for
  * Concept displaying. Two options OVAL and RECTANGLE are currently
  * available.
  * And the position for the concept names, IN, OUT are available.
  */
  public OohvrSchema(String root,int Shapev,String pos)
  {
    super(true);
    NameToIndex=new Hashtable();
    CLabelPosition=pos;
    Node N=new Node();
    if(Shapev==1)
        CShape=N.OVAL;
    else
        CShape=N.RECTANGLE;
```

```
  RootId=AddNewNode(root);
}
/**
*   Returns the current root of the schema
*/
public int getRootId()
{
   return RootId;
}
/**
*  Sets Id as new root.
*/
public void setRootId(int id)
{
   RootId=id;
}
/**
* Adds a new node in the graph by having concept name.
*/
int AddNewNode(String Labelv)
{
    int id;
    id=insertNode();
    Node NewNode=getNodeFromIndex(id);
    NewNode.setShape(CShape);
    NewNode.setLabel(Labelv);
    NewNode.setPosition(10,10);
    NewNode.setLabelPosition(CLabelPosition);
    System.out.println(id+":"+Labelv+" In Add");
    NameToIndex.put(Labelv,new Integer(id));
    return id;
}

/**
*   Adds new child when Parent Id is given with new child's name.
*   Returns the new child's ID.
*/
int AddChild(int ParentId,String Child)
{
       Integer inVal;
       int id;
       inVal=(Integer)NameToIndex.get(Child);
       if(inVal==null)
          id=AddNewNode(Child);
```

```
        else
           id=inVal.intValue();
        AddEdge(id,ParentId);
        System.out.println(id+":"+Child+" In AddChild");
        System.out.println("Size of hash table:"+NameToIndex.size());

        return id;
}


/**
* Adds a new child by taking Parent name and Child name
* Returns the new child ID as return value.
*/
public int AddChild(String Parent,String Child)
{
      int id;
      int no=((Integer)NameToIndex.get(Parent)).intValue();
      System.out.print("Adding to "+no+" i.e "+Parent);

      return AddChild(no,Child);
}
/**
* Adds a new edge b/w two given nodes while nodes Ids are given.
*/
void AddEdge(int idParent,int idChild)
{
  insertEdge(idParent,idChild);
}
/**
* Adds a new edge b/w two given nodes while nodes names are given.
*/
public void AddEdge(String Parent,String Child)
{
  insertEdge(((Integer)NameToIndex.get(Parent)).intValue(),
             ((Integer)NameToIndex.get(Child)).intValue());
}
public void LabelEdge(String Parent,String Child, String label)
{
   Edge eg=getEdge(((Integer)NameToIndex.get(Parent)).intValue(),
           ((Integer)NameToIndex.get(Child)).intValue());
   eg.setLabel(label);
}
```

```
public void removeEdge(String source, String destination)
{
     int Source_id = ((Integer)NameToIndex.get(source)).intValue();
     int Dest_id = ((Integer)NameToIndex.get(destination)).intValue();

     removeEdge(Source_id,Dest_id);
}
// added by Gowtham on 07/09/97
/**
*  This function removes a child from the present schema given the
*  the parent name and the child name
*/
public void RemoveChild(String parent, String child)
{
     int child_id,parent_id;

     child_id = ((Integer)NameToIndex.get(child)).intValue();
     parent_id = ((Integer)NameToIndex.get(parent)).intValue();

     //cleanup
     removeEdge(child_id,parent_id);
     removeNode(getNodeFromIndex(child_id));
     if(NameToIndex.remove(child) == null) // removes entry from hashtable
     {
          System.out.println("Shucks");
     };
     System.out.println("Number of elements in the hash table"+
                 NameToIndex.size());
}



 public boolean isNodepresent(String name)
 {
      return NameToIndex.containsKey(name);
 }

 public Edge getEdge(String str1, String str2)
 {
     int id1 = ((Integer)NameToIndex.get(str1)).intValue();
     int id2 = ((Integer)NameToIndex.get(str2)).intValue();

     return getEdge(id1,id2);
}
```

```
  public int AddParent(String Child, String Parent)
  {
       Integer inVal = (Integer) NameToIndex.get(Parent);
       int child_id,parent_id;

       //obtain parent id ( allocate node if parent not present)
       if(inVal == null)  // node does not exist
       {
            parent_id = AddNewNode(Parent);
       }
       else
       {
            parent_id = inVal.intValue();
       }
       // obtain child id
       child_id = ((Integer) (NameToIndex.get(Child))).intValue();
       AddEdge(child_id,parent_id);

       return parent_id;

  }

  public int get_nodeid(String str)
  {
       Integer id =  (Integer) (NameToIndex.get(str));
       if(id != null)
       {
            return id.intValue();
       }

       return -1;

  }
}

/**
*   File: OOhvrCGI.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/
import java.net.*;
```

```
import java.io.*;
import java.util.*;
import java.applet.Applet;
import java.awt.*;
import EDU.auburn.VGJ.gui.MessageDialog;

/**
*    This class provide transparent inteface to API's using CGI calls
*/

public class OOhvrCGI
 {
        int TotalRead;
        String Server,cgiName;
        Applet oohvr;
        /**
        *   Indicator about the status of last CGI call
        */
        public boolean OK;
        /**
        *   Constructor takes HTTP server name, CGI file and path name and
        *   the calling Applet reffrence to show the status.
        */
        public OOhvrCGI(String sr,String cgiNam,Applet ohvr)
        {
               Server=sr;
               OK=false;
               cgiName=cgiNam;
               oohvr=ohvr;
        }
       /**
       *    Actual CGI call to server takes the call string as input.
       */
       String[] CallServer(String CallString)
       {
           Frame fr2=new Frame();
           Message msg=new Message(fr2,"Wait","Contacting to "+Server 12 +
                                 " ..... ",false);
           //MessageDialog(fr2,"Wait","Getting Data ....",true);
           msg.show();
           System.out.println("Calling : "+CallString);
           TotalRead=0;
           OK=false;
           oohvr.showStatus("Connecting to "+Server+" ...... ");
```

```
        CallString=Server+"/"+cgiName+"?"+CallString;
        String Str[]=new String[4000];
        try {
              URL url = new URL(CallString);

              DataInputStream URLinStream=new DataInputStream
                                         (url.openStream());
              String tmpStr;
              oohvr.showStatus("Connected Waiting for Data .... " );
              msg.UpdateMessage("Contected Waiting for Data . . . . .");
              boolean flg=false;
              tmpStr = URLinStream.readLine();
              if(tmpStr.equals("OK")) // OK would come from server
              {
                if(flg)
                {
                   msg.UpdateMessage("Retrieving Data . . . . .");
                   flg=false;
                 }
                 else
                 {
                   msg.UpdateMessage("Retrieving Data  . . . . .");
                   flg=true;
                 }
                 while((tmpStr = URLinStream.readLine())!=null)
Str[TotalRead++]=tmpStr;
                 OK=true;
              }
              else
              {
                 if(!tmpStr.equals("CONTINUE"))
                 {
                  Frame fr=new Frame();
                  MessageBox mbox=new MessageBox(fr,"Error!",
                                 "Error Message from Server: "+tmpStr,true);
                  mbox.show();
                 }
                 OK=true;
              }
              oohvr.showStatus("Data Retrieved " );
              msg.End();
              URLinStream.close();
            } catch (MalformedURLException mexp)
                 {
```

```
                Frame fr=new Frame();
                MessageBox mbox=new MessageBox(fr,"Exception!",
                         "Exception Occured during Data retriveval:
                          mexp,true);
                mbox.show();
                    System.err.println("MalFormedURL: "+mexp);
                    System.out.println("Called : "+CallString);
             }
             catch (IOException ioexp)
             {
                Frame fr=new Frame();
                MessageBox mbox=new MessageBox(fr,"Exception!",
                                "Exception Occured during Data''
  `` retriveval:  "
                                +ioexp,true);
                mbox.show();
                System.err.println("I/O Exception : "+ioexp);
                System.out.println("Called : "+CallString);
             }
        return Str;
}
/**
*    Returns the total number of lines read from last API call.
*/
 public int ReadTotal()
 {
     return TotalRead;
 }
 /**
 *  Returns the root of Vocabulary. It's static value ENTITY for
 *  this version which cab be dynamic by adding a CGI call to
 *  server in future.
 */
 public String GetRoot()
 {
      return "ENTITY";
 }

 /**
 *    API: Returns the properties of given concept.
 */
 public String[] GetProperties(String conc)
 {
      String Str[];
```

```
            String CallString;
            CallString="FUNCTION=AllAttribValue&CONCEPT="+
URLEncoder.encode(conc);
            Str=CallServer(CallString);
            return Str;
      }
      /**
      *   API: Returns all the childern of given concept.
      */
      public String[] GetChild(String parent)
      {
         String Str[];
         TotalRead=0;
         String CallString;
         CallString="FUNCTION=CHILDREN&PARENT="+URLEncoder.encode(parent);
         Str=CallServer(CallString);
         return Str;
      }
      /**
      *  API: Returns all relationships from a concept.
      */
      public String[] GetRelations(String conc)
      {
            String Str[];
            String CallString;
            CallString="FUNCTION=AllRelationValue&CONCEPT="+
URLEncoder.encode(conc);
            Str=CallServer(CallString);
            return Str;
      }
      /**
      *  API: Deletes a given concept from the Vocabulary.
      */
      public String[] DeleteConcept(String conc)
      {
            String Str[];
            String CallString;
            CallString="FUNCTION=DeleteConcept&CONCEPT="+
URLEncoder.encode(conc);
            Str=CallServer(CallString);
            return Str;
      }
      /**
      *   API: Get all the concepts available in the Vocabulary.
```

```
        */
        public String[] GetAllConcepts()
        {
             String Str[];
             String CallString;
             CallString="FUNCTION=AllConcepts";
             Str=CallServer(CallString);
             return Str;
        }
        /**
        *   API: Get the parent of a given concept.
        */
       public String[] GetParent(String child)
       {
           TotalRead=0;
           String CallString;
           CallString="FUNCTION=PARENTS&CHILD="+
URLEncoder.encode(child);
           String Str[];
           Str=CallServer(CallString);
           return Str;
      }
        /**
        *   API: Add a new child to given parent concept.
        */
      public void AddNewChild(String Parent,String Child)
      {
           TotalRead=0;
           String CS;
           CS="FUNCTION=AddChild&CHILD="+URLEncoder.encode(Child);
           CS=CS+"&"+"PARENT="+URLEncoder.encode(Parent);
           String Str[];
           Str=CallServer(CS);
      }
        /**
        *   API: Add A new property to given concept.
        */
      public void AddNewProperty(String Concept,String pName)
      {
           TotalRead=0;
           String CS;
           CS="FUNCTION=AddProperty&CONCEPT="+URLEncoder.encode(Concept);
           CS=CS+"&"+"PROPERTY="+URLEncoder.encode(pName);
           String Str[];
```

```
        Str=CallServer(CS);
}
  /**
  *   API: Change the property value of given concept provided that
  *   the old value to concept is also given.
  */
public void ChangePropertyValue(String CName,String PName,
                                          String LVal,String NVal)
{
        TotalRead=0;
        String CS;
        CS="FUNCTION=ChangeAttribValue&CONCEPT="+URLEncoder.encode(CName);
        CS=CS+"&"+"ATTRIBUTE="+URLEncoder.encode(PName);
        CS=CS+"&"+"OLDVAL="+URLEncoder.encode(LVal);
        CS=CS+"&"+"VALUE="+URLEncoder.encode(NVal);
        String Str[];
        Str=CallServer(CS);
}
  /**
  *   API: Change the relationship value of given concept provided that
  *   the old value of relationship is also given.
  */
public void ChangeRelationValue(String CName,String PName,
                          String LVal,String NVal)
{
        TotalRead=0;
        String CS;
        CS="FUNCTION=ChangeRelationValue&CONCEPT="+URLEncoder.encode(CName);
        CS=CS+"&"+"ATTRIBUTE="+URLEncoder.encode(PName);
        CS=CS+"&"+"OLDVAL="+URLEncoder.encode(LVal);
        CS=CS+"&"+"VALUE="+URLEncoder.encode(NVal);
        String Str[];
        Str=CallServer(CS);
}
  /**
  *   API: Adding a new Relationship to a Concept target concept is
  *   also given.
  */

public void AddRelationValue(String CName,String rName,
                        String tConceptName)
{
        TotalRead=0;
        String CS;
```

```
        CS="FUNCTION=AddRelationshipValue&CONCEPT="+URLEncoder.encode(CName);
        CS=CS+"&"+"Relation="+URLEncoder.encode(rName);
        CS=CS+"&"+"Target="+URLEncoder.encode(tConceptName);
        String Str[];
        Str=CallServer(CS);
    }
      /**
      *   API: Creating a new Relationship to a Concept target concept is
      *   and reverse relationship name  is also given.
      */

    public void CreateNewRelation(String CName,String rName,
                          String tCName,String rrName)
    {
        TotalRead=0;
        String CS;
        CS="FUNCTION=CreateNewRelation&CONCEPT="+URLEncoder.encode(CName);
        CS=CS+"&"+"RELATION="+URLEncoder.encode(rName);
        CS=CS+"&"+"Target="+URLEncoder.encode(tCName);
        CS=CS+"&"+"RevRELATION="+URLEncoder.encode(rrName);
        String Str[];
        Str=CallServer(CS);
    }
      /**
      *   API: Deleting an attribute from a Concept.
      */
    public void DeleteAttribute(String CName,String PName)
    {
        TotalRead=0;
        String CS;
        CS="FUNCTION=DeleteAttribute&CONCEPT="+URLEncoder.encode(CName);
        CS=CS+"&"+"ATTRIBUTE="+URLEncoder.encode(PName);
        String Str[];
        Str=CallServer(CS);
    }
      /**
      *   API: Deleting  relationship from a Concept.
      */
    public void DeleteRelation(String CName,String rName,
String tCName,String rrName)
    {
        TotalRead=0;
        String CS;
        CS="FUNCTION=DeleteRelation&CONCEPT="+URLEncoder.encode(CName);
```

```
        CS=CS+"&"+"RELATION="+URLEncoder.encode(rName);
        CS=CS+"&"+"Target="+URLEncoder.encode(tCName);
        CS=CS+"&"+"RevRELATION="+URLEncoder.encode(rrName);
        String Str[];
        Str=CallServer(CS);
    }
      /**
      *   API: Removing an Attribute value   from a Concept value value
      *   to that property is also provided.
      */

    public void RemoveAttributeValue(String CName,String PName,String PValue)
    {
        TotalRead=0;
        String CS;
        CS="FUNCTION=RemoveAttributeValue&CONCEPT="+URLEncoder.encode(CName);
        CS=CS+"&"+"ATTRIBUTE="+URLEncoder.encode(PName);
        CS=CS+"&"+"VALUE="+URLEncoder.encode(PValue);
        String Str[];
        Str=CallServer(CS);
    }
      /**
      *   API: Removing a  Relationship from a Concept
      *   relationship name  is also given.
      */
    public void RemoveRelationValue(String CName,String rName,String Value)
    {
        TotalRead=0;
        String CS;
        CS="FUNCTION=RemoveRelationValue&CONCEPT="+URLEncoder.encode(CName);
        CS=CS+"&"+"RELATION="+URLEncoder.encode(rName);
        CS=CS+"&"+"VALUE="+URLEncoder.encode(Value);
        String Str[];
        Str=CallServer(CS);
    }
};


import java.awt.*;

public class NList extends List
{
     String prop[];
     PropertyDialog pd;
```

```
PropertyEditor ped;

public NList(int no,boolean v,String pro[],PropertyDialog pdb)
{
  super(no,v);
  prop=pro;
  pd=pdb;
}

public NList()
{
    super();
}

public NList(int rows, boolean multipleSelections)
{
     super(rows,multipleSelections);
}


public boolean mouseDown(Event ev,int x,int y)
{
   //if(ev.clickCount==2)
   System.out.println("You are Editing "+getSelectedItem());
   return true;
}

// deselectAll deselects all selected items in the list
public void deselectAll()
{
     int i;
     for(i=0; i< getItemCount(); i++)
     {
          if(isSelected(i) == true)
          {
               deselect(i);
          }
     }
}

public boolean is_Itempresent(String str)
{
     for(int i=0; i<countItems(); i++)
     {
```

```
                if(str.equalsIgnorecase(getItem(i)))
              return true;
            }
            return false;
        }
}


import java.awt.List;

public class Enhanced_list extends List
{
    public Enhanced_list()
    {
        super();
    }

    public Enhanced_list(int x)
    {
        super(x);
    }

    public Enhanced_list(int x, boolean bool)
    {
        super(x,bool);
    }

    public boolean isPresent(String str)
    {
        for(int i=0; i<countItems(); ++i)
        {
            if(str.equals(getItem(i)) == true)
            {
                return true;
            }
        }
        return false;
    }
}

/**
*   File: MessageBox.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
```

```
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/

import java.awt.*;

/**
*   Message box  which shows messages to user.
*/

public class Message extends Frame
{

      Label Msg;

      /**
      * MessgageBox gets Frame and String for heading and boolean true
      * for showing buttons or not, To diffrentiate b/w Status messages
      * of Data transfer or Error message.
      */
      public Message(Frame par,String Head,String Message,boolean bt)
      {
         super(Head);
         Panel pnl=new Panel();
         Msg=new Label(Message);
         pnl.add(Msg);
         add("Center",pnl);
         Panel pnl2=new Panel();
         if(bt)
         {
            pnl2.add(new Button("Ok"));
            pnl2.add(new Button("Cancel"));
         }
         add("South",pnl2);
         resize(300,150);
         move(250,250);
      }

      /**
      * event handler.
      */
       public boolean handleEvent(Event ev)
       {
         if(ev.id == Event.ACTION_EVENT)
```

```
          {
             if("Ok".equals(ev.arg))
             {
                   show(false);
                   dispose();
                   return true;
             }

             if("Cancel".equals(ev.arg))
             {
                   show(false);
                   dispose();
                   return true;
             }
          }
          return false;
        }

        public void UpdateMessage(String str)
        {
           Msg.setText(str);
        }

        public void End()
        {
            dispose();
        }
};

/**
*   File: MessageBox.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/

import java.awt.*;

/**
*   MessageBox dialog which shows messages to user.
*/

public class MessageBox extends Dialog
```

```
{

    Label Msg;
    private boolean Okclicked;
    /**
    * MessgageBox gets Frame and String for heading and boolean true
    * for showing buttons or not, To diffrentiate b/w Status messages
    * of Data transfer or Error message.
    */
    public MessageBox(Frame par,String Head,String Message,boolean bt)
    {
       super(par,Head);
       Panel pnl=new Panel();
       Msg=new Label(Message);
       pnl.add(Msg);
       add("Center",pnl);
       Panel pnl2=new Panel();
       if(bt)
       {
          pnl2.add(new Button("Ok"));
          pnl2.add(new Button("Cancel"));
       }
       add("South",pnl2);
       resize(300,150);
       Okclicked = false;
    }
    /**
    * event handler.
    */
     public boolean handleEvent(Event ev)
     {
       if(ev.id == Event.ACTION_EVENT)
        {
           if("Ok".equals(ev.arg))
           {
                 Okclicked = true;
                 show(false);
                 return true;
           }

           if("Cancel".equals(ev.arg))
           {
                 Okclicked = true;
                 show(false);
```

```
                return true;
            }
        }
        return false;
    }

    public void UpdateMessage(String str)
    {
       Msg.setText(str);
    }

    public void End()
    {
        dispose();
    }

    public boolean Ok_clicked()
    {
         return Okclicked;
    }
};


/**
*   File: NewChild.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import EDU.auburn.VGJ.gui.*;
import EDU.auburn.VGJ.graph.*;
import EDU.auburn.VGJ.algorithm.tree.*;
import EDU.auburn.VGJ.algorithm.GraphAlgorithm;
import java.util.Stack;

/**
*   A dialog box to take new Child name from the user.
*/
public class NewChild extends Frame
```

```
{
        NewFrame ooh;
        TextField fld;
        String Parent;
        Panel pnl,pl;
        /**
        *  Constructor takes Parent of the new child and main frame reffrence.
        */
        public NewChild(String pParent,NewFrame ohv)
        {
             super("New Child of "+pParent);
             setLayout(new BorderLayout());
             ooh=ohv;
             pnl=new Panel();
             pnl.setLayout(new FlowLayout());
             Parent=pParent;
             pnl.add(new Label("Child's Name : "));
             fld=new TextField(30);
             pnl.add(fld);
             add("Center",pnl);
             pl=new Panel();
             pl.add("Center",new Button("OK"));
             pl.add("Center",new Button("Cancel"));
             add("South",pl);
             pack();
             move(200,200);
             show();
        }
        /**
        *   Envent handler for the dialog.
        */
        public boolean handleEvent(Event ev)
        {
          if(ev.id == Event.ACTION_EVENT)
           {
              if("OK".equals(ev.arg))
              {
                     ooh.AddNewChild(Parent,fld.getText());
                     dispose();
                     return true;
              }else if("Cancel".equals(ev.arg))
               {
                     dispose();
                     return true;
```

```
                }
            }
            return false;
        }

}


import java.util.*;
import java.awt.*;
import java.awt.event.*;


class PropertyEditor extends Dialog
{
//          PropertyDialog pd;
        TextField fld;
        String name,oval;
        Panel pnl,pl;
//          NList mylist;
        private boolean Ok_clicked,Cancel_clicked;

        public PropertyEditor(String Name,String Value,Frame fr)
        {
             super(fr,true);
             setTitle("Properties/relationships editor");
             setLayout(new BorderLayout());
//               pd = pdb;
             pnl=new Panel();
             pnl.setLayout(new FlowLayout());
             name=Name;
             oval=Value;
//               mylist = li;
             pnl.add(new Label(Name+" : "));
             System.out.println("Length of String "+Value.length());
             fld=new TextField(Value,35);
             pnl.add(fld);
             add("Center",pnl);
             pl=new Panel();
             pl.add(new Button("OK"));
             pl.add(new Button("Cancel"));
             add("South",pl);
             pack();
             resize(preferredSize());
```

```
            System.out.println("Property Editor Should PopOut");

            //initialize button related variables
    Ok_clicked = false;
    Cancel_clicked = false;
       }


       /*  Event handler */

       public boolean handleEvent(Event ev)
       {
         if(ev.id == Event.ACTION_EVENT)
         {
             if("OK".equals(ev.arg))
             {
//                   pd.UpdateProperty(name,oval,fld.getText(),mylist);
//                   dispose();
            Ok_clicked = true;
   show(false);
                    return true;
             }

             if("Cancel".equals(ev.arg))
             {
//                   dispose();
           Cancel_clicked = true;
  show(false);
                   return true;
             }
         }
         return false;
      }

      public boolean is_Okclicked()
      {
           return Ok_clicked;
      }

      public boolean is_Cancelclicked()
      {
           return Cancel_clicked;
      }
```

```
        public String getText()
        {
             return fld.getText();
        }

}

class NewRelationshiptype extends Dialog
{
     //declare dialog components

     TextField text;
     Label label;
     Panel panel,bpanel;
     private boolean Ok_clicked,Cancel_clicked;

     public NewRelationshiptype(Frame fr)
     {
  super(fr);
          // initialize components
          text = new TextField(20);
          label = new Label("Enter new RelationShip type");
          panel = new Panel();
          bpanel = new Panel();

          panel.add(label);
          panel.add(text);

          bpanel.add(new Button("OK"));
          bpanel.add(new Button("CANCEL"));

          add("South",bpanel);
          add("Center",panel);

   // initialize button related variables
          Ok_clicked = false;
  Cancel_clicked = false;
     }

     public boolean handleEvent(Event evt)
     {
          if("OK".equals(evt.arg))
          {
               return true;
```

```
        }

        if("CANCEL".equals(evt.arg))
        {
             dispose();
             return true;
        }
  return false;
     }
}



class NewProperty extends Dialog
{
       PropertyDialog pdlg;
       TextField fld;
       String Parent;
       Panel pnl,pl;
       private boolean Ok_clicked,Cancel_clicked;

       public NewProperty(String Concept,PropertyDialog pd,Frame fr)
       {
     //       super(fr,"New Property of "+Concept);
            super(fr,true);
            setTitle("New Property of "+Concept);
            setLayout(new BorderLayout());
            pdlg=pd;
            pnl=new Panel();
            pnl.setLayout(new FlowLayout());
            Parent=Concept;
            pnl.add(new Label("New Property Name : "));
            fld=new TextField(30);
            pnl.add(fld);
            add("Center",pnl);
            pl=new Panel();
            pl.add(new Button("OK"));
            pl.add(new Button("Cancel"));
            add("South",pl);
            pack();

    Ok_clicked = false;
    Cancel_clicked = false;
       }
```

```
        public boolean handleEvent(Event ev)
        {
          if(ev.id == Event.ACTION_EVENT)
          {
              if("OK".equals(ev.arg))
              {
 //                     pdlg.AddNewProperty(Parent,fld.getText());
      //                dispose();
    Ok_clicked = true;
    if((fld.getText()).length() != 0)
    {
         show(false);
    }
                    return true;
              }

       if("Cancel".equals(ev.arg))
               {
//                      dispose();
    Cancel_clicked = true;
    show(false);
                    return true;
               }
          }
          return false;
        }

      public String getText()
      {
            return fld.getText();
      }

      public boolean is_Okclicked()
      {
           return Ok_clicked;
      }

      public boolean is_Cancelclicked()
      {
           return Cancel_clicked;
      }

}
```

```
class NewRelation extends Dialog
{
//        PropertyDialog pdlg;
       TextField fld;
       NList conList,relList;
       String Parent;
       Panel pnl,pl;
       OOhvrCGI cgi;
       private boolean Ok_clicked,Cancel_clicked;


       public NewRelation(String Concept,OOhvrCGI cg,Frame fr)
       {
            super(fr,true);
            setTitle("New Relation of "+Concept);
            cgi=cg;
            setLayout(new BorderLayout());
//            pdlg=pd;
            pnl=new Panel();
            pnl.setLayout(new FlowLayout());
            Parent=Concept;
            conList=new NList(10,false);
            String str[]=cgi.GetAllConcepts();
            for(int i=0;i<cgi.ReadTotal();i++)
                     conList.addItem(str[i]);
            relList=new NList(10,false);
            String str2[]=cgi.GetRelations(Concept);
            for(int i=0;i<cgi.ReadTotal();i+=2)
    {
if(relList.is_Itempresent(str2[i]) == false)
{
                     relList.addItem(str2[i]);
}
    }

            //pnl.add(new Label("Relation Name: "));
            //fld=new TextField(30);
            pnl.add(relList);
            add("North",pnl);
            add("Center",conList);
            pl=new Panel();
            pl.add(new Button("OK"));
            pl.add(new Button("Cancel"));
```

```
            pl.add(new Button("New Relation"));

            add("South",pl);
            pack();

    Ok_clicked = false;
    Cancel_clicked = false;
       }

       public boolean handleEvent(Event ev)
       {
         if(ev.id == Event.ACTION_EVENT)
          {
             if("OK".equals(ev.arg))
             {
/*                   cgi.AddRelationValue(Parent,relList.getSelectedItem(),
                          conList.getSelectedItem());
                   if(cgi.OK == true)
                   {
                 //        dispose();
  show(false);
   }
   else
   {
        MessageBox msg = new MessageBox(new Frame(),
                                     "Warning","CGI operation failed",true);
                   } */
   show(false);
   Ok_clicked = true;
                   return true;
             }

      if("Cancel".equals(ev.arg))
             {
//                   dispose();
   show(false);
   Cancel_clicked = true;
                   return true;
             }

             if("New Relationship".equals(ev.arg))
             {

  System.out.println("New Relationship button clicked");
```

```
                    NewRelationshiptype xyz = new
 NewRelationshiptype(new Frame());
                    return true;
     }
          }
          return false;
        }

public boolean is_Okclicked()
        {
     return Ok_clicked;
}

        public boolean is_Cancelclicked()
        {
     return Cancel_clicked;
}

   public String get_Selectedrelation()
        {
               return relList.getSelectedItem();
        }

        public String get_Selectedconcept()
{
      return conList.getSelectedItem();
}
}


public class PropertyDialog extends Dialog // implements ActionListener
{
       OOhvrCGI cgi;
       TextField fld;
       PropertyEditor ped;
       NewFrame aplt;
       Panel pnl,lpnl;
       NList propList,relList;
       String pro[];
       String rel[];
       String Name;
       Button ok,edit,del,add;
       int Active_list_box;
```

```
    boolean Dialog_active;
    boolean Ok_clicked;

    String[] selection_buffer = new String[50];
    int changed;

    public PropertyDialog(NewFrame ap,OOhvrCGI ocgi,String conc,
                String properties[],String relations[],int tot1,int tot2)
    {
        super(ap,true);
        setTitle("Properties and Relationships of "+conc);
        aplt=ap;
        System.out.println("Starting Property and Attributes Dialog");
        pro=properties;
        rel=relations;

        Name=conc;
        setLayout(new BorderLayout());

        pnl=new Panel();
        lpnl=new Panel();
        Label centerlabel = new Label("^ Attributes  v Relationships",
                                Label.CENTER);
        System.out.println("Panel Created");
        lpnl.setLayout(new BorderLayout());

        // button panel
        ok = new Button("OK");
        edit = new Button("Edit");
        del = new Button("Delete");
        add = new Button("Add");

        pnl.add(ok);
        pnl.add(edit);
        pnl.add(del);
        pnl.add(add);

        System.out.println("Creating List");

        propList = new NList(10,false,properties,this);
        relList  = new NList(10,false,relations,this);

        propList.select(0);   // Selects the first
//Item in the properties list
```

```
            Active_list_box = 0;

            propList.setName("Properties");
            relList.setName("Relations");

            // add the action listeners
//            propList.addActionListener(this);
//            relList.addActionListener(this);

            // add the focus listener to the class components

            propList.addFocusListener(new fAdapt(this));
            relList.addFocusListener(new fAdapt(this));

            // add the items to the list boxes
            for(int i=0;i<tot1;i+=2)
               propList.addItem(properties[i]+"="+properties[i+1]);
            for(int i=0;i<tot2; i+=2)
               relList.addItem(relations[i]+"="+relations[i+1]);

            System.out.println("List Updated");

            lpnl.add("North",propList);
            lpnl.add("Center",centerlabel);
            lpnl.add("South",relList);

            add("Center",lpnl);
            add("South",pnl);

            cgi=ocgi;

    Dialog_active = false;
    Ok_clicked = false;
    changed = 0;
            for(int j=0; j<50; j++)
            {
                 selection_buffer[j] = null;
            }
      }

      public void UpdateProperty(String name,String oval,String nval,NList li)
      {
         if(li == propList)
         {
```

```
            cgi.ChangePropertyValue(Name,name,oval,nval);
        }
        else
        {
            cgi.ChangeRelationValue(Name,name,oval,nval);
        }
        UpdatePropertyValues(li);
    }

    void UpdatePropertyValues(NList li)
    {
        String properties[];
        if(li == propList)
        {
            properties=cgi.GetProperties(Name);
            propList.clear();
            for(int i=0; i<cgi.ReadTotal(); i+=2)
                propList.addItem(properties[i]+"="+properties[i+1]);
        }
        else
        {
            properties=cgi.GetRelations(Name);
            relList.clear();
            for(int i=0;i<cgi.ReadTotal();i+=2)
               relList.addItem(properties[i]+"="+properties[i+1]);
        }
    }

    public boolean is_Okclicked()
    {
          return Ok_clicked;
    }

    public boolean handleEvent(Event ev)
    {
      if(ev.id == Event.ACTION_EVENT)
      {

           // if the ok button is depressed
           if("OK".equals(ev.arg))
           {
                  Ok_clicked = true;
                  show(false);
                  return true;
```

```
            }


 /*             if("Ok".equals(ev.arg)) // Ok from Message Box
       //TAKE CARE OF IT IN THE END
            {

                  if(ev.target == (Object) propList)
                  {
                        String tmp=propList.getSelectedItem();
                        cgi.RemoveAttributeValue(Name,
                            tmp.substring(0,tmp.indexOf('=')),
                            tmp.substring(tmp.indexOf('=')+1,tmp.length()));
                        UpdatePropertyValues(propList);
                  }
                  else
                  {
                        String tmp=relList.getSelectedItem();
                        cgi.RemoveRelationValue(Name,
                            tmp.substring(0,tmp.indexOf('=')),
                            tmp.substring(tmp.indexOf('=')+1,tmp.length()));
                        UpdatePropertyValues(relList);
                  }

                  return true;
            }

*/
            // If the edit button is depressed
            if("Edit".equals(ev.arg))
            {
                Frame fr = new Frame();
                if((Active_list_box == 0) && (Dialog_active == false))
                {
       Dialog_active = true;
                      String tmp=propList.getSelectedItem();
                      ped=new PropertyEditor(tmp.substring(0,
tmp.indexOf('=')),tmp
                                  .substring(tmp.indexOf('=')+1,
tmp.length()),fr);
                      ped.setModal(true);
                      ped.pack();
                      ped.move(200,200);
                      ped.show(true);
```

```
System.out.println("Comes here");
       // button handlers for "ped"
                       if(ped.is_Okclicked() == true)
          {
               System.out.println("Comes here too");
                            UpdateProperty(tmp.substring(0,tmp.indexOf('=')),
                                        tmp.substring(tmp.indexOf('=')+1
,tmp.length()),
                                        ped.getText(),propList);
                     System.out.println("It works !!!");
    Dialog_active = false;
    ped.show(false);
    ped.dispose();
             }

       if(ped.is_Cancelclicked() == true)
                       {
    System.out.println("It works!!!");
    ped.dispose();
    Dialog_active = false;
       }
                 }

 if((Active_list_box ==1) && (Dialog_active == false))
                 {
       Dialog_active = true;
                       String tmp=relList.getSelectedItem();
                       ped=new PropertyEditor(tmp.substring(0,
     tmp.indexOf('=')),
                                     tmp.substring(tmp.indexOf('=')+1
                                     ,tmp.length()),fr);
                       ped.setModal(true);
                       ped.pack();
                       ped.move(200,200);
                       ped.show(true);

       // button handlers
       if(ped.is_Okclicked() == true)
       {
                      System.out.println("It works !!!");
                            UpdateProperty(tmp.substring(0,tmp.indexOf('=')),
                                      tmp.substring(tmp.indexOf('=')+1
        ,tmp.length()),
                                      ped.getText(),relList);
```

```
    ped.dispose();
                  Dialog_active = false;
       }

       if(ped.is_Cancelclicked() == true)
                         {
    System.out.println("It works !!!");
    ped.dispose();
            Dialog_active = false;
                         }
                   }
                   return true;
               }

               // If the add button is depressed
               if("Add".equals(ev.arg))
               {
Frame fr = new Frame();
                 if((Active_list_box == 0) && (Dialog_active == false))
                 {
     Dialog_active = true;
                     NewProperty nchld=new NewProperty(Name,this,fr);

                     nchld.setModal(true);
                     nchld.show(true);

                     // button handlers
                     if(nchld.is_Okclicked() == true)
                     {
                          if((nchld.getText()).length() != 0)
          {
                               System.out.println("Comes here");
       AddNewProperty(Name,nchld getText());
       nchld.dispose();
       Dialog_active = false;
                          }
                          else
                          {
                               System.out.println("Invalid entry");
                          }
                     }

                     if(nchld.is_Cancelclicked() == true)
                     {
```

```
                    System.out.println("Cancel clicked");
               nchld.dispose();
     Dialog_active = false;
                 }
            }

     if((Active_list_box == 1) && (Dialog_active == false))
            {
                 Dialog_active = true;
                 NewRelation nRelsh=new NewRelation(Name,cgi,fr);
                 nRelsh.move(250,250);
                 nRelsh.setModal(true);
                 nRelsh.show(true);

                 // button handlers
                 if(nRelsh.is_Okclicked() == true)
                 {
                    if( (nRelsh.get_Selectedrelation() != null) &&
                         (nRelsh.get_Selectedconcept() != null) )
  {
     cgi.AddRelationValue(Name,
nRelsh.get_Selectedrelation(),
                                          nRelsh.get_Selectedconcept());
            if(cgi.OK == false)
     {
          MessageBox msg = new MessageBox(new
 Frame(),
                                           "Warning",
"CGI operation failed",true);
     }
     else
     {
   // include the new IS-A link
   String str = new String();
   str = nRelsh.get_Selectedrelation() + "=" +
                                          nRelsh.get_Selectedconcept();
   System.out.println(str);
   relList.addItem(str);
   selection_buffer[changed] =
                                        nRelsh.get_Selectedconcept();
                                  changed++;
          nRelsh.dispose();
     }
  }
```

```
          else
  {
        System.out.println("no items selected");
  }
                          Dialog_active = false;
                      }

                      if(nRelsh.is_Cancelclicked() == true)
                      {
                          nRelsh.dispose();
                          Dialog_active = false;
                      }
                  }
                  return true;
              }


              // If the delete button is depressed
              if("Delete".equals(ev.arg))
              {
                  Frame fr = new Frame();
                  if(Active_list_box == 0)
                  {
                      MessageBox mbox=new MessageBox(fr,"Warning",
"Do You want to Delete "
                                  +propList.getSelectedItem(),true);
                      mbox.move(250,250);
                      mbox.setModal(true);
                      mbox.show(true);

                      if(mbox.Ok_clicked() == true)
                      {
                          String tmp=propList.getSelectedItem();
                          cgi.RemoveAttributeValue(Name,
                              tmp.substring(0,tmp.indexOf('=')),
                              tmp.substring(tmp.indexOf('=')+1,tmp.length()));
                          UpdatePropertyValues(propList);
                      }
                  }
                  else
                  {
                       MessageBox mbox3=new MessageBox(fr,"Warning",
"Do You want to Delete "
                                   +relList.getSelectedItem(),true);
```

```
                    mbox3.move(250,250);
                    mbox3.setModal(true);
                    mbox3.show(true);
                    if(mbox3.Ok_clicked() == true)
                    {
                         String tmp=relList.getSelectedItem();
                         cgi.RemoveRelationValue(Name,tmp.substring(0,
tmp.indexOf('=')),
                                   tmp.substring(tmp.indexOf('=')+1,
        tmp.length()));
                         UpdatePropertyValues(relList);
                    }
              }
              return true;
            }
        }
 return false;
     }


     public String[] get_newconstraints()
     {
          return selection_buffer;
     }

     void AddNewProperty(String Parent,String pName)
     {
          cgi.AddNewProperty(Parent,pName);
          if(cgi.OK)
               propList.addItem(pName+"=");
     }
        /*
        public void actionPerformed(ActionEvent evt)
        {
             Frame fr = new Frame();
             if(evt.getSource() == (Object) propList)
     {
          String tmp=propList.getSelectedItem();

                 ped=new PropertyEditor(tmp.substring(0,tmp.indexOf('=')),
                              tmp.substring(tmp.indexOf('=')+1,
                              tmp.length()),this,propList,fr);
                 ped.setModal(true);
                 ped.move(250,250);
```

```
                    ped.show(true);
                    System.out.println("You are Editing "+
propList.getSelectedItem());
  System.out.println("It works,yipee!!");
    }

               if(evt.getSource() == (Object) relList)
               {
                    String tmp = relList.getSelectedItem();
                    ped = new PropertyEditor(tmp.substring(0,tmp.indexOf('=')),
                                    tmp.substring(tmp.indexOf('=')+1,
                                    tmp.length()),this,relList,fr);
                    ped.setModal(true);
                    ped.move(250,250);
                    ped.show(true);
                    System.out.println("You are Editing"+
relList.getSelectedItem());
                    System.out.println("It works,yipee!!");
               }
          }
            */
 };


class fAdapt extends FocusAdapter
{

      PropertyDialog myDialog;

     //class constructor method
     public fAdapt(PropertyDialog pd)
     {
          myDialog = pd;
     }

     public void focusGained(FocusEvent e)
     {
          if( (e.getID() == FocusEvent.FOCUS_GAINED) &&
                           (e.getSource() == myDialog.propList) )
          {
               myDialog.Active_list_box = 0;
               System.out.println(myDialog.Active_list_box);
```

```
            if( (e.getID() == FocusEvent.FOCUS_GAINED) &&
                        (e.getSource() == myDialog.relList) )
            {
                 myDialog.Active_list_box = 1;
                 System.out.println(myDialog.Active_list_box);
            }
      }
}


/**
*   File: SchemaCanvas.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import EDU.auburn.VGJ.gui.*;
import EDU.auburn.VGJ.graph.*;
import EDU.auburn.VGJ.algorithm.tree.*;
import EDU.auburn.VGJ.algorithm.GraphAlgorithm;
import java.util.Stack;

/**
*    Canvas for showing schema graph.
*/

public class SchemaCanvas extends GraphCanvas
{
         /**
         *  Contructor for canvas takes Schema Graph, and frame reffrence
         */
         public SchemaCanvas(OohvrSchema s,Frame Nf)
         {
              super(s,Nf);
         }

    /*     public boolean mouseDown(Event evt,int x, int y)
         {
              System.out.println("Mouse is moving over me"+x+":"+y);
```

```
            return (super.mouseDown(evt,x,y));
        } */
}


import java.awt.*;
import java.awt.event.*;
import EDU.auburn.VGJ.graph.*;
import java.util.Hashtable;

public class listframe extends Dialog
{
    Enhanced_list myList,selectList;
    Button done,cancel,Add,Remove;


    int no_of_items;
    Set selected_set;

    private  Hashtable hash;

    public listframe(String[] contents,int number,NewFrame frame)
    {
        super(frame,"List of children",true);
        resize(600,600);
        no_of_items = number;

        GridBagLayout layout = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
    setLayout(layout);


        // SETUP THE GRIDBAGLAYOUT

    c.insets = new Insets(2,5,2,5);

     // set up label 1
     c.gridx = 1;
     c.gridy = 1;
     c.gridwidth = 1;
     c.gridheight = 1;
    c.anchor = GridBagConstraints.CENTER;
    c.fill = GridBagConstraints.HORIZONTAL;
    c.weightx = 0.0;
```

```
  c.weighty = 0.0;

   Label label1 = new Label("Candidates");
        layout.setConstraints(label1,c);
        add(label1);

        // set up label 2
        c.gridx = 11;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.anchor = GridBagConstraints.CENTER;
        c.fill = GridBagConstraints.HORIZONTAL;
        c.weightx = 0.0;
        c.weighty = 0.0;

        Label label2 = new Label("Chosen");
        layout.setConstraints(label2,c);
        add(label2);

        // setup the list box
        c.gridx = 1;
        c.gridy = 2;
        c.gridwidth = 10;
        c.gridheight = 10;
        c.anchor = GridBagConstraints.CENTER;
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        c.weighty = 1.0;

        myList = new Enhanced_list(10,false);

        for(int i=0; i<number; i++)
             myList.addItem(contents[i]);// fill the list box with
                                         // the children concepts
layout.setConstraints(myList,c);
        add(myList);


        // setup ya list box
        c.gridx = 11;
        c.gridy = 2;
        c.gridwidth = 10;
        c.gridheight = 10;
```

```
        c.anchor = GridBagConstraints.CENTER;
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        c.weighty = 1.0;

        selectList = new Enhanced_list(10,false);
        layout.setConstraints(selectList,c);
        add(selectList);


        // Add the "done" button
        c.gridx = 2;
        c.gridy = 12;
        c.gridwidth = 2;
        c.gridheight = 2;
        c.anchor = GridBagConstraints.CENTER;
        c.fill = GridBagConstraints.HORIZONTAL;
        c.weightx = 0.0;
        c.weighty = 0.0;

        done = new Button("DONE");
layout.setConstraints(done,c);
        add(done);

        // Add the "Add" button
        c.gridx = 4;
        c.gridy = 12;
        c.gridwidth = 2;
        c.gridheight = 2;
        c.anchor = GridBagConstraints.CENTER;
        c.fill = GridBagConstraints.HORIZONTAL;
        c.weightx = 0.0;
        c.weighty = 0.0;

        Add = new Button("Add all");
        layout.setConstraints(Add,c);
        add(Add);

        //Add the "Remove all" button
        c.gridx = 8;
        c.gridy = 12:
        c.gridwidth = 2;
        c.gridheight =2;
        c.anchor = GridBagConstraints.CENTER;
```

```
        c.fill = GridBagConstraints.HORIZONTAL;
        c.weightx = 0.0;
        c.weighty = 0.0;

        Remove = new Button("Remove all");
        layout.setConstraints(Remove,c);
        add(Remove);


        // allocate selected set
        selected_set = new Set();

        // Initialize hash table
        hash = new Hashtable(),

}

public Set get_selected()
{
      return selected_set;
}

// class event handler function

public boolean handleEvent(Event ev)
{
        if(ev.id == Event.ACTION_EVENT)

                if("DONE".equals(ev.arg))
                {
            System.out.println("Clicked on ok");
            show(false);
                        return true;
                }

                if("Add all".equals(ev.arg))
   {
            System.out.println("Clicked on add");
            for(int i=0; i<no_of_items; i++)
            {
                    if(selectList.isPresent(myList.getItem(i)) == false)
                    {
                            selected_set.includeElement(i);
                            selectList.addItem(myList.getItem(i));
```

```
                        hash.put(myList.getItem(i),new Integer(i));
                    }
                }
         return true;
        }

        if("Remove all".equals(ev.arg))
        {
     System.out.println("Clicked on remove");
     while(selected_set.isEmpty() != true)
     {
         selected_set.removeElement(selected_set.first());
     }
     hash.clear();
     selectList.clear();
     return true;
        }
           }

   if(ev.id == Event.LIST_SELECT)
   {

        if(ev.target == (Object) myList)
        {
            if(selectList.isPresent(myList.getSelectedItem()) == false)
            {
                selectList.addItem(myList.getSelectedItem());
                selected_set.includeElement(
myList.getSelectedIndex());
                hash.put(myList.getSelectedItem(),
                              new Integer(myList.getSelectedIndex()));
                System.out.println(selected_set);
                return true;
            }
        }

        if(ev.target == (Object) selectList)
        {
            int x = selectList.getSelectedIndex();
            String str = selectList.getItem(x);
            Integer y = (Integer) hash.get(str);
            selected_set.removeElement(y.intValue());
            hash.remove(str);
            selectList.delItem(x);
```

```
            System.out.println(selected_set);
            return true;
        }
  }

          return false;
     }

}



/**
*   File: ViewFrame.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import EDU.auburn.VGJ.gui.*;
import EDU.auburn.VGJ.graph.*;
import EDU.auburn.VGJ.algorithm.tree.*;
import EDU.auburn.VGJ.algorithm.GraphAlgorithm;
import java.util.Stack;

/**
*    Main window in which schema would be shown
*/

public class ViewFrame extends Frame
{
    OohvrSchema mainSchema;
    SchemaCanvas gCnv;
    double view;
    OOhvrCGI cgi;
    Frame fr;
    PropertyDialog propDlg;
    char ViewType;
    /**
    *   Constructor takes schema graph, Schema canvas and CGI
```

```
*   and reffrences.
*/
public ViewFrame(OchvrSchema g,OOhvrCGI ocgi)
{
   super();
   ViewType='d';
   view=1;
   cgi=ocgi;
   mainSchema=g;
   setLayout(new BorderLayout());
   Panel pnl=new Panel();
   pnl.add(new Button("+"));
   pnl.add(new Button("-"));
   pnl.add(new Button("Layout Graph"));
   pnl.add(new Button("Properties"));
   pnl.add(new Button("ChangeView"));

     pnl.add(new Button("Attributes"));
   pnl.add(new Button("Relationships"));*/
   add("North",pnl);
}
public void UpdateCanvas(SchemaCanvas cnv)
{
      gCnv=cnv;
}
/**
*  Event handler for the main window.
*/
public boolean action(Event evt, Object arg) {

    if ("+".equals(arg)) { // To Increase view size
        view=view+1;
        gCnv.setScale(view);
        System.out.println("Added in view");
return true;
    }else
    if ("ChangeView".equals(arg)) { // To Increase view size
        if(ViewType=='d')
             ViewType='r';
        else
             ViewType='d';
        Refreash();
return true;
    }else
```

```
    if ("-".equals(arg)) {  // To decrease view size
        view=view-1;
        if(view<0) view=1;
        gCnv.setScale(view);
        gCnv.update(true);
        System.out.println("Subtracted in view");
return true;
    }else                          // show up the property window
                                   // for selected concept
    if("Properties".equals(arg))
    {
           //Status="Contacting Server Wait ...... ";
           //showStatus(Status);
           Node root = gCnv.getSelectedNode();
           if(root!=null)
           {
              String selected=root.getLabel();

              String prop1[]=cgi.GetProperties(selected);
              int noprop1 = cgi.ReadTotal();
              boolean attOK = cgi.OK;

              String prop2[]=cgi.GetRelations(selected);
              int noprop2 = cgi.ReadTotal();
              boolean relOK = cgi.OK;

              if(attOK == true && relOK==true)
              {
               /* fr=new Frame();
                fr.resize(50,50);
                System.out.println("Activating My Dialog");
                propDlg=new PropertyDialog(fr,this,cgi,
                        selected,prop1,prop2,noprop1,noprop2,'A');
                propDlg.show();
                propDlg.resize(310,410);*/
              }
            }
              return true;
    }
    else
    if ("Layout Graph".equals(arg)) { // Relayout the graph.
        /*double i=gCnv.SELECT_NODES;
        if(gCnv==null)
               System.out.println("Canvas Null");
```

```
            Node root = gCnv.getSelectedNode();
            GraphAlgorithm alg=new TreeAlgorithm('d');
            mainSchema.removeGroups();
            mainSchema.pack();
            String msg=alg.compute(mainSchema,gCnv);
            gCnv.update(true);
            System.out.println("Layout:"+msg); */
            Refreash();
    return true;
        }
return false;
    }
    /**
    *   Runs the layout algorithem on the graph and
    *   Updates the screen with new layout.
    */
    public void Refreash()
    {
           gCnv.setSelectedNode(mainSchema.getRootId());
           mainSchema.setDirected(false);
           GraphAlgorithm alg=new TreeAlgorithm(ViewType);
           mainSchema.removeGroups();
           mainSchema.pack();
           String msg=alg.compute(mainSchema,gCnv);
           mainSchema.setDirected(true);
           gCnv.update(true);
           System.out.println("Layout:"+msg);
    }
}



/**
*   File: NewFrame.java
*   Project Title: VOCABULARY CREATOR
*   Author: Muhammad Arif
*   Last updated on : 07/09/97
*   Last updated by : Muhammad Arif
*/

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import EDU.auburn.VGJ.gui.*;
```

```
import EDU.auburn.VGJ.graph.*;
import EDU.auburn.VGJ.algorithm.tree.*;
import EDU.auburn.VGJ.algorithm.GraphAlgorithm;
import java.util.Stack;

/**
*    Main window in which schema would be shown
*/

public class NewFrame extends Frame
{
    OohvrSchema mainSchema;
    SchemaCanvas gCnv;
    double view;
    OOhvrCGI cgi;
    Frame fr;
    PropertyDialog propDlg;
    char ViewType;
    Menu MainMenu,Options;
    MenuBar mb;
    int Expansion_number;

    /**
    *   Constructor takes schema graph, Schema canvas and CGI
    *   and reffrences
    */
    public NewFrame(OohvrSchema g,OOhvrCGI ocgi)
    {
       super();

       MainMenu = new Menu("View");
       MainMenu.add(new MenuItem("Ancestor View"));
       MainMenu.add(new MenuItem("Childern View"));
       MainMenu.add(new MenuItem("First Level Neighborhood"));
       MainMenu.add(new MenuItem("Exit"));

       Options = new Menu("Options");
       Options.add(new MenuItem("Select Detail"));

       mb=new MenuBar();
       mb.add(MainMenu);
       mb.add(Options);

       setMenuBar(mb);
```

```
    ViewType='d';
    view=1;
    cgi=ocgi;
    mainSchema=g;
    setLayout(new BorderLayout());

    Panel pnl=new Panel();
    pnl.add(new Button("+"));
    pnl.add(new Button("-"));
    pnl.add(new Button("New Child"));
    pnl.add(new Button("Layout Graph"));
    pnl.add(new Button("Properties"));
    pnl.add(new Button("Expand"));
    pnl.add(new Button("Detract"));
    pnl.add(new Button("ChangeView"));
    add("North",pnl);


    Expansion_number = 5;
}
public void UpdateCanvas(SchemaCanvas cn)
{
    gCnv=cn;
}
/**
*  Event handler for the main window.
*/
public boolean action(Event evt, Object arg)
{
    if(evt.arg.equals("Ancestor View"))
    {
            Node root = gCnv.getSelectedNode();
            System.out.println("Activating Ancestor View");
            if(root!=null)
            {
                int id=0;
                String selected=root.getLabel();
                Stack stk=new Stack();
                stk.push(selected);
                OohvrSchema os=new OohvrSchema(selected,2,"IN");
                while(!stk.empty())
                {
                    String Child=(String)stk.pop();
```

```
            String str[]=cgi.GetParent(Child);
            for(int i=0;i<cgi.ReadTotal();i++)
            {
                    id=os.AddParent(Child,str[i]);
                    stk.push(str[i]);
            }
        /*    String rel[]=cgi.GetRelations(Child);
            for(int k=0;k<cgi.ReadTotal();k+=2)
            {
              if(!rel[k].equals("SUBCLASS_OF") &&
                        !rel[k].equals("SUPERCLASS_OF"))
              {  os.AddChild(Child,rel[k+1]);
                 os.LabelEdge(Child,rel[k+1],rel[k]);
}
            }   */
          }
          os.setRootId(id);
          ViewFrame vf=new ViewFrame(os,cgi);
          vf.resize(800,700);
          SchemaCanvas scnv=new SchemaCanvas(os,vf);
          scnv.setMouseMode(scnv.SELECT_NODES);
          vf.UpdateCanvas(scnv);
          ScrolledPanel vPanel=new ScrolledPanel(scnv);
          vf.add("Center",vPanel);
          vf.pack();
          vf.show();
          vf.Refreash();
       }
       return true;
}

if(evt.arg.equals("Childern View"))
{
       Node root = gCnv.getSelectedNode();
       System.out.println("Activating Child View");
       if(root!=null)
       {
          int id;
          String selected=root.getLabel();
          Stack stk=new Stack();
          stk.push(selected);
          OohvrSchema os=new OohvrSchema(selected,2,"IN");
          while(!stk.empty())
          {
```

```
                String Parent=(String)stk.pop();
                String str[]=cgi.GetChild(Parent);
                for(int i=0;i<cgi.ReadTotal();i++)
                {
                        id=os.AddChild(str[i],Parent);
                        stk.push(str[i]);
                }
            }
            ViewFrame vf=new ViewFrame(os,cgi);
            vf.resize(800,700);
            SchemaCanvas scnv=new SchemaCanvas(os,vf);
            scnv.setMouseMode(scnv.SELECT_NODES);
            vf.UpdateCanvas(scnv);
            ScrolledPanel vPanel=new ScrolledPanel(scnv);
            vf.add("Center",vPanel);
            vf.pack();
            vf.show();
            vf.Refresh()
        }
        return true;
}

if(evt.arg.equals("First Level Neighborhood"))
{
        System.out.println("Activating Neighborhood View");
        Node root = gcnv.getSelectedNode();
        if(root!=null)
        {
            int id;
            String selected=root.getLabel();
            OohvrSchema os=new OohvrSchema(selected,2,"IN");
            String str[]=cgi.GetParent(selected);
            for(int i=0;i<cgi.ReadTotal();i++)
            {
                    id=os.AddChild(str[i],selected);
            }
            String str2[]=cgi.GetChild(selected);
            for(int i=0;i<cgi.ReadTotal();i++)
            {
                    id=os.AddChild(selected,str2[i]);
            }
            ViewFrame vf=new ViewFrame(os,cgi);
            vf.resize(800,700);
            SchemaCanvas scnv=new SchemaCanvas(os,vf);
```

```
                scnv.setMouseMode(scnv.SELECT_NODES);
                vf.UpdateCanvas(scnv);
                ScrolledPanel vPanel=new ScrolledPanel(scnv);
                vf.add("Center",vPanel);
                vf.pack();
                vf.show();
                vf.Refreash();
            }
            return true,
    }

    if(evt.arg.equals("Exit"))
    {
        dispose(),
    }

    if(evt.arg.equals("Select Detail"))
    {
      // pop up the dialog box for selecting no of children
        Info_box select_detail = new
            Info_box("Number of children during expansion",
3,this);
        select_detail.move(200,200);
        select_detail.setModal(true);
        select_detail.pack();
        select_detail.show(true);

        if(select_detail.Ok_clicked() == true)
        {
            Expansion_number =
select_detail.get_textfieldvalue();
        }
    }



     if ("+".equals(arg)) { // To Increase view size
         view=view+1;
         gCnv.setScale(view),
         System.out.println("Added in view");
   return true;
     }

     if ("ChangeView".equals(arg)) { // To Increase view size
```

```
        if(ViewType=='d')
             ViewType='r';
        else
             ViewType='d';
        Refreash();
return true;
    }

    if ("-".equals(arg)) {  // To decrease view size
        view=view-1;
        if(view<0) view=1;
        gCnv.setScale(view);
        gCnv.update(true);
        System.out.println("Subtracted in view");
return true;
    }                      // show up the property window
                           // for selected concept
    if("Properties".equals(arg))
    {
           //Status="Contacting Server Wait ...... ";
           //showStatus(Status);
           Node root = gCnv.getSelectedNode();
           if(root!=null)
           {
              String selected=root.getLabel();

              String prop1[]=cgi.GetProperties(selected);
              int noprop1 = cgi.ReadTotal();
              boolean attOK = cgi.OK;

              String prop2[]=cgi.GetRelations(selected);
              int noprop2 = cgi.ReadTotal();
              boolean relOK = cgi.OK;

              if(attOK == true && relOK==true)
              {
                System.out.println("Activating My Dialog");
                propDlg=new PropertyDialog(this,cgi,selected,prop1
                         ,prop2,noprop1,noprop2);
                propDlg.setModal(true);
                propDlg.pack();
                propDlg.resize(310,410);
                propDlg.move(200,200);
                propDlg.show(true);
```

```
    if(propDlg.is_Okclicked() == true)
    {
      // code to add new links (if any)
      //Implemented only for SUB_CLASS relationships

      String[] new_cons = new String[50];
      new_cons = propDlg.get_newconstraints();
                      for(int x=0; new_cons[x] != null; x++)
      {
  System.out.println("New entry = "+new_cons[x]);
  int id = mainSchema.get_nodeid(new_cons[x]);
  if(id != -1)
  {
     mainSchema.AddEdge(selected,new_cons[x]);

  }
  else
  {
   System.out.println("Node"+new_cons[x]+
                                  "may not be inserted correctly");
  }
    }
 Refreash();
         propDlg.dispose();
     }
                  }
                  else
                  {
                    System.out.println("Something messed up");
                  }
              }
              else
              {
                    System.out.println("Node not selected");
              }
              return true;
        }

        if("Expand".equals(arg))
        {
              Node root = gCnv.getSelectedNode();
              int no_of_children
              if(root!=null)
```

```
            {
                String selected=root.getLabel();
                String child[]=cgi.GetChild(selected);
                no_of_children = cgi.ReadTotal();

                if(no_of_children>0)
                {
                   if(no_of_children <= Expansion_number)
                   {
                     for(int i=0; i<no_of_children; i++)
                     {
                       if(mainSchema.isNodepresent(child[i]) ==
                            true)   //child already present
                       {
mainSchema.AddEdge(child[i],selected);
                       }
   else
   {
                        mainSchema.AddChild(selected,child[i]);
                           //adds children nodes


                         // ADD ADDITIONAL LINKS IF ANY...

                        String Parents[] = cgi.GetParent(child[i]);
                        int no_of_additional_links = cgi.ReadTotal();
                        if(no_of_additional_links > 1)
                        {
                         for(int j=0; j<no_of_additional_links;j++)
                          {
                            if(Parents[j].equals(selected))
                            {
                               System.out.println(
"This link already exists");
                            }
                            else
                            {
                              System.out.println(
"Link Needed");
                              if(mainSchema.isNodepresent(
Parents[j]) == true)
   {
                                 mainSchema.AddEdge(
child[i],Parents[j]);
```

```
    }
                                          } // end if
                                      } // end for
                                    }
                                    else
                                    {
                                        System.out.println("no more IS A links
+'' for"+child[i]);
                                    }  // end if
                                   }
                                  }  // end for
                                }
                                else
                                {
                                     // include code to display a list box of concepts
                                     listframe childlist = new listframe(child,
no_of_children,this);
       childlist.pack();
                                     childlist.show(true);
                                     childlist.move(200,200);
                                     Set selected_set = new Set();
                                     selected_set = childlist.get_selected();
                                     System.out.println("In NewFrame"+selected_set);
                                     if(selected_set.isEmpty() == false)
// do only if an item is selected
     {
                                          while(selected_set.isEmpty() == false)
                                          {
       int z = selected_set.first();
//               System.out.println(z);
       selected_set.removeElement(z);

                                               if(mainSchema.isNodepresent(child[z])
 == false)
                                                   // check if child already exists
       {
      mainSchema.AddChild(selected,
child[z]);  //adds children nodes

                                                      // ADD ADDITIONAL LINKS IF ANY...

                                                      String Parents[] = cgi.GetParent
(child[z]);
                                                      int no_of_additional_links =
```

```
cgi.ReadTotal();
                                        if(no_of_additional_links > 1)
                                        {
                                            for(int j=0;
j<no_of_additional_links; j++)
                                            {
                                                if(Parents[j].equals(
selected))
                                                {
                                                    System.out.println
("This link already exists");
                                                }
                                                else
                                                {
                                                    System.out.println
("Link Needed");
                                                    if(
mainSchema.isNodepresent(Parents[j]) == true)
            {

 mainSchema.AddEdge(child[z],Parents[j]);
            }
                                                } // end if
                                            } // end for
        }
}
else
{
    mainSchema.AddEdge(child[z] selected);
    System.out.println(child[z]+" already exists");
}
                        }
  System.out.println("In NewFrame after removal"+selected_set);
                    }
                } //end if no_of_children <= 5
            }
            else
            {
                System.out.println("No Children, no expansion");
            }
            Refreash();
        }
        else
        {
```

```
                System.out.println("Node not selected");
            }
            return true;
        }


//                      Refreash();
                }
                else
                {
                    System.out.println("Node not selected");
                }
                return true;
        }

        if ("New Child".equals(arg)) {  // Add a new child to a
                                        // selected concept
                Node root = gCnv.getSelectedNode();
                if(root!=null)
                {
                    String parent=root.getLabel();
                    NewChild nch=new NewChild(parent,this);
                }
                else
                {
                    System.out.println("Node not selected");
                }
      return true;
        }

        if ("Layout Graph".equals(arg)) // Relayout the graph.
        {
/*              double i=gCnv.SELECT_NODES;
                if(gCnv==null)
                        System.out.println("Canvas Null");
                Node root = gCnv.getSelectedNode();
                GraphAlgorithm alg=new TreeAlgorithm('d');
                mainSchema.removeGroups();
                mainSchema.pack();
                String msg=alg.compute(mainSchema,gCnv);
                gCnv.update(true);
                System.out.println("Layout:"+msg);
 */
```

```
            Refreash();

    return true;
        }

        if(evt.id == Event.WINDOW_ICONIFY)
        {
             System.out.println("comes here, great!");
        }
return false;

    }
    /**
    *   Runs the layout algorithm on the graph and
    *   Updates the screen with new layout.
    */
    public void Refreash()
    {
           gCnv.setSelectedNode(mainSchema.getRootId());
           mainSchema.setDirected(false);
           GraphAlgorithm alg=new TreeAlgorithm(ViewType);
//           mainSchema.removeGroups();
//           mainSchema.pack();
           String msg=alg.compute(mainSchema,gCnv);
           mainSchema.setDirected(true);
           gCnv.update(true);
           System.out.println("Layout:"+msg);
    }
    /**
    *    Add a new child to the concept by calling the API
    *    from CGI class
    */
    public void AddNewChild(String Parent,String Child)
    {
           cgi.AddNewChild(Parent,Child);
           if(cgi.OK)
           {
            mainSchema.AddChild(Parent,Child);
            Refreash();
           }
    }
}
```

APPENDIX C

API DESCRIPTION

The defination of APIs is presented here.

```
/*********************************************************************
 *
 * Funciton MV_List_Children
 * Input: [ ConceptName ] Concept name.
 * Output:[ number ] Number of children
 *        [ list ] keeps all children name
 * Description:
 * Function caller need to release the memory.
 * delete list [number][];
 *
 *********************************************************************/

void MV_List_Children( char* ConceptName,int& number, char ** &list);

/*********************************************************************
 *
 *      Funciton MV_List_Parents
 *      Input: [ ConceptName ] Concept name.
 *      Output:[ number ] Number of children
 *             [ list ] keeps all parent name
 *      Description:
 *              Function caller need to release the memory.
 *              delete list [number][];
 *
 *********************************************************************/

void MV_List_Parents( char* ConceptName,int& number, char ** &list);

/*********************************************************************
 *
 *      Funciton MV_Show_Attribute_Value
 *      Input: [ ConceptName ] Concept name.
 *        [ AttributeName ] Attribute name.
 *      Output:[ number ] Number of value of this attribute
 *             [ list ] keeps all value of this attribute
 *      Description:
 *              Function caller need to release the memory.
```

```
 *              delete list [number][];
 *
 ****************************************************************/

void MV_Show_Attribute_Value( char* ConceptName,
char* AttributeName,
int& number,char** &Value);

/****************************************************************
 *
 *      Funciton MV_Show_Relationship_Value
 *      Input: [ ConceptName ] Concept name.
 *             [ AttributeName ] Relationship name.
 *      Output:[ number ] Number of value of this Relationship
 *             [ list ] keeps all concept name of this Relationship
 *      Description:
 *              Function caller need to release the memory.
 *              delete list [number][];
 *
 ****************************************************************/

void MV_Show_Relationship_Value(char* ConceptName,
char* RelationshipName,
int& number,char** &Value);

/****************************************************************
 *
 *      Funciton MV_List_All_Property_Value
 *  ( MV_List_All_Attribute_Value )
 *  ( MV_List_All_Relationship_Value )
 *      Input: [ ConceptName ] Concept name.
 *      Output:[ number ] keeps the number of list
 *             [ list ] store property name and value
 * Example: list[1] = "SUBCLASS_OF" <- property name
 *  list[2] = "ENTITY" <- value
 *  list[3] = "NAME" <- property name
 *  list[4] = "Procedure" <- value
 *      :
 *      Description:
 *              Function caller need to release the memory.
 *              delete list [number][];
 *
 ****************************************************************/
```

```
void MV_List_All_Property_Value(char* ConceptName,
int& number, char ** &list);
void MV_List_All_Attribute_Value(char* ConceptName,
int& number, char ** &list);
void MV_List_All_Relationship_Value(char* ConceptName,
int& number, char ** &list);

/******************************************************************
 *
 *      Funciton MV_Change_Attribute_Value
 *  Input: [ ConceptName ] Concept name.
 *         [ AttributeName ] Attribute name.
 *         [ Value ] New value.
 * Output: 0 if no such concept exist or
 *      no such attribute exist in this concept
 * 1 normal update
 * Description:
 *              If no such attribute exist in this certain concept,
 * function will return without any change.
 * If the front end AP needs some error codes, I need to
 * rewrite this function
 *
 ******************************************************************/

int MV_Change_Attribute_Value( char* ConceptName,
char* AttributeName,
char* oldValue,
char* Value);

/******************************************************************
 *
 *      Funciton MV_Change_Relationship_Value
 *      Input: [ ConceptName ] Concept name.
 *             [ RelationshipName ] Relationship name.
 *             [ CName ] New target concept name.
 *      Output: 0 if no such concept exist or
 *                   no such attribute exist in this concept
 *             1 normal update
 *      Description:
 *             If no such attribute exist in this certain concept,
 *             function will return without any change.
 *             If the front end AP needs some error codes, I need to
 *             rewrite this function.
 *
```

```
*****************************************************************/

int MV_Change_Relationship_Value(  char* ConceptName,
                                  char* AttributeName,
char* OName,
                                  char* CName);

/*****************************************************************
 *
 *      Funciton MV_Show_Number_Of_Children
 *      Input: [ ConceptName ] Concept name.
 *      Output:[ number ] Number of children.
 * Descriptioin:
 * This function takes a ConceptName and return the number
 * of its children.
 *
 *****************************************************************/

void MV_Show_Number_Of_Children(char* ConceptName,int& number);

/*****************************************************************
 *
 *      Funciton MV_Show_Number_Of_Parent
 *      Input: [ ConceptName ] Concept name.
 *      Output:[ number ] Number of parents.
 *      Descriptioin.
 *              This function takes a ConceptName and return the number
 *              of its parents
 *
 *****************************************************************/

void MV_Show_Number_Of_Parents(char* ConceptName,int& number);

/*****************************************************************
 *
 *      Funciton MV_Does_Property_Exist
 *      Input: [ PropertyName ] Property Name.
 *      Output:[ IntroNode ] if we have this property, this IntroNode
 * keeps the node first introduced this property.
 * Descriptioin:
 *              This function return 1 if we do have this Property and
 * setup the value of IntroNode. Return 0 if we don't have
 * this property and IntroNode point to NULL
 *
```

```
**************************************************************/

int MV_Does_Property_Exist(char* PropertyName,char* &IntroNode );

/*************************************************************
 *
 *      Funciton MV_Create_Attribute
 *      Input: [ ConceptName ] Concept Name
 *             [ AttributeName ] Attribute Name
 *      Output: None
 *      Descriptioin:
 *              This function will add Attribute in Concept and
 * PROPAGATE this attribute to all the descendents of
 * Concept with NULL value.  If some descendents of
 * Concept already have this Attribute, this function will
 * not add anything to it
 *
 **************************************************************/

void MV_Create_Attribute(char* ConceptName,char* AttributeName);

/*************************************************************
 *
 *      Funciton MV_Delete_Attribute
 *      Input: [ ConceptName ] Concept Name
 *             [ AttributeName ] Attribute Name
 *      Output: None
 *      Descriptioin:
 *              This function will delete Attribute in Concept and
 *              recursively delete all this attribute from its
 * descendents.  IF THE NODE IS NOT THE PLACE WHICH
 * INTRODUCES THIS ATTRIBUTE, THE FUNCTION WILL RETURN
 * AND DO NOTHING.
 *
 **************************************************************/

void MV_Delete_Attribute(char* ConceptName,char* AttributeName);

/*************************************************************
 *
 *      Funciton MV_Create_Relationship
 *      Input: [ Rel1 ] Relationship Name
 *             [ Rel2 ] Reverse Relationship Name
 *        [CName1] Concept Name which introduce Rel1
```

```
*        [CName2] Concept Name which introduce Rel2
*      Output: None
*      Descriptioin:
*              This function will add relationships in Concepts and
*              PROPAGATE these relationships to all the descendents of
*              Concept with NULL value.  If some descendents of
*              Concept already have the Relationship, this function will
*              not add anything to it.
*
******************************************************************/

void MV_Create_Relationship(char* CName1,char* Rel1,
                            char* CName2,char* Rel2);

/*****************************************************************
 *
 *      Funciton MV_Delete_Relationship
 *      Input: [ Rel1 ] Relationship Name
 *             [ Rel2 ] Reverse Relationship Name
 *             [CName1] Concept Name which introduce Rel1
 *             [CName2] Concept Name which introduce Rel2
 *      Output: None
 *      Descriptioin:
 *              This function will remove relationships in Concepts and
 *              PROPAGATE this deletion to all the descendents of
 *              CName1 and CName2.  If some descendents of CName1 and
 * CName2 don't have those Relationships, this function will
 * just return and do nothing.
 *
 *****************************************************************/

void MV_Delete_Relationship(char* CName1,char* Rel1,
                            char* CName2,char* Rel2);

/*****************************************************************
 *
 *      Funciton MV_Find_Lowest_Common_Node
 *      Input: [ CName1 ] first concept name
 *        [ CName2 ] second concept name
 *      Output:[ LCNode ] lowest common node of two input nodes.
 *      Descriptioin:
 *              If CName1 is ancestor of CName2, then LCNode will be
 * set up as CName1, vice versa.
 * Currently we don't have a perfect algorithm to judge
```

```
 * which node is LCN.  So, I use greedy algorithm from
 * CName1.  In other words, if you exchange the order
 * of CName1 and CName2, you may get different answer.
 *
 *******************************************************************/

void MV_Find_Lowest_Common_Node(char* CName1,char* CName2, char* & LCNode);

/******************************************************************
 *
 *      Funciton MV_Create_Concept
 *      Input: [ ConceptName ] concept name
 *              [ ParentName ] parent name
 *      Output:none
 *      Descriptioin
 *              This function create a new concept with same property
 * as its parent has  The value of its properties will be
 * null except NAME and SUBCLASS_OF
 *
 *******************************************************************/

void MV_Create_Concept(char* ConceptName,char* ParentName);

/******************************************************************
 *
 *      Funciton MV_List_All_Concept
 *      Input: Nothing
 * Output: All concepts in the database
 *      Descriptioin.
 *
 *******************************************************************/

void MV_List_All_Concept(int& number, char ** &list);

/******************************************************************
 *
 *      Funciton MV Add_Attribute_Value
 *  Input:  [ ConceptName ] concept name
 * [ AttributeName ] attribute name
 * [ Value ] a value
 *      Output: 0 if no such concept exist or
 *                  no such attribute exist in this concept
 *              1 normal update
 * Descriptioin:
```

```
* Add one attribute-value pair into this certain concept.
* If this concept has a pair of this attribute with
* value NULL, then this pair will be changed to
* attribute-value with Value.
*
*********************************************************************/

int MV_Add_Attribute_Value( char* ConceptName,
                            char* AttributeName,
                                  char* Value);

/********************************************************************
 *
 *      Funciton MV_Add_Relationship_Value
 *      Input:  [ ConceptName ] concept name
 *              [ RelationshipName ] relationship name
 *              [ TargetConcept ] target concept
 *      Output: 0 if no such concept exist or
 *                 no such relationship exist in this concept
 *              1 normal update
 *      Descriptioin:
 *              Add one relationship-value pair into this certain concept.
 *              If this concept has a pair of this relationship with
 *              value pointer to NULL, then this pair will be changed
 * to relationship-value point to target concept.
 *
 ********************************************************************/

int MV_Add_Relationship_Value(  char* ConceptName,
                                char* RelationshipName,
                                char* TargetConcept);

/********************************************************************
 *
 *      Funciton MV_Remove_Attribute_Value
 *      Input:  [ ConceptName ] concept name
 *              [ AttributeName ] attribute name
 *              [ Value ] a value
 *      Output: 0 if no such concept exist or
 *                 no such attribute exist in this concept
 *              1 normal update
 *      Descriptioin:
 *              Remove one attribute-value pair from this certain concept.
 * If the pair is the last pair of this attribute, this
```

```
* function will keep this pair and clean the value to NULL.
*
**********************************************************************/

int MV_Remove_Attribute_Value(  char* ConceptName,
                                char* AttributeName,
                                char* Value);

/**********************************************************************
 *
 *      Funciton MV_Remove_Relationship_Value
 *      Input:   [ ConceptName ] concept name
 *               [ RelationshipName ] relationship name
 *               [ TargetConcept ] target concept
 *      Output: 0 if no such concept exist or
 *                   no such relationship exist in this concept
 *               1 normal update
 *      Descriptioin:
 *               Remove one relationship-target pair from this certain concept.
 *               If the pair is the last pair of this relationship, this
 *               function will keep this pair and clean the value to NULL.
 *
 **********************************************************************/

int MV_Remove_Relationship_Value( char* ConceptName,
                                  char* RelationshipName,
                                  char* TargetConcept);

/**********************************************************************
 *
 *      Funciton MV_Show_Reverse_Relationship
 *      Input:   [ RelationshipName ] relationship name
 * [ RevRelationshipName ] a pointer to carry the reverse
 * relationship name
 * Output: 0 if no such relationship exist or
 *               1 normal query
 *      Descriptioin:
 *               This function will return the reverse relationship name
 * to the caller.  The caller has the responsibility to clean
 * the momory of RevRelationshipName.
 *
 **********************************************************************/

int MV_Show_Reverse_Relationship( char* RelationshipName,
```

```
                                               char* & RevRelationshipName);

int MV_List_Descendant(char* ConceptName,int& number, char ** &list);


//
//  MV_Delete_Concept still need to be optimized
//  If we delete concept A, all decendants of A will be deleted, too.
//  If A or its decendants introduced any relationship, we will abort this
// operation.
//  Problem:  if concept B uses relationship R to point to one of A's decendants
// what should we do?  ( R is introduced above of concept A. )
//

int MV_Delete_Concept(char* ConceptName);

int MV_List_All_Relationship_In_One_Node(char* ConceptName,
                                         int& number,
                                         char ** &list);
```

REFERENCES

1. Goldfarb C. *SGML Handbook*. New York: Oxford University Press, 1990.

2. J. J. Cimino. Personal communication. Associate Professor Medicine, Medical Informatics, Columbia University, 1996.

3. J. J. Cimino, P. D. Clayton, G. Hripcsak, and S. B. Johnson. Knowledge-based approaches to the maintenance of a large controlled medical terminology. *JAMIA*, 1(1):35–50, 1994.

4. J. J. Cimino, G. Hripcsak, S. B. Johnson, and P. D. Clayton. Designing an introspective, multipurpose, controlled medical vocabulary. In *Proc. Thirteenth Annual Symposium on Computer Applications in Medical Care*, pages 513–517, Washington, DC, November 1989.

5. William M. Detmer and Edward H. Shortliffe. A model of clinical query management that supports integration of biomedical information over the world wide web. Section on Medical Informatics, Stanford University School of Medicine, 1994.

6. D. H. Fischer. Consistency rules and triggers for multilingual terminology. In *Proc. TKE'93, Terminology and Knowledge Engineering*, pages 333–342, 1993.

7. H. Gu, J. Cimino, M. Halper, J. Geller, and Y. Perl. Utilizing OODB schema modeling for vocabulary management. In J. Cimino, editor, *Proc. 1996 AMIA Annual Fall Symposium*, pages 274–278, Washington, DC, October 1996.

8. Michael Kifer, Won Kim, and Yehoshua Sagiv. Querying object-oriented databases. In *Proc. 1992 ACM SIGMOD Conference on Management of Data*, San Diego, CA, June 1992.

9. L. Liu, M. Halper, H. Gu, J. Geller, and Y. Perl. Modeling a vocabulary in an object-oriented database. In K. Barker and M. T. Özsu, editors, *CIKM-96, Proc. 5th Int'l Conference on Information and Knowledge Management*, pages 179–188, Rockville, MD, November 1996.

10. L. Liu, M. Halper, H. Gu, J. Geller, and Y. Perl. Controlled vocabularies in OODBs: Modeling issues and implementation. In preparation, 1997.

11. Y. Perl and J. Geller. Using object-oriented databases to make medical vocabularies comprehensible. *NJIT Research*, 5, 1997. To appear.

12. Y. Perl, J. Geller, and H. Gu. Identifying a forest hierarchy in an OODB specialization hierarchy satisfying disciplined modeling. In *Proc. First IFCIS Int'l Conference on Cooperative Information Systems (CoopIS96)*, pages 182–195, Brussels, Belgium, 1996.

13. Berners-Lee T, Cailliau R, Luotonen A, Frystyk Nielsen H, and Secret A. The world-wide web. *Communications of the ACM*, pages 37(8)–76, 1994.

14. Stanley B. Zdonik and David Maier, editors. *Readings in Object-Oriented Database Systems*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.