ABSTRACT

COMPUTER AND INFORMATION SCIENCE

BARNWELL, KEITH W. A. B.S. MOREHOUSE COLLEGE, 1991

DEVELOPING A HYPER MEDIA INTERFACE AS A NAVIGATIONAL TOOL FOR AN OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM

Advisor: Dr. Roy George

Thesis dated May, 1995

A common difficulty associated with any large scale information base is traversing the repository in a coherent and purposeful manner. The scope and diversity of the media therein tends to be more of a distraction rather than a source of information. This phenomena is particularly relevant in current hypertext or hypermedia systems and is often referred to as becoming "*lost in hyperspace*".

A solution to this hyperspace problem involves modeling the hyper system after the structured links associated with a database schema. The database schema inherently defines the formations necessary for the two basic forms of navigation in a hyperbase - *structural* and *associative*.

In order to depict the semantic relationships between nodes and links in a hyper system a model that is both readable and logic-oriented is necessary. Such a model can be expressed through conceptual graph notation. This paper documents the design of a hyper-media interface as a navigational tool for an object oriented database management system called O2. The navigational paths through the database are represented as a conceptual graph model.

DEVELOPING A HYPER MEDIA INTERFACE AS A NAVIGATIONAL TOOL FOR AN OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM

A THESIS

SUBMITTED TO THE FACULTY OF CLARK ATLANTA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF COMPUTER SCIENCE

BY

KEITH BARNWELL

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

ATLANTA, GEORGIA

MAY, 1995

V vi.T.67

ACKNOWLEDGEMENTS

Special thanks to my advisor, Dr. George, members of my thesis defense committee, Dr. Srikanth, and Dr. Perry, and to the faculty of the Computer and Information Science Department.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF TABLES v

LIST OF ILLUSTRATIONS vi

INTRODUCTION 1

PART 1. OVERVIEW

Chapter

I. HYPER SYSTEMS 5

Limitations of Current Hyper Systems 10

II. THE OODB DATABASE MODEL SOLUTION 13

Limitations of the OODB Model 16

III. THE CONCEPTUAL GRAPH AS A MEANS OF REPRESENTATION 17

PART 2. CURRENT IMPLEMENTATION

IV. *SAU* OODB MODEL 24

V. *SAU* HYPER MODEL 28

VI. *SAU* CONCEPTUAL GRAPH MODEL 32

VII. DEMONSTRATION OF *SAU* 34

PART 3. FUTURE IMPLEMENTATION

VIII. CONCLUSION 44

Future Work 46

Appendix

1. *SAU* OODB SCHEMA DEFINITION 48

2. SAMPLE *SAU* APPLICATION CODE 51

Method bodies of the class University 53
Method bodies of the class School 59
Sample Hyper Access Code 62

REFERENCES 66

Appendix

1. *SAU* OODB SCHEMA DEFINITION 48

2. SAMPLE *SAU* APPLICATION CODE 51

Method bodies of the class University 53
Method bodies of the class School 59
Sample Hyper Access Code 62

REFERENCES 66

LIST OF TABLES

Table Page

1. Table of Attributes 27

LIST OF ILLUSTRATIONS

Figure Page

1. A Simple Hypermodel 6
2. An Example Power Index Expression 8
3. Example of a Generalized Association 15
4. Three Conceptual Graph Forms 19
5. Conceptual Graph Showing a Generic Class 23
6. *SAU* OODB Model 26
7. *SAU* Pilot Presentation 35
8. An Editable Tuple Display 36
9. A Selectable List of the Universities 37
10. Iconic Representation of a University 38
11. *SAU* Hyper Access Presentation 40
12. *SAU* Hyper Presentation Showing First Level 41
13. *SAU* Hyper Presentation Showing Second Level 42

INTRODUCTION

A multimedia database presents an interesting challenge to the novice user who wishes to quickly peruse its contents. This problem is compounded by the inclusion of unformatted media (Tanaka and Qiang 1992) such as text, video, images and sound. This type of media is difficult to retrieve with an ad hoc querying facility unless some appropriate viewer is used to look at the data. Even if the database is designed to model complex real world situations as in an object oriented system, the interface is often as complicated as the system itself. Consider for example an information base filled with an encyclopedic knowledge on diverse topics. This library can present an interesting challenge to a user who wishes to compare and contrast segments of information from different subject areas. Although, this is a realistic request, the user is responsible for examining each subject area separately, and then identifying common areas of interest. This task is not only time consuming, but its success greatly depends on the analytical powers of the user. Some users may find it difficult to identify common trends simply because they encountered too many distracting bits of information.

However, if the system provided a mechanism where the user is presented with information units that are related to each other, then it would be easier to complete the aforementioned task. The user is now able to selectively identify viable access paths through the enterprise in order to view common trends in the data. In the scenario described previously, given a subject area, the user can select information of interest and follow predefined links that may span different topics. This desirable feature led to the development of *hypertext* and *hypermedia* databases. Hypertext traditionally refers to text based information repositories, whereas hypermedia encompasses text as well as other media (Bruza and van der Weide 1990). Both of these systems involve representing the enterprise as a collection of information units and providing a means of traversal via associations.

In order to emphasize the difference between these types of systems and traditional databases, the term *hyper* will be used as a separate or conjunctive adjective when referring to processes that support hypertext or hypermedia concepts. Generally, the purpose of hyper systems is to allow users to selectively identify and view units of information (Faloustos et al. 1990). This flexibility affords a more analytical view of the data in the enterprise by giving the user the power to interpret information in various ways. However, given the flexibility and scope of a hyper system, a common phenomena occurs when a user encounters several possible information alternatives and inadvertently selects a path that does not enhance the current information context. In other words, the user is sidetracked from his original query, and finds it almost impossible to relocate his original frame of reference. The broader the scope of information presented, the easier it is to become overwhelmed or sidetracked. This resulting disorientation is referred to as becoming "lost in hyperspace" (Bruza and van der Weide 1990).

Attempts to solve this hyperspace problem range from improving the architecture of the hyper model (Bruza and van der Weide 1990) to using the schematics of an existing database model as a means of providing some structure to the hyper system (Wang and Hitchcock 1991). The first approach addresses the flaws of the hyper model and provide internal solutions to the problem. This strategy involves several refinements of the hyper model itself. The latter technique of utilizing a traditional database model enhances the structure and generality of the hyper model with database features such as type hierarchy and querying facilities.

In spite of the relative successes of both of these methodologies, the semantic nature of the hyper model still require interpretation. Neither hyper graph notations or database schemas can fully express the meaning of the associations between different categories of information units. A representation that is both readable and logic-oriented is essential to effectively communicate the intricacies of the hyper network. Such a representation can be expressed through conceptual graphs. Conceptual graphs are used to depict relationships between diverse and abstract categories of information or concepts (Sowa 1993). In addition, the semantic nature of the resulting network can be illustrated.

This paper explores the current issues involved in the development of a hyper system and discusses the utilization of conceptual graphs as a means of providing adequate representation of the hyper network. The fact that conceptual graphs are important as a source of documentation for the proper maintenance of the hyper system is also discussed. In addition, an implementation of a hyper system using an existing database model, namely the object oriented database model, is presented, and a conceptual graph is generated as a formal notation for the system. The system prototype, called *SAU*, resides on a SUN workstation and features a showcase of the Atlanta area universities. *SAU* provides a graphical hyper interface to a multimedia database implemented using an object oriented database management system called O2, Version 4.4.5.

Finally, comparisons are made between *SAU*'s hyper interface and the traditional object oriented database, and the benefits of the conceptual graph as a means of maintaining the database is discussed. Some future enhancements to the *SAU* prototype is also presented.

CHAPTER I

HYPER SYSTEMS

The primary purpose of a hyper system, whether text based or other, is to provide a single flexible, and coherent interface (Conklin 1987) to complex large scale information repositories. Essentially, a hyper system involves packaging information into discrete categories called *nodes* and providing a means of traversal from node to node in a flexible and intuitive manner via associations called *links*. Node content may be portions of text, sound, graphics or video images, or combinations of all of the above (Tompa 89), and links are created based on the author's interpretation of the associations between various information units. The resulting semantic network is sometimes referred to as a *hyperdocument* or *hyperlibrary* (Schatz 89) and is a representation of the information content as a meaningful whole.

The simplest approach to the architecture of a hyper system is to use an implicit labelled directed graph (Tompa 89). This model consists of a finite set of nodes or information units, a set of labels, and a set of directed labelled edges or links (see Figure 1). In this model, there are no explicit provisions for groups or sets of nodes and links, therefore a series of iterations may be required to display generic information. Other common architectures utilize first-order logic and hypergraphs (Tompa 89).

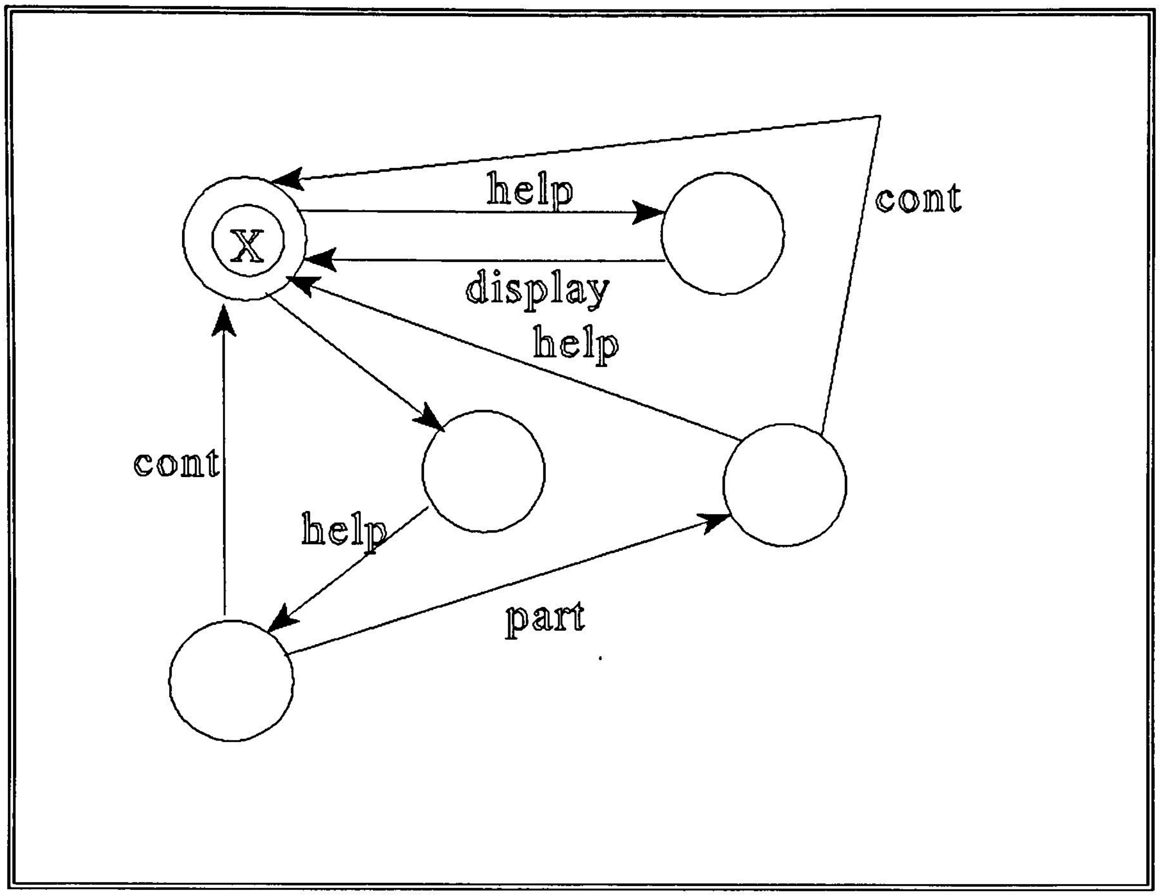


Figure 1. A simple hypermodel as a labelled directed graph. The current node is marked by an X. Adapted from Tompa 89.

The hypergraph data model emphasizes the difference in structure and content (Tompa 89). In this model, the database state includes additional concepts such as pages of data objects, readers for viewing data content, and a mapping function to correlate nodes with data content. Pages, like labels, are atomic units, and readers are used to interpret and view the contents of a page. This model allows the creation of links between collections of nodes, and views that correspond to virtual hypergraphs.

As an expansion to the hypergraph concept of separating structure from content, some researchers (Bruza and van der Weide 1990) (Tanaka and Qiang 1992) introduce the notion of a two level hypermedia architecture. Bruza separates the levels by referring to the top layer as a *hyperindex* and the bottom layer is the *hyperbase*. The hyperindex provides an abstract view of the underlying hyperbase. Nodes in the hyperindex correspond to sets of information nodes in the hyperbase. Users can traverse through the index, and expand or refine the current context by selecting specific information corresponding to an index node or retreating from the hyperbase to the hyperindex. The hyperindex also facilitates the notion of index expressions which consist of the set of all selected entries. General index expressions can be used to generate sub-expressions which eventually translate to the most specific expression corresponding to a node in the hyperbase. The resulting lattice of index expressions is called a *power index expression* (Bruza and van der Weide 1990, 82). See Figure 2 for an example.

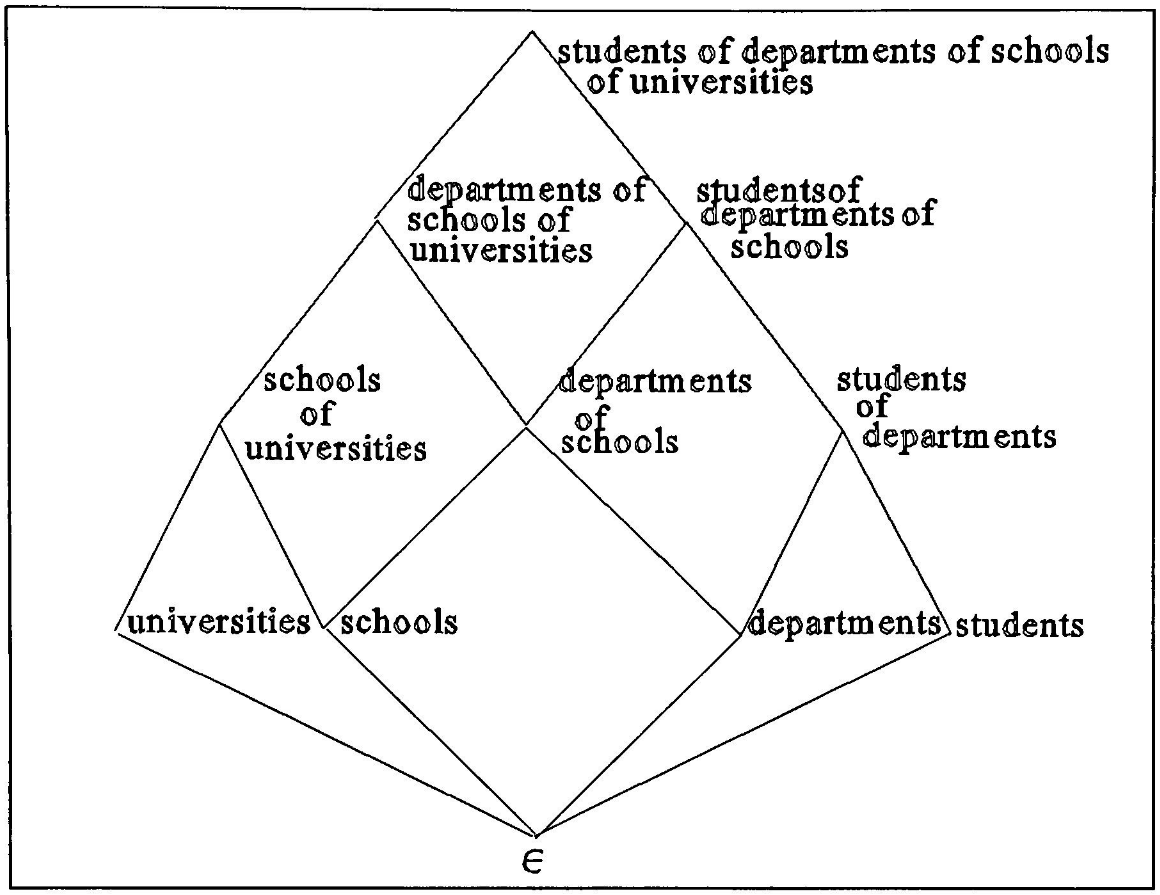

students of departments of schools of universities
departments of schools of universities
studentsof departments of schools
schools of universities
departments of schools
students of departments
universities
schools
departments
students
ϵ

Figure 2. An example power index expression. ϵ represents the empty index expression.

Another two-leveled approach to the hyperspace problem is to provide a type and view schema architecture (Tanaka and Qiang 1992). The type schema corresponds to the attribute and logical structure of objects, whereas the view schema represents conceptual objects which are virtual groupings of the physical objects themselves. An implementation of this architecture may utilize an underlying OODBMS to generate the type schema, since its objects and associations correspond directly to the class hierarchy. The view schema can then be generated from the abstract groupings (the classes in the OODB) or portions of object instances in the type schema. The resulting hyper model is therefore dependent on the underlying architecture of the existing database model, namely the OODB. This strategy of using a database model as a framework for the hyperbase is demonstrated in the *SAU* prototype, and will be discussed further in Chapter II.

Other concepts that are introduced to the hypermedia architecture as a result of utilizing the OODB model include structured types and annotated versions of links and nodes (Wang and Hitchcock 1991). All of these functions are based on object oriented precepts. A node type is viewed as an abstract data type which describes structure, and may have attributes as well as content. Node types fall into the two general categories of *primitive* or base types and *structured* types that are created from combinations of primitive types. Link types can be *annotational* or *referential.* Annotative links allow the creation of annotations within the contents of a single node, and referential links create associations between the contents of several nodes.

There are several ways that users can traverse the network of nodes and links formulated from a hyper architecture. The two basic forms of navigation are categorized as *structural* or *associative* (Bruza and van der Weide 1990). Structural navigation allows the traversal from node to node based on an underlying hierarchy. If, as is typical in hypertext systems, the hyperbase is modeled after a document structure, structural navigation follows the predefined links between chapters or section and, according to Bruza, either "enlarges or refines the current context". For the hyper model that is generated from an underlying database, structural navigation entails following the natural

IS-A or IS-PART-OF hierarchies that are inherent to the data model.

Associative navigation, on the other hand, allows a "change of context" via cross referential links between related material. These links may be formulated based on footnote or other references in a hypertext system, for example, or some other common trend identified by the author of the hyperdocument. Referential links may also be generated dynamically by search mechanisms or a querying engine.

One other form of navigation introduced with the two level hypermedia architecture is *inter-layer* navigation (Bruza and ver Weide 1990). This allows users to move between the indexed and expanded layers of the hypermedia network. This type of traversal is not to be confused with an expansion or refinement of the current context. Whereas, context enlargement or refinement occurs within the hyperindex itself, inter-layer navigation occurs between the top layer and the hyperbase, and expands or refines the database view. A third form of navigation related to traversing the hyper index is *Query By Navigation* (Bruza and van der Weide 1990). This entails the user traversing or browsing the hyperindex until a node of interest is identified. This type of navigation is facilitated by the power index expression lattice structure described earlier.

It should be noted that the two leveled architectures offer the most flexibility and diversity for inter-node traversal or navigation through the hyper document. Users can not only change, but can expand or refine the current context. In addition, the closure offered by such a system allows room for optimization.

Limitations of Current Hyper Systems

The fact that information is scattered through an interconnected web of nodes and links introduces the possibility of a user becoming entangled in the network. As the number of nodes and links increases, the number of alternate paths to any specific goal also increase. Although this increases the flexibility of the network, the opportunity to backtrack safely after following a sidetrack diminishes. Sidetracks usually occur when a user has to guess at the interpretation of the associations between information nodes. If the user guesses incorrectly then an inadvertent change of context occurs. In other words the user is now responsible for creating, and managing new links, and choosing existing links. The resulting disorientation that may occur is a common problem of current hyper systems.

There are several solutions to address this hyperspace problem. The usual approach is to provide a *browser* or a *book-mark* to facilitate orientation in the network (Bruza and van der Weide 1990). Browsers maintain the current position in the hypermedia map, while book-marks are place holders in the map. Other approaches attempt to improve the architecture of the hyper system itself with two-layered mechanisms discussed previously. The top layer can be viewed as an index to the underlying layer, and users can retreat to the index as a means of recovering or expanding the current context.

Another related problem of hyper systems has to do with the fact that most of these systems rely only on a browsing mechanism and limited textual search capability to locate information elements (Gallagher et al. 1990). This usually results in the user being bombarded by irrelevant information and further contributes to disorientation. The systems that employ a database schema address this problem by providing more powerful search mechanisms via a querying engine (Schatz 1989).

One other major drawback of current hypertext systems is their unsuitability for environments other than which they were specifically designed (190). Most hyperdocuments are modeled using a bottom-up approach simply because the expansiveness of the information base makes the task of composing material into separate nodes and providing associations between them very complicated. Also, the lack of a structure for defining types of nodes and links allows the construction of meaningless links and associations. Again, the systems that employ an inherent database model to construct the hyperbase allow the creation of types for nodes and links. In addition, the object oriented database model allows the creation of versions and annotations of nodes and links.

CHAPTER II

THE OODB DATABASE MODEL SOLUTION

The benefits of the database model solution to address the problems associated with disorientation in a hyper network is apparent. The fact that a traditional database model is used to organize the data store, and a hyper interface is used for data manipulation offers a distinct advantage. The hyper network benefits from the structured schematics offered by the underlying database, and the traditional text based or browsing mechanisms used to retrieve information elements can be enhanced by the querying facilities offered by the database.

This chapter examines in detail the benefits and limitations of using an object oriented database management system to generate a data model for a hyper system. Mapping the features of the OODB to the hyper model requires an examination of the OODB model itself. The primitives of an OODB model are *classes* of *objects*, their means of manipulation as defined in *methods*, and the generalized *associations* between them (Rumbaugh et al. 1991). These associations are usually in the form of IS-PART-OF and IS-A relationships. Objects correspond to abstract data entities, and are categorized by common elements of structure and behavior, into classes. Each object is an instance of a particular class. Classes embody the structure of an object by providing an abstract data type definition, and dictate the behavior or set of operations for manipulating objects via methods. In addition, the visibility of objects, methods and attributes of the objects themselves are also specified.

The associations formed in the OODB are classified via embedded classes, inheritance relationships, and between classes and objects themselves. An embedded class is defined as an attribute of its parent class. An inherited class may possess some or all of the qualities of its parent class, and each object instance belongs to a particular class definition. Embedded and inherited classes form IS-PART-OF associations and objects are related to classes via IS-A generalizations (see Figure 3).

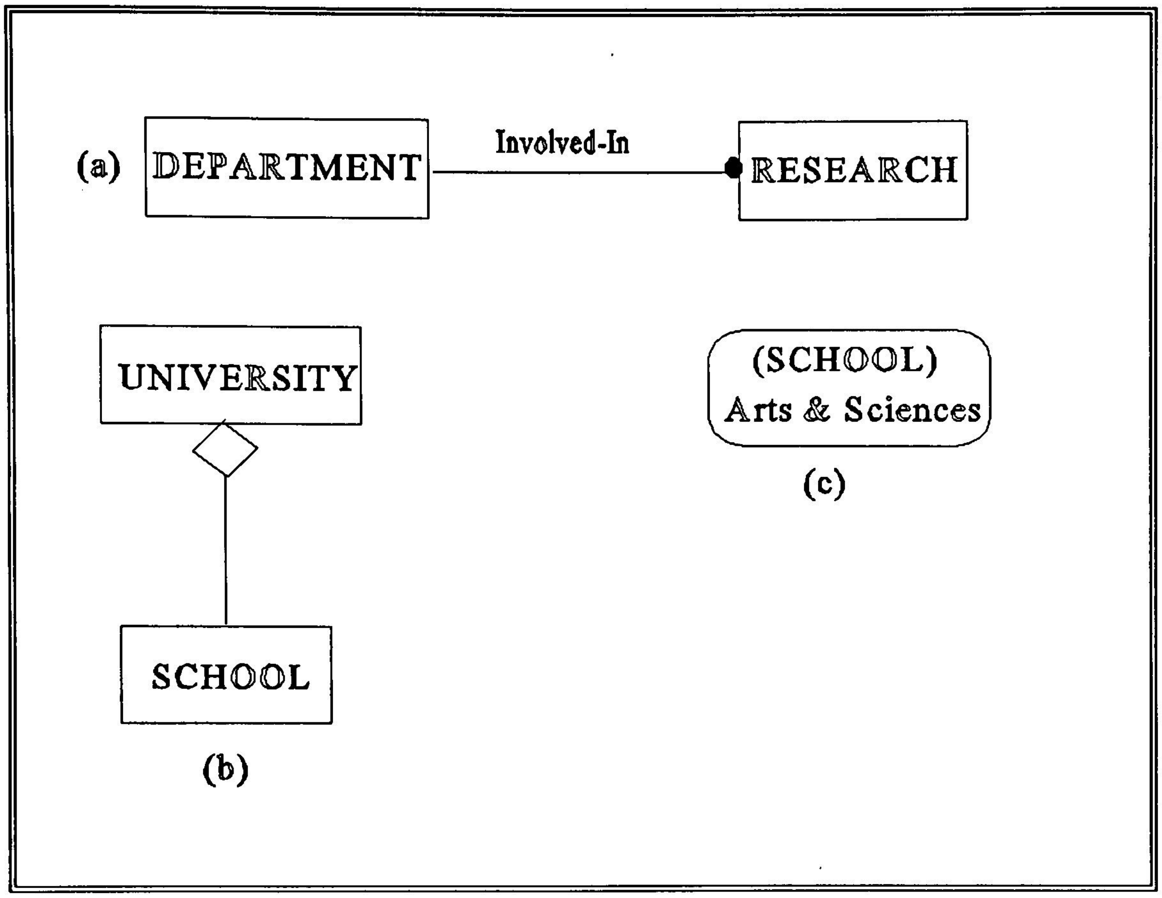


Figure 3. (a) Example of a generalized association between 2 classes. (b) An IS-PART-OF association. (c) An implicit IS-A association.

Given the inherent OODB concepts discussed above, a direct correlation can be drawn between classes of objects, inter-class relationships, object instances, and node and link types, node content and structure. The OODB model can therefore be used to automatically generate a basic hyper network which can be modified and maintained via traditional database mechanisms.

Limitations of the OODB Model

Although the object oriented data model as it exists provides some useful mechanisms for the development of a basic hyper network, it cannot be used to model the entire hyperdocument. This is due to the current limitations of existing object oriented programming environs (Tanaka and Qiang 1992). The OODB model only dictates nodes and links that correspond directly to the underlying data model. In a traditional hyperdocument, however, traversals between nodes that do not conform to the existing classifications of the OODB model via links that do not necessarily correlate with the intrinsic associations between classes is also possible.

Another serious limitation that is prevalent among traditional database environments is that there are no mechanisms to support the dynamic creation and modification of node and link types and the creation of views or versions of the data in the enterprise. In the conventional OODB the database schema is fixed and classes cannot be created or modified dynamically, and each object instance, although modifiable, is not allowed different versions.

These limitations can be addressed with extensions to the OODB model that allow the dynamic creation and modification of classes and versions of the object instances themselves. The *SAU* prototype will demonstrate such an extension.

CHAPTER III

THE CONCEPTUAL GRAPH AS A MEANS OF REPRESENTATION

Although the object oriented data model can be used to structurally organize the hyperdocument, it does not provide a formal notation to accurately convey the semantic associations that exist in the network. Early hyper models also fail to convey in a concise, readable format the semantic relationships between information nodes and links. The major deterrent is that it is difficult to communicate the author's interpretation of associations between categories of information. One common misinterpretation, for example, involves links that are unidirectional where the user may traverse the information link in the wrong direction.

In order to document clearly this intended meaning of information categories and associations, it is necessary to choose a notation that allows for detailed semantic expression and conveys it in a readable format. The model of choice is a conceptual graph which incorporates logic considerations in relationships between concepts. A concept may map to simple entities, situations or contexts such as states, processes, or events, or to other conceptual relations. This flexibility allows conceptual graphs to be easily translated to predicate calculus or directly mapped to other systems of logic, entity relationships models, or object oriented data models.

Conceptual graphs are formally defined (Sowa 1990) as a "system of logic based on . . . existential graphs . . . and the semantic network of artificial intelligence." They serve as a translation mechanism for other diagrammatic representations and embody the literal meaning of natural languages. Natural language notions such as coreference (relations between the same individual instance), quantification (such as *every* or *almost all*), and higher order types (these contain instances that are types themselves - eg. *model*, *species*, *color*, and *shape*) are all addressed via conceptual graphs. The simplest way to demonstrate the various forms and notations that are available in conceptual graphs is by example. Figure 3 shows an example of forms for a simple conceptual graph that represents the meaning of the situation or concept *A cat chasing a mouse*. From this example one can derive that the conceptual graph notation provides symbolic representations that explicitly dictate the *agent* and *patient* of the conceptual relation. There is no room for misinterpretation. The agent of the relation chase is the cat, and the patient is the mouse. Any correct interpretation of this graph will not allow, for example, the construction of the concept *A mouse chasing a cat*.

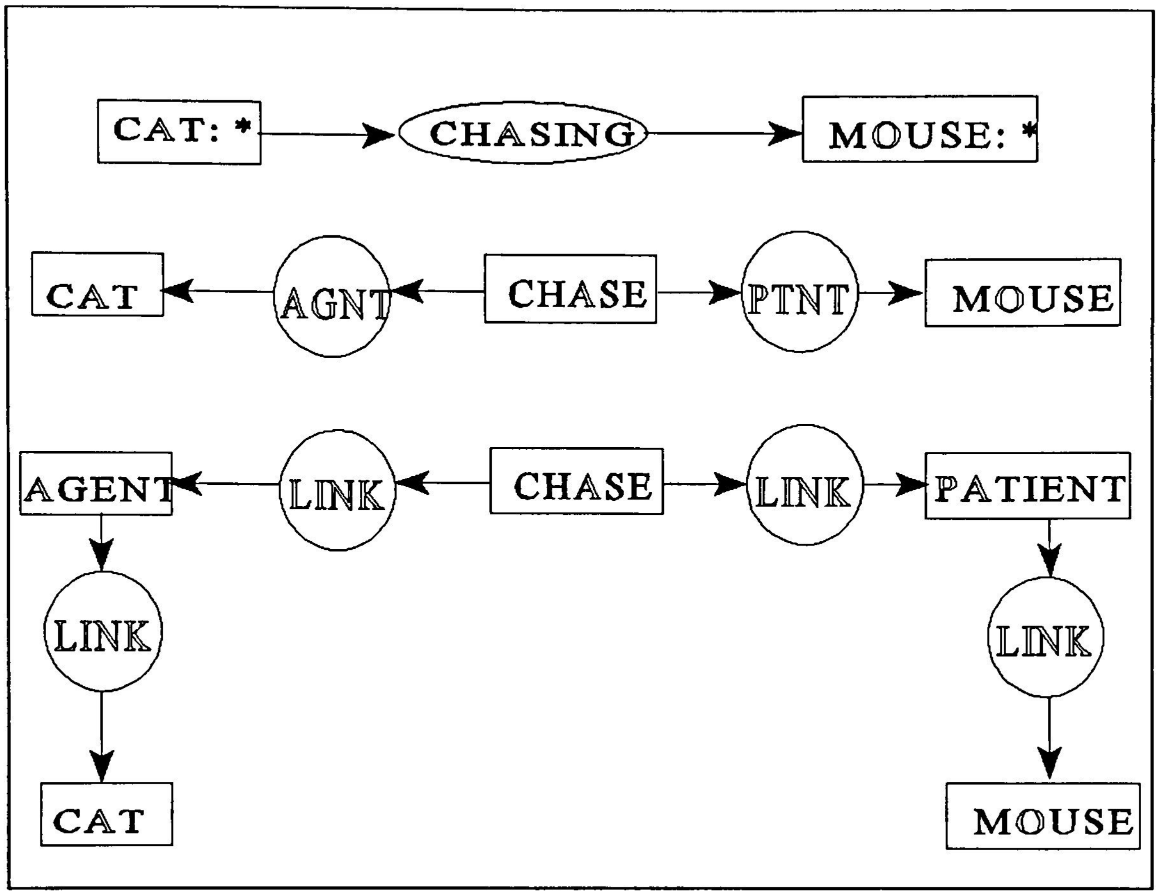


Figure 4. Three conceptual graph forms for the meaning of *A cat chasing a mouse.*

Further examination of Figure 4 indicate that concepts are represented pictorially by boxes and denote entities, actions, properties, or events. Boxes are composed of a type section and a corresponding referent field. Referents may refer to specific named instances or quantifiers of a particular type. The basic forms of referents allowed in conceptual graphs include *existential*, *individual* and *literal.*

Existential referents are denoted by the symbol * and specify the existence of an instance of a type - this can be mapped directly to the existential quantifier ∃ in predicate logic. Individual referents are symbolized by # followed by a unique identifier. These markers, as they are sometimes called, can be mapped to logic constants or serial numbers in a database. Literals are the form or value of entities and correspond to logic constants or programming language Literals. Referents may allow other logic considerations like the universal quantifier ∀ and queries (denoted by the ? symbol). Furthermore, in order to save space, the pictorial conceptual graph can be represented in linear form with square brackets denoting boxes and rounded parentheses for circles as in the following:

[CAT] <- (AGNT) <- [CHASE] -> (PTNT) -> [MOUSE]

Since conceptual graphs contain a rich library of symbolic and logical notations, relations are divided into three basic categories. These are *primitive*, *starter set*, and *defined.*

The primitive category encompasses the dyadic relation LINK and is used to derive other conceptual relations.

The starter set is an assortment of acceptable elementary conceptual relations. It contains recommended rules mandated by researchers in linguistics and artificial intelligence. Some examples include the following Sowa 1990):

- *Case relations* that show the connection between the action or state and the subject or entities represented. These include agent (AGNT), patient (PTNT), state (STAT), experience (EXPR), recipient (RCPT), instrument (INST), destination (DEST), and result (RSLT).

- *Spatial relations* which range from the generic location (LOC) to the more specific prepositions such as (IN), (ON), and (ABOV).

- *Attributes* that range from (ATTR) to characteristics (CHRC) and part (PART).

- *Intersentential relations* which are those relations that have referents that may represent several conceptual graphs themselves. They include before (BFOR), after (AFTR), cause (CAUS), purpose (PURP), method (METH), and the logical (AND) and (OR).

- *Mathematical relations* which denote measure (MEAS), greater than (>), less-than (<), equal (=), not-equal (≠), average (AVG) count (CNT), and functional arguments (ARG1) or (ARG2).

- *Metarelations* that describe the relation between concept types and relations, and conceptual graphs. Kind (KIND), subtype (SUBT), description (DSCR), statement (STMT), and representation (REPR) are all metarelations.

The third branch of conceptual relations is the defined category. This classification entails any relation used in logic, relational databases, or entity-relationship diagrams. These relations are predefined by their original frame of reference. In other words, the dialect of existing database models are also incorporated as part of the conceptual graph notation.

As shown from the above examples, the rules for conceptual graphs are general enough to encompass varying systems of logic, as well as other diagrammatic and notational representation. However, there is enough specificity and flexibility to accommodate the nuances of natural languages. This duality allows conceptual graphs to be used as a universal mechanism for expressing the syntax and semantics of any given system.

To demonstrate the translation of an object oriented data model to a conceptual graph, Figure 5 shows the representation of a generic class definition which has two sub-classes and three methods. The definition includes the universal quantifier ∀ which implies every instance of the class. The nested graph shows that ?c has subclasses *ca and *cb, and methods 1, 2 and 3.

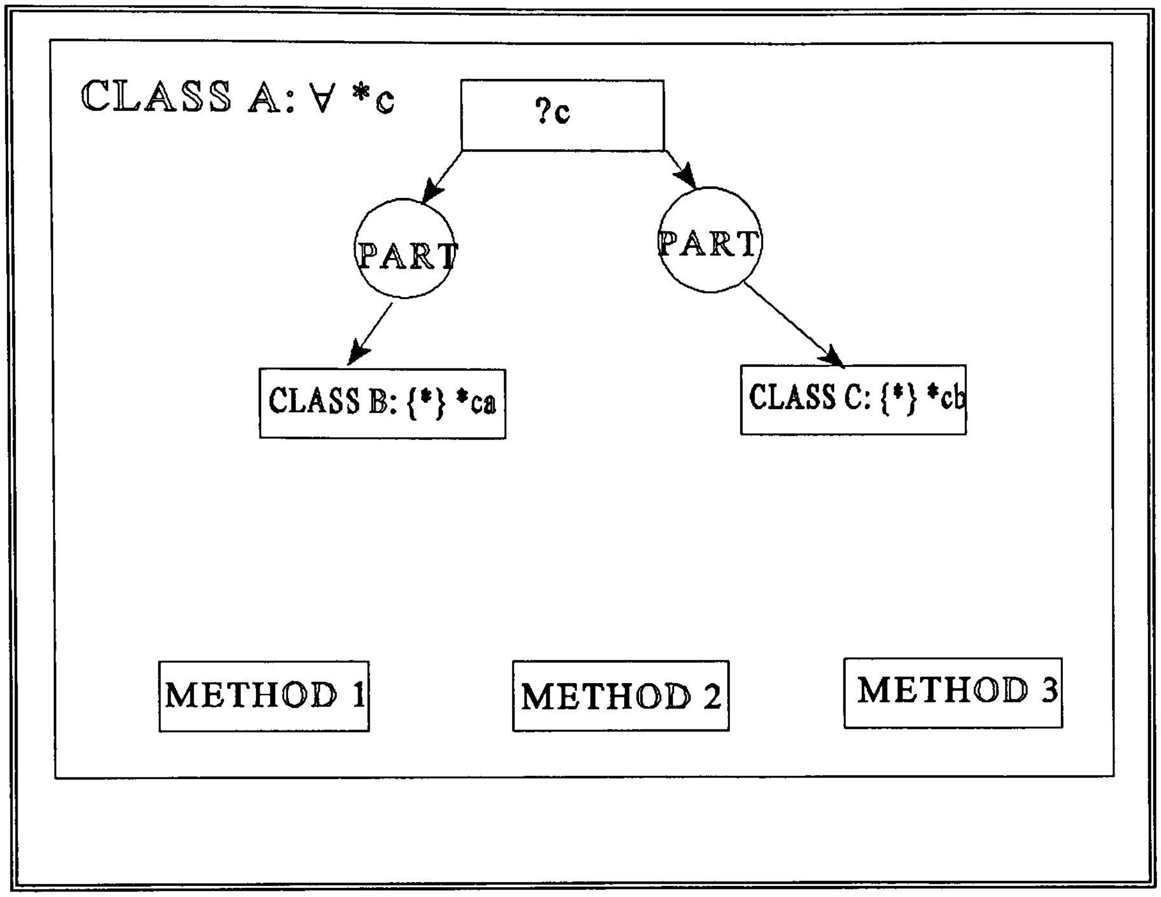


Figure 5. Conceptual graph showing a generic class definition.

CHAPTER IV

SAU OODB MODEL

An object oriented data model allows the user to manipulate classes of objects via predefined methods. In the *SAU* application, seven primary classes of objects are defined. They are as follows:

UNIVERSITY	-	This class possesses general attributes such as Name, Location, Acreage, Population and History.
RESOURCE	-	RESOURCE objects are associated with UNIVERSITY objects and depict the general resources associated with each university, such as its library and computing facilities.
SCHOOL	-	Objects of this class are embedded within objects of class UNIVERSITY. They possess attributes such as Name, Dean, and a pictorial representation of the school's primary building.
DEPARTMENTS	-	School objects have embedded links to objects of this class. Research Projects is the main attribute of this class and is represented as an embedded object of the class RESEARCH.
RESEARCH	-	RESEARCH objects define the categories of research associated with individual departments.
FACULTY	-	FACULTY objects contain information on the Name, Education, and Specialty of faculty members for each department. Sets of these objects are associated with objects of the class DEPARTMENT via embedded links.

STUDENTS	-	These objects are similar to the FACULTY objects in the sense that they are also embedded in objects of the class DEPARTMENT. They contain relevant attributes such as current GPA and Major.

The above classes and their interrelationships are depicted in Figure 6.

In the *SAU* OODB model, each class has several predefined methods and attributes. Attributes define the structure of objects and methods are used for data manipulation. See table 1 for a description of the major attributes and methods associated with the classes in the *SAU* OODB model.

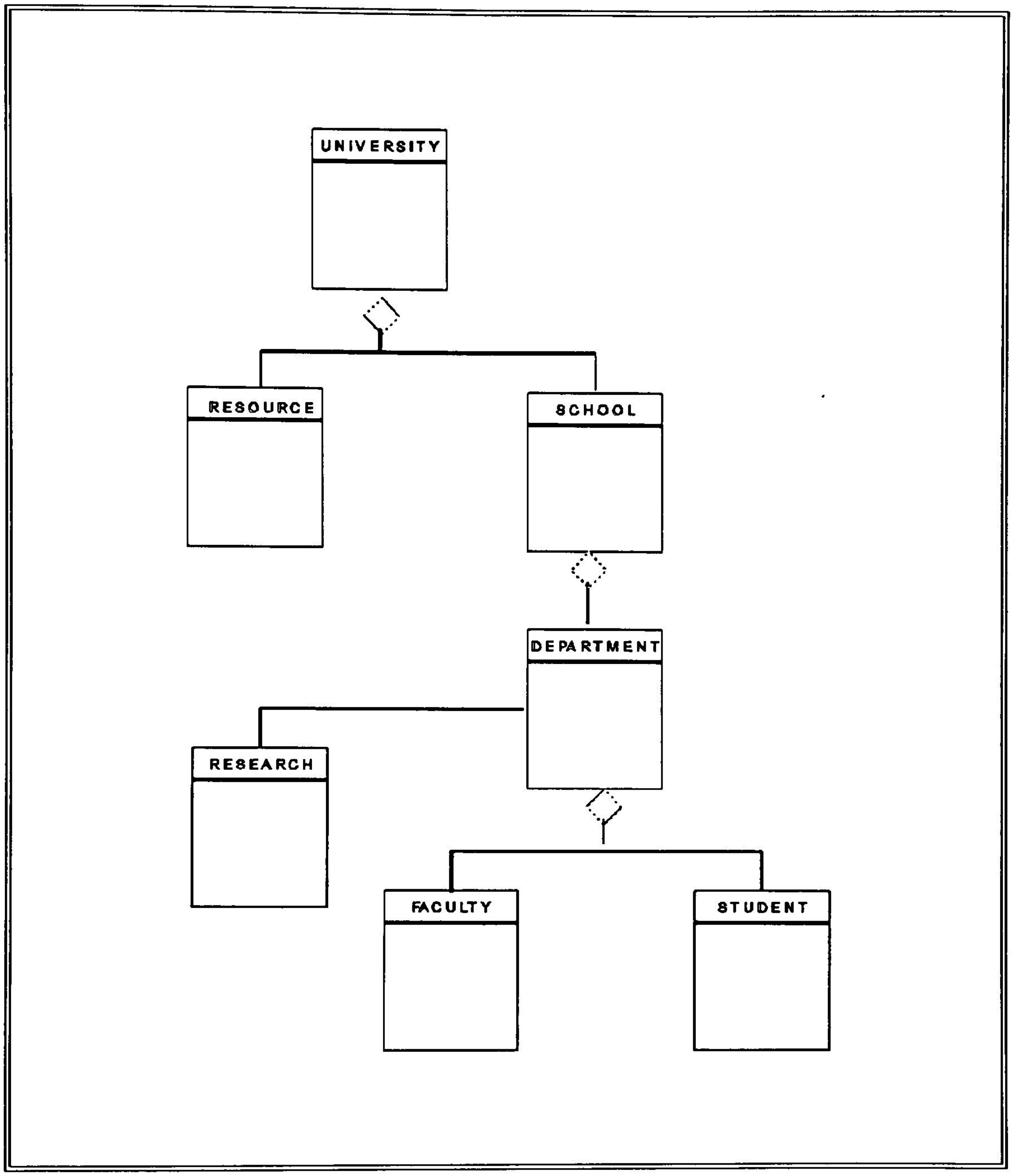


Figure 6 SAU OODB Model (OMT symbolic notation).

Table 1.--Table showing classes, attributes and methods of the SAU OODB Model.

CLASS NAME	Major Attributes	Methods
UNIVERSITY	Name (string) Type (string) Location (Address) History (Sound) Logo (Bitmap) Picture (Image)	Edit Display Add Resource Add School
RESOURCE	Name (string) Type (string) Picture (Image)	Edit Display
SCHOOL	Name (string) Dean (string) Dean's Picture (Image) School's Picture (Image)	Edit Display Add Department
DEPARTMENT	Name (string) Chair (string) Chair's Picture (Image) Department's Picture (Image)	Edit Display Add Faculty Member Add Student
FACULTY	Name (string) Credentials (Sound) Picture (Image)	Edit Display
STUDENT	Name (String) Statement (String) Picture (Image)	Edit Display

CHAPTER V

SAU HYPER MODEL

The *SAU* interface must be engaging and flexible enough to allow varying interpretations of the data in the enterprise. This entails allowing users not only to select a university of interest and explore its attributes, but to compare and contrast common elements associated with these objects. A user should be able to display the faculty and students associated with a university, for example, as well as define and follow common trends in the research projects sponsored by each institution. Providing an interface that facilitates this dynamic interpretation of the data in the enterprise generally requires a model that incorporates the definition and creation of nodes and links.

An elementary hyper model for the *SAU* prototype can be generated from the underlying object oriented database. However, the dynamics of the hyper interface that are not supported by the underlying OODB must be incorporated as extensions to the traditional model. This necessity influenced the choice of the O2 Object Oriented Database Management System Version 4.4.5 as the implementation instrument. O2 facilitates the major characteristics of traditional hyper systems. To allow the ad hoc creation of information nodes and links, for instance, a meta schema (O2 User Manual 1993) allows the dynamic evolution of classes and associations. Furthermore, versions of object instances and their associations are allowed which can facilitate the creation of different user views in the hyper network.

This chapter explores the possible navigational paths through the *SAU* database made possible by its hyper interface. The interface allows users the same flexibility offered by traditional hyper systems.

SAU supports the two most common forms of hyper navigation in a database - structural and associative. Structural navigation follows the IS-PART-OF and IS-A associations that are already a part of the object oriented database. In the *SAU* application there are several of these types of associations (refer to Figure 6). For instance, there is an IS-PART-OF association between the classes UNIVERSITY and RESOURCE, and between DEPARTMENT and FACULTY. IS-A generalizations are implicit in the data model and are formed whenever an object instance of a particular class is created.

Associative navigation is the traversal across broad categories in the UoD. Whereas the *SAU* OODB model only allows associative links between information categories that map directly to a predefined class, its hyper interface allows typed links between nodes that may contain portions of one or several classes. For instance, a link from Clark Atlanta University to Georgia Tech is a specific instance of a typed link between objects of the class University. However, typed links are also defined for comparisons between schools from the same or different universities. A comparison that includes the number of departments in schools across universities implies that a typed information segment be created that includes the attribute name from the class UNIVERSITY and SCHOOL and the number of departments. These attributes do not correspond to any one class in the OODB model.

Accessing information through a hyper interface may also involve some logical decision(s) being made. This method of accessing information may use either structural or associative navigations, however, at some point in the traversal the result of a condition determines the intermediate or final outcomes. Consider for example, a request to identify all the Universities that are privately funded and that have a Computer Science department. The traversal of such a link may not necessarily include all of the university nodes even if such a path already exists. All of the universities that are publicly funded need to be skipped. In addition, all of the universities without a Computer Science department must not be included. Generally, such logically influenced access paths may be controlled either by identifying attributes or the results of a query, or both. The hyper link described above, for instance, may be determined by some attribute of the class UNIVERSITY which identifies whether it is public or private, along with some querying facility to determine whether a Computer Science department exists. It should be noted that the identifying attribute and querying facility are all provided via the underlying database model.

To further illustrate the effectiveness of using an existing data model to formulate a hyper interface SAU allows nodes to be created that contain information from the direct results of a database query. This type of node is dynamic in the sense that the information it represents will reflect any modifications performed on the contents of the database. Therefore, it is not necessary to update the node content whenever a database change occurs. Furthermore, since the entire structure of the hyper network is closely related to the database model, any changes to the existing model will be automatically reflected not only in node content but in the associations or links themselves.

For a demonstration of some of the hyper links formed by the *SAU* interface, refer to chapter VII.

CHAPTER VI

SAU CONCEPTUAL GRAPH MODEL

Given the complexity and scope of the inter-relationships that exist and that may be formed between information nodes and links, it is necessary to maintain a dynamic, and logic oriented model which captures all of its semantic constraints. This model should be structured enough to allow developers to keep track of the existing hyper web, yet readable and flexible enough to allow users to understand the patterns of associations that exist and create their own. Since the OODB model itself suggests some of the associations and links of the database, it would also be helpful to extract direct correlations from the object model. The *SAU* application utilizes the conceptual graph style of notation to document its hyper interface, since it not only captures logic in relations but can be mapped directly to the underlying OODB.

The initial conceptual model for the hyper interface can easily be generated from the underlying OODB. The association between the class UNIVERSITY and RESOURCE as depicted in Figure 3 is directly translated to the linear conceptual graph segment as in

[UNIVERSITY] <- (HAS) -> [RESOURCE]

One other common access paths or hyper links documented in the database follows the heiarchy of embedded classes. For instance, the link from a University object to a School object can be depicted in conceptual graph notation for the general case and for specific instances as in

[UNIVERSITY: CAU] <- (PART) -> [SCHOOL: Business]

This notation can be repeated at various levels of detail and would suffice to document the entire SAU hypermodel.

CHAPTER VII

DEMONSTRATION OF *SAU*

This chapter is devoted to a brief pictorial demonstration of the *SAU* prototype. The demonstration walks the user through the traditional object oriented database access methods and then displays some of the common access paths already defined by the hyper interface. The demonstration was run on a SUNOS workstation with OpenLook graphical user interface. Screen images were captured using Snapshot version 3.3.

Images are included to give the reader an idea of the look and feel of the *SAU* prototype. Since the application is not completely populated with real data, test data is used wherever appropriate. The primary objective is to demonstrate the ease with which the interface allows the user to view complex, varied, and informative views of the data in the enterprise.

Figure 7 shows the main icon or presentation which pilots the rest of the application. A pull down menu of the application level programs is also shown. The two major entries are the *DB Interface* and the *Hyper Interface* selections.

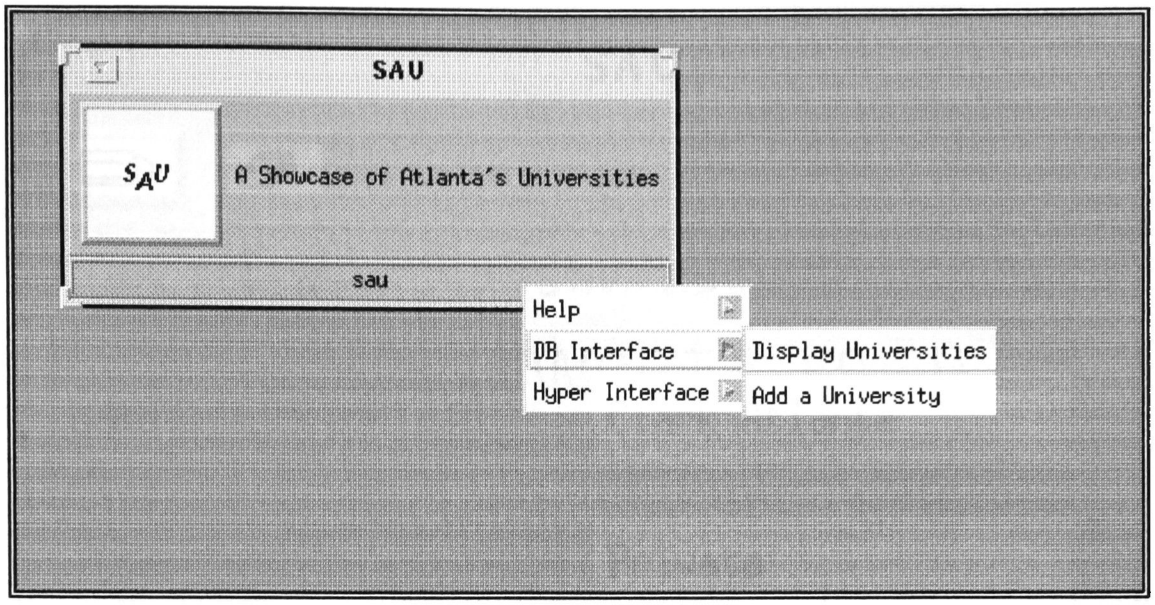
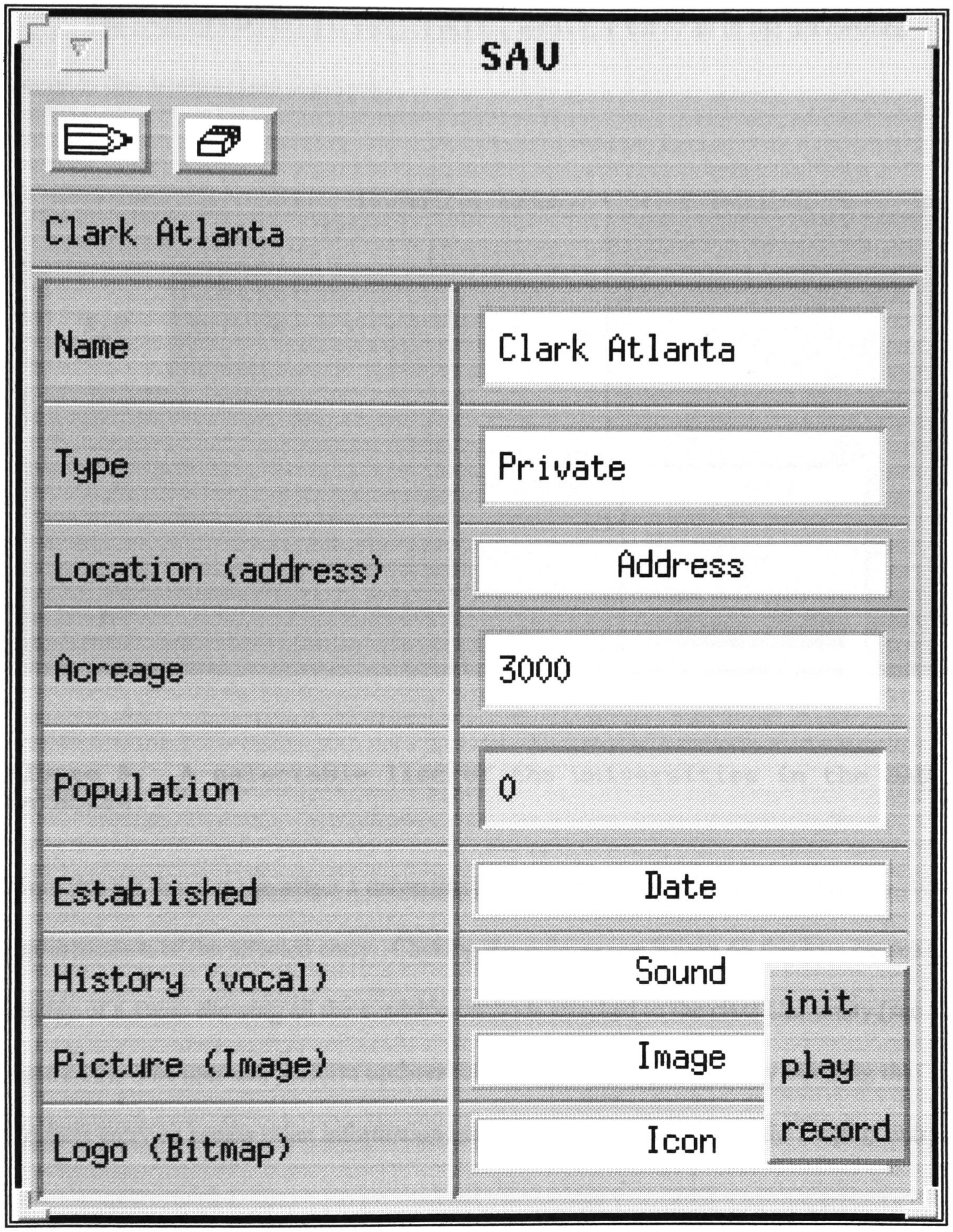


Figure 7. *SAU* pilot presentation showing main menu selections *DB Interface* and *Hyper Interface*.

Selecting the *DB Interface* allows the user to display all of the traditional database application programs. These include the *Add University* and *Display Universities* entries (refer to Figure 7). The *Add University* selection triggers a data entry window which contains the tuple of attributes associated with the class UNIVERSITY (see Figure 8). This presentation is modifiable and clicking on the "Pencil" button stores any changes made to the database, provided that certain constraints are met. These constraints include duplicate checking, and required field entries. A method menu is displayed by clicking on the right mouse button for those fields that correspond to objects themselves. Clicking on the "Eraser" abandons any changes made to the presentation.


SAU

Clark Atlanta

Name	Clark Atlanta
Type	Private
Location (address)	Address
Acreage	3000
Population	0
Established	Date
History (vocal)	Sound
Picture (Image)	Image
Logo (Bitmap)	Icon

Figure 8. An editable tuple display resulting from the Add University selection.

The *Display Universities* entry triggers a selectable list of all of the universities stored in the database (see Figure 9).

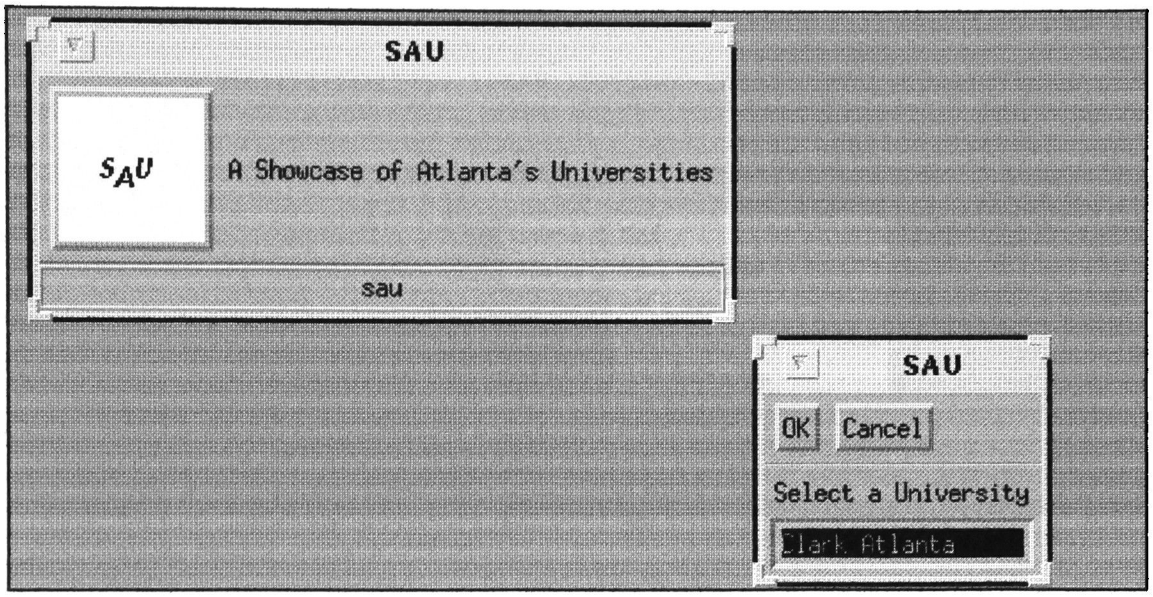


Figure 9. A selectable list of the universities in the *SAU* database.

From this list the user can select a university by name and be presented with an iconic representation of the selected entry. Clicking the right mouse button on this icon causes a display of a menu showing all the available methods attached to the class University (see Figure 10). The user can perform updates to the university object itself by selecting the *Updates* entry or access other information associated with the object via the *Display* entry.

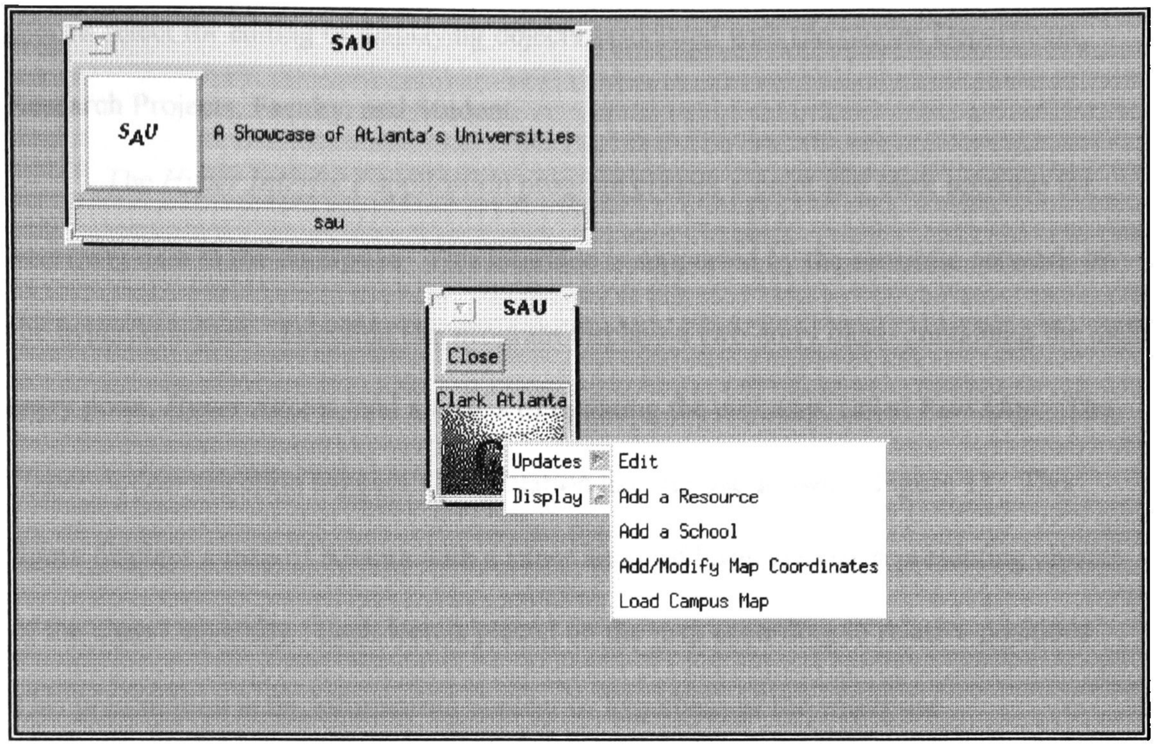


Figure 10. Iconic representation of a university with associated methods.

Displays, like edits, can be performed on each of the tuple of visible attributes. Pictorial and sound attributes can be read via the appropriated methods which are selected by clicking the right mouse button on the corresponding entry fields. In addition, the resources associated with the university can be accessed via a selectable list for edits or viewing, by selecting the Display Resources method.

The *Display* entry also contains the method that facilitates stepping through the heiarchy of embedded objects in the database. This method is called *Display Schools*, and when selected, allows the user to select from a list of all the schools associated with a particular university. From this point on the *DB Interface* repeats in form, allowing the user to select for editing or displaying objects associated with the classes Department, Research Projects, Faculty, and Student.

The *Hyper Interface* application program creates a point and click scenario for accessing data in the enterprise. This interface is supported by the semantic network of the hyperdocument. Selecting this program displays a two entry menu containing the main entry point, *Hyper Access*, and a means of updating the city map, called *Edit Map*. The *Hyper Access* selection triggers a presentation like the one shown in Figure 11. This figure displays a map of Atlanta with a super imposed layer of icons representing objects of the class University. Each icon is placed on the map according to relative positions. This presentation is the information content or hyperbase of the enterprise.

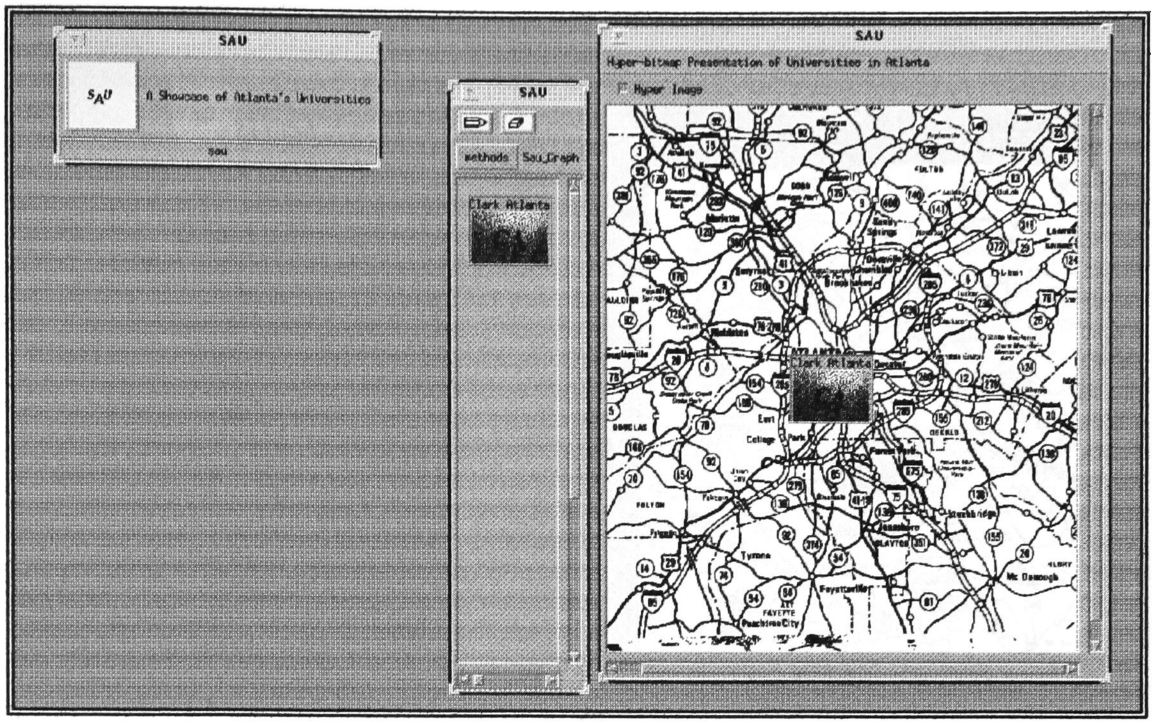

SAU
A Showcase of Atlanta's Universities
SAU
methods
Sau_Graph
Clark Atlanta
SAU
Hyper-bitmap Presentation of Universities in Atlanta
Hyper Image

Figure 11. *SAU Hyper Access* presentation.

The accompanying presentation is a *hyper graph* showing the corresponding nodes and any associated links. The hyper graph in Figure 11 corresponds to the top level hyperindex layer. This layer is used to display the current access path traversed by the user and actually pilots the user via the hyperdocument through the information in the database. For example, selecting a node from Figure 11 highlights the selected node and displays a pictorial representation of the corresponding University object (see Figure 12). At the same time an audio segment containing the contents of the history attribute is played. In addition, the user is presented with the option of displaying the resources

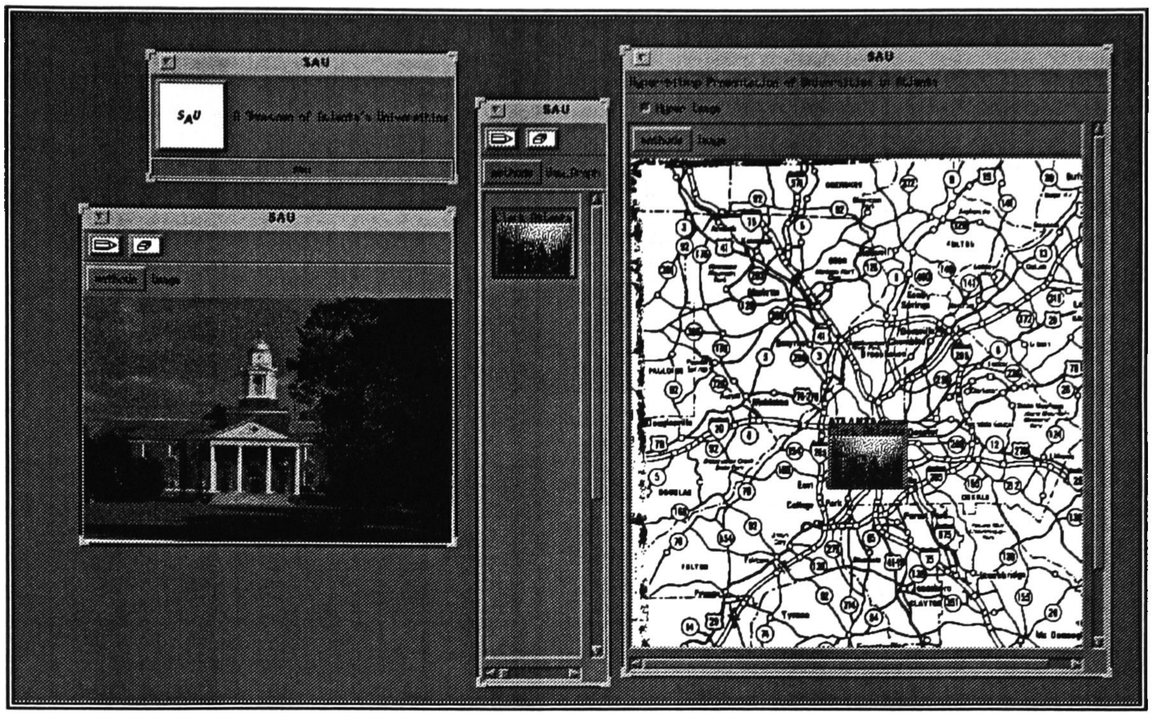


Figure 12. *SAU* hyper presentation showing first level of display for an expanded view of the university node.

associated with the university. After the user erases the university image, or finishes perusing the resource information, a display of the campus map with a layer of superimposed School objects placed according to relative positions is triggered (see Figure 13). A hyper graph demonstrating the path of traversal in the form of a simple directed graph is also shown. This graph in turn contains the nodes corresponding to all the associated School objects.

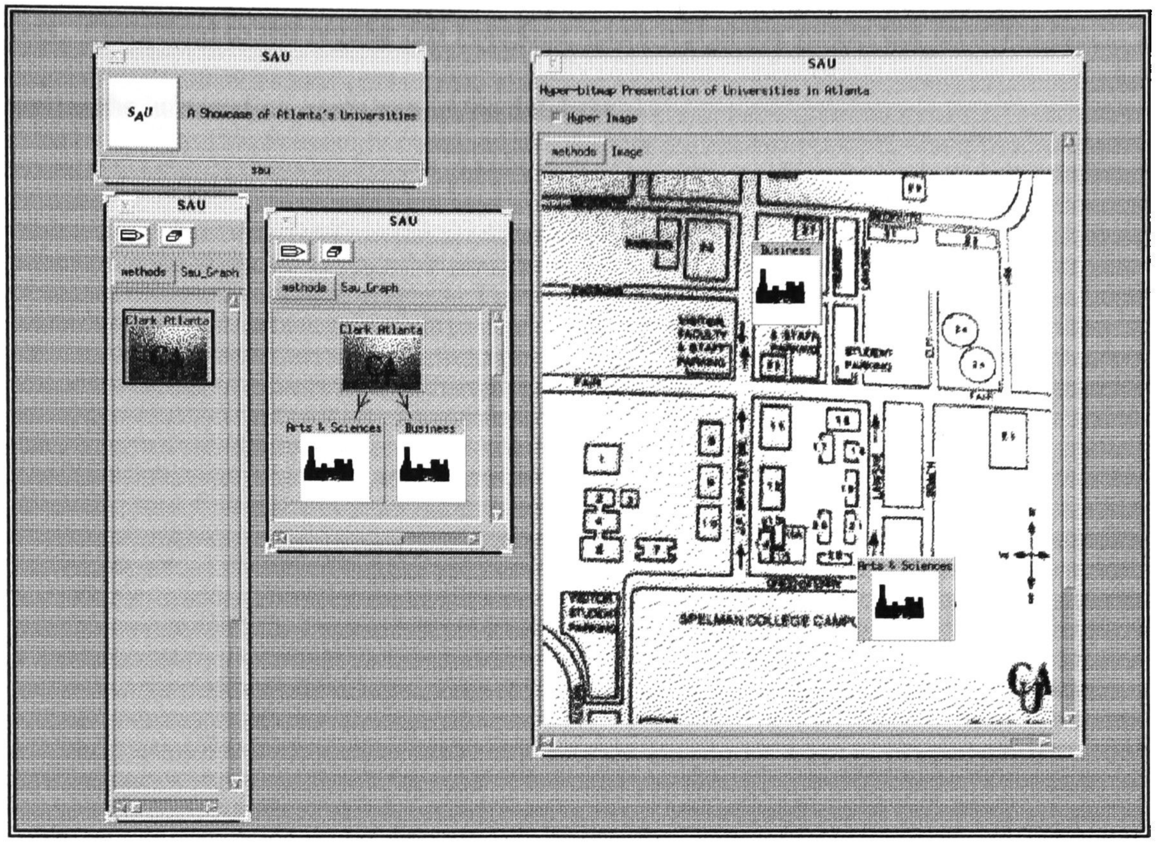


Figure 13. SAU hyper access presentation showing second level of display for the expanded university node.

The *Hyper Access* application program repeats in form to allow the user to view all the information in the database. Although not shown, modifications to existing node content and the addition of new nodes and links are allowed interactively via the hyper graphs. These actions will trigger the appropriate database updates in future revisions of the SAU prototype.

Ease of use and power to present comprehensive views of the database is clearly demonstrated through the *Hyper Interface* application program. This facility offers a single, and coherent outlook on the enterprise, and also allows the user to interact with the data. Compared to the regular *DB Interface*, users will prefer to use this interface to peruse the information contained in the database.

CHAPTER VIII

CONCLUSION

The demonstration of *SAU* clearly shows that a hyper interface provides a useful tool for viewing and interpreting complex information in the enterprise. The fact that the interface itself was easily generated from the underlying database model suggests that this approach is a feasible mechanism for enhancing the power and generality of existing hyper systems. This two-leveled architecture also addresses some of the common problems associated with hyper systems. The abstract interpretation of the top layer allows it to be used as an index so that users can retreat from more specific to general contexts. The specificity offered by the bottom layer enables continual refinement of the current context. This ability of inter-layer navigation can be used as an effective strategy for recovering from a too narrow context to one that allows the user to explore other navigational options within the enterprise. Furthermore, the predefined data manipulation mechanisms offered by the underlying database, in this instance method definitions, is used to provide a means of viewing information about node content. In other words, the readers for viewing the content of a node are actually predefined methods that belong to the class that corresponds to the node type. If the node content contains a bitmap image, for instance, the predefined method to display the image is readily available as part of the underlying database.

One other benefit of the two-layered approach is that it provides the means of creating abstractions or structured types for information nodes and associations. These abstractions can be used to structurally organize the hyperdocument and allow the creation of multiple instance versions within the enterprise. In the *SAU* prototype, the abstractions are provided as part of the general class schema of the object oriented database management system. Classes are used to provide structure to information nodes, and the associations formed between classes allow the creation of typed links. In addition, the search for and creation of new associations is augmented by the ad hoc querying facilities that are included as part of the O2 database management system.

Using O2 to implement the *SAU* prototype overcomes the limitations of the OODB model as it relates to the dynamic creation of hyper nodes and links. O2 allows runtime modifications of the database schema and the creation of versions of object instances within the enterprise. Since the hyper interface is derived from the database model, any changes to the schema is automatically reflected in the nodes and links themselves. Therefore, the ability to create new classes facilitates the dynamic creation of node types and associations. In addition, the ability to create versions of object instances allows versions of node content within the hyper network.

Using the conceptual graph style of notation to document the hyper interface of the prototype system allows the author to represent in sufficient detail the semantics behind the network of information nodes and links. Since the network is subject to interpretation, the documentation of the associations between categories of information is imperative to readability and maintainability. The capacity for conceptual graphs to illustrate the meaning behind relations, and to do so in a manner that offers varying levels of detail, makes it an ideal tool to document hyper systems. The conceptual graph notation is also desirable because of its direct correlation to database paradigms. The *SAU* OODB model was easily translated to a conceptual graph model. This conceptual model forms the basis for documenting the hyper interface.

For future development, the conceptual graph will serve as a reminder of prior interpretations of the data within the enterprise. Maintaining the hyper network will be easier since its semantics are documented pictorially. Therefore, the authoring of information nodes and links for future versions of the *SAU* prototype will be facilitated.

Future Work

This section provides an indication of future revisions to the *SAU* application. The primary focus of all revisions will be to make improvements to the hypermedia interface while attempting to minimize the resulting disorientation that may occur from navigating through the database. The intent is to provide ease of access to the information in the enterprise. In addition, the user should be able to define additional views that show correlations between units of information. Currently, the application only allows limited definition of such views, namely only associations between existing classes. These associations follow the structural links in the database. Implementation of the traversal between associative links in the enterprise is an immediate concern. Currently the hyper graph facility allows the insertion of new links between instances of any class. This capability will be used to define pre-existing and user defined associations. In addition, future revisions will utilize the meta schema definition of O2 to allow users to interactively create new classes and associations.

Another current limitation of the *SAU* prototype is that there are no separate versions of the data in the enterprise. This possibility can easily be facilitated by the version creating mechanisms in O2, however the initial data in the enterprise does not support such interpretations as yet. As the database is populated with more realistic and complete data, opportunities to create separate versions will be more readily available.

When *SAU* is completed with regards to its hyper interface and data content it will not only serve as a true showcase of Atlanta's area universities, but as a model for future development of hyper interfaces for object oriented databases. The hyper interface is generic in the sense that it does not depend on data content, but uses the generality afforded by the object oriented database schema to create its semantic network of nodes and links. Therefore, in principle, a hyper interface can be automatically generated from an underlying OODB. Since the *SAU* prototype documents this generation via conceptual graphs, this notation can be used as a formal guideline for replicating the process.

As mentioned before, the process of generating the hyper interface is independent of data content. Therefore, any data system that can be supported by the object oriented data model may suffice. This includes software documentation repositories, Computer Aided Design systems, and other complex information enterprises. A hyper interface will facilitate ease of use and allow such systems to be easily maintained and configurable. Future spinoffs of *SAU* may include example of such systems.

APPENDIX 1

SAU OODB SCHEMA DEFINITION

```
class University
        type tuple (public name        : string,
                    public type        : string,
                    public location    : Address,
                    public acreage     : integer,
                    public population  : integer,
                    public established : Date,
                    public history     : Sound,
                    public picture     : Bitpic,

                    private Resources  : set (Resource),
                    private Schools    : set (School),
                    ... )

        method      public edit : integer,
                    public Add_Resource,
                    public Display_Resources,
                    public Add_School,
                    public Display_Schools,
                    ...

end;

class School
        type tuple (public name        : string,
                    public dean        : string,
                    private Departments : set (Department)
                    ... )

        method      public edit : integer,
                    public Add_Department,
                    public Display_Departments,               ...
end
```

```
class Resource
        type tuple (public name        : string,
                        public owner       : string,
                        public picture     : Bitpic,
                        public description : Sound
                        ... )

        method          ...
end


class Department
        type tuple (public name        : string,
                        public chairperson  : string,
                        public degrees      : list (string),
                        public majors       : list (string),
                        public num_faculty  : integer,
                        public num_students : integer,

                        private  Research_Projects : set (Research),
                        private  Faculty           : set (Faculty),
                        private  Students          : set (Student)
                        ... )

method                  public Add_Research_Project,
                        public Display_Research_Projects,
                        public Add_Faculty_Member,
                        public Display_Faculty,
                        public Add_Student,
                        public Display_Students,
                        ...
end;

class Research
        type tuple (public name        : string,
                        public sponsor     : string,
                        public director    : string,
                        public description : Sound
                        ... )

        method          ...
end
```

```
class Faculty
        type tuple (public name       : string,
                    public degree     : string,
                    public alma_mater : string,
                    public specialty  : string,
                    public statement  : Sound,
                    public photo_id   : Bitmap,
                    ... )

        method         ...
end;

class Student
        type tuple (public name      : string,
                    public major     : string,
                    public statement : Sound,
                    public photo_id  : Bitmap,
                    ... )

        method         ...
end;
```

APPENDIX 2

SAMPLE SAU APPLICATION CODE

```
/*****    The Showcase of Atlanta's Universities (SAU) application
 ******************************************************/
application sau
        program
                init,
                restart (why            : integer,
                        ...),

                public Describe_Application,
                public Display_Authors,
                public Display_Universities,
                public Add_University,
                public HyperAccess,
                public edit_map,
                public disp_graph
end;

transaction body init in application sau {
  commit; /* launch restart */
};

...

/*****    Dialoger (imported from o2kit)
 ******************************************************
 *****
 ***** Dialoger is a named object which serves for*
 ***** * displaying messages
 ***** * making multiple selections
 *****/

import schema o2kit name Dialoger;
import schema o2kit class Box;
```

```
run body { Dialoger = new Box; };

/*****    Date (imported from o2kit)
 ******************************************************
 *****/
import schema o2kit class Date;

/*****    Bitmap (imported from o2kit)
 ******************************************************
 *****/
import schema o2kit class Bitmap;
create class Bitpic
  inherit Bitmap
    public
end;

/*****    Image (imported from o2kit)
 ******************************************************
 *****/
import schema o2kit class Image;

/*****    Percistency roots.
 ******************************************************/
name Universities : set(University);
name Atlanta      : Univmap;
run body { Atlanta = new Univmap; };
```

Method bodies of the class University

```
method body init in class University {
/*MAY NEED TO PASS ATTR. AS PARAMETERS HERE*/
  self->established = new Date (0, 0, 0);
};

method body title: string in class University
{  /* use University name as title */
  return self->name;
};


method body menu: list (string) in class University {
  return list ("display", "edit",
            "Add_School", "Display_Schools",
            "Add_Resource", "Display_Resources");
};

method body edit: integer in class University {
  o2 extern set (string) Exist_Univ_Names;
  o2 University univ;
  o2 University temp = new University;
  o2 string oldname = self->name;
  o2 string un;

  Lk_presentation uvp, tmpp, cp = lk_current_presentation();

  int wval;

  temp = (o2 University) self->deep_copy;
  tmpp = univtpl (self);
  uvp = univtpl (self);

  lk_map (tmpp, LK_COORDINATE, 0, LK_SCREEN, 200, 300);
  wval = lk_wait (tmpp);
  if (wval == LK_SAVE)
  /* user saves edits */
  {

    lk_consult (tmpp, temp);

    if (temp->name == "")
    {  /* University name is required */
```

```
      Dialoger->message("Name is required", "");
      lk_consult (uvp, self);

      lk_delete_presentation (tmpp);
      return LK_ERASE;
    }




    for (un in Exist_Univ_Names where un == temp->name)
      if (temp->name != oldname)
      {  /* Duplicate Universities are not allowed */
        Dialoger->message("Duplicate name", "");
        lk_consult (uvp, self);

        lk_delete_presentation (tmpp);
        return LK_ERASE;
      }
  }
  lk_consult (tmpp, self);

  Exist_Univ_Names += set (self->name);
  Exist_Univ_Names -= set (oldname);




  self->refresh_all;

  lk_delete_presentation (uvp);
  lk_delete_presentation (tmpp);
  return wval;
};


method body bitmap: tuple (width  : integer,
                           height : integer,
                           bitsmap: bits)
                      in class University {

  if (self->picture != nil)
    return *(self->picture);
};
```

```
method body Add_School in class University {
  o2 School sch = new School;
  o2 School school = new School;
  o2 list(string) sch_names;
  Lk_presentation scp;
  int wval;

  scp = schtpl (school);

  school->disable_method ("edit");
  school->disable_method ("display");
  school->disable_method ("Display_Departments");

  lk_map (scp, LK_COORDINATE, 0, LK_SCREEN, 200, 100);
  wval = lk_wait (scp);
  if (wval == LK_SAVE)
  {  /* user saves edits */
     lk_consult (scp, school);

    if (school->name == "")
    {  /* School name is required */
      Dialoger->message("Name is required", "");
      lk_delete_presentation (scp);
      return;
    }
    for (sch in self->Schools)
      /* collect all Schools that belong to this University */
      sch_names += list(sch->name);

    if ((school->name in sch_names ) != -1)
    {  /* Duplicate Schools are not allowed */
      Dialoger->message("Duplicate School", "");
      lk_delete_presentation (scp);
      return;
    }

    transaction;
    self->Schools += set(school);
    validate;
  }
  lk_delete_presentation (scp);
};
```

```
method body Display_Schools in class University {
 o2 extern set (string) Exist_Sch_Names;
 o2 string college, edname;
 o2 list(string) school_names;
 o2 School sch = new School;

 Lk_presentation scp;
 int wval;

 Exist_Sch_Names = set ("");
 for (sch in self->Schools) {
   /* collect all Schools for this University */
   school_names += list (sch->name);
   Exist_Sch_Names += set (sch->name);
 }

  if (count(school_names))
    /* get user selection */
    college = Dialoger->selection("Select a school", "", school_names);
  else
  {
    Dialoger->message("No data found", "");
    return;
  }

  if (count(college))
  {
    for (sch in self->Schools)
      /* find and display selected School */
      if (college == sch->name)
        break;

     edname = "scobj";
     scp = ictpl (sch, edname);
     lk_map (scp, LK_COORDINATE, 0, 0, 200, 100);
     wval = lk_wait (scp);
     lk_delete_presentation (scp);
     /*display (sch);*/
  }
  /* commit; */
};

method body Add_Resource in class University {
  o2 Resource resource = new Resource;
  o2 Resource reso = new Resource;
  o2 list(string) reso_names;
  Lk_presentation rsp;
  int wval;

  rsp = resotpl (resource);

  resource->disable_method ("edit");
  resource->disable_method ("display");

  lk_map (rsp, LK_COORDINATE, 0, LK_SCREEN, 200, 100);
  wval = lk_wait (rsp);

  if (wval == LK_SAVE)
  {  /* user saves edits */
    lk_consult (rsp, resource);
    if (resource->name == "")
    {  /* Resource name is required */
      Dialoger->message("Name is required", "");
      lk_delete_presentation (rsp);
      return;
    }
    for (reso in self->Resources)
      /* collect all Resources for this University */
      reso_names += list (reso->name);

    if ((resource->name in reso_names) != -1)
    {  /* duplicate Resources not allowed */
      Dialoger->message("Duplicate Resource", "");
      lk_delete_presentation (rsp);
      return;
    }
    transaction;
    self->Resources += set(resource);
    validate;
  }
  lk_delete_presentation (rsp);
};

method body Display_Resources in class University {
  o2 string res_name;
  o2 list(string) res_names;
  o2 Resource reso = new Resource;
```

```
 for (reso in self->Resources)
   /* collect all Resources for this University */
   res_names += list(reso->name);

 if (count(res_names))
   /* get user selection */
   res_name = Dialoger->selection("Select a resource", "", res_names);
 else
 {
   Dialoger->message("No data found", "");
   return;
 }

 if (count(res_name))
 {  /* find and display selected Resource */
    for (reso in self->Resources)
       if (res_name == reso->name)
          break;
    reso->display;
 }
};
```

Method bodies of the class School

```
method body init in class School {

  self->name = "";
};

method body title: string in class School
{  /* use School name as title */
  return self->name;
};

method body menu: list (string) in class School {
  return list ("display", "edit",
            "Add_Department", "Display_Departments");
};


method body edit: integer in class School {
  o2 extern set (string) Exist_Sch_Names;
  o2 set (School) schs;
  o2 School school;
  o2 School temp = new School;
  o2 string oldname = self->name;
o2 string sn;

  Lk_presentation scp, tmpp;

  int wval;

  temp = (o2 School) self->deep_copy;
  tmpp = schtpl (self);
  scp = schtpl (self);

  lk_map (tmpp, LK_COORDINATE, 0, LK_SCREEN, 200, 200);
  wval = lk_wait (tmpp);
  if (wval == LK_SAVE)
  /* user saves edits */
  {
    lk_consult (tmpp, temp);
    if (temp->name == "")
    {  /* School name is required */
      Dialoger->message("Name is required", "");
```

```
        lk_consult (scp, self);
        lk_delete_presentation (tmpp);
        return LK_ERASE;
      }


      for (sn in Exist_Sch_Names where sn == temp->name)
        if (temp->name != oldname)
        {  /* Duplicate Universities are not allowed */
          Dialoger->message("Duplicate name", "");

          lk_consult (scp, self);
          lk_delete_presentation (tmpp);
          return LK_ERASE;
        }
    }
    lk_consult (tmpp, self);

    Exist_Sch_Names -= set (oldname);
    Exist_Sch_Names += set (self->name);

    self->refresh_all;

    lk_delete_presentation (scp);
    lk_delete_presentation (tmpp);
    return wval;
  };

  method body Add_Department in class School {
    o2 Department department = new Department;
    o2 Department dept = new Department;
    o2 list(string) dept_names;
    Lk_presentation dpp;
    int wval;



    department->disable_method ("edit");
    department->disable_method ("display");
    department->disable_method ("Display_Faculty");
    department->disable_method ("Display_Students");
    department->disable_method ("Display_Projects");

    dpp = deptpl (department);
  lk_map (dpp, LK_COORDINATE, 0, LK_SCREEN, 200, 100);
  wval = lk_wait (dpp);

  if (wval == LK_SAVE)
  {  /* user saves edits */
    lk_consult (dpp, department);
    if (department->name == "")
    {  /* Depatment name is required */
      Dialoger->message("Name is required", "");
      lk_delete_presentation (dpp);
      return;
    }
    for (dept in self->Departments)
      /* get all Departments for this School */
      dept_names += list(dept->name);

    if ((department->name in dept_names) != -1)
    {  /* diplicate departments now allowed */
      Dialoger->message("Duplicate Department", "");
      lk_delete_presentation (dpp);
      return;
    }

    transaction;
    self->Departments += set(department);
    validate;
  }
  lk_delete_presentation (dpp);
};

method body Display_Departments in class School {

  o2 extern set (string) Exist_Dpt_Names;
  o2 string dept_name, edname;
  o2 list(string) dept_names;
  o2 Department dept = new Department;
  Lk_presentation dpp;
  int wval;

  for (dept in self->Departments) {
    /* get Departments for this School */
    dept_names += list (dept->name);
    Exist_Dpt_Names += set (dept->name);
  }
  if (count(dept_names))
    /* get user selection */
    dept_name = Dialoger->selection("Select a department", "", dept_names);
  else
  {
     Dialoger->message("No data found", "");
     return;
  }

  if (count(dept_name))
  {  /* find and display selected Department */
    for (dept in self->Departments)
      if (dept_name == dept->name)
        break;

    edname = "dpobj";
    dpp = ictpl (dept, edname);
    lk_map (dpp, LK_COORDINATE, 0, 0, 200, 100);
    wval = lk_wait (dpp);
    lk_delete_presentation (dpp);
  }
};
```

Sample Hyper Access Code

```
function build_gph_pres (gph: Sau_Graph): integer;

function body build_gph_pres (gph: Sau_Graph): integer {

  Lk_mask node_mask = lk_object ("nilmenu", 0, 0, 0);
  Lk_presentation gphp;
  Lk_resource res_graph[1];

  o2 GraphDialoger gd = new GraphDialoger;
  char objectstring [16];

  strcpy (objectstring, gd->initialize);
  gph->dialoger = gd;

  res_graph[0].name = "dialoger";
  res_graph[0].value = (char *) malloc (strlen (objectstring) + 1);
  strcpy (res_graph[0].value, objectstring);
  gd->set_trigger_event (0, SauIndex, "expand_node");
/*
  gd->set_trigger_event (1, SauIndex, "select_a_link");
  gd->set_trigger_event (2, SauIndex, "paste_a_node");
  gd->set_trigger_event (3, SauIndex, "delete_a_node");
  gd->set_trigger_event (6, SauIndex, "pre_insert_link");
*/
  gd->set_trigger_event (4, SauIndex, "post_insert_link");
  gd->set_trigger_event (5, SauIndex, "add_a_node");

printf ("after triggers\n");
  gphp = lk_present (gph, lk_object (0, 0, 0,
                lk_specific ("uvgraph", 1, res_graph, "graph",
                        1, &node_mask)));
printf ("after lk_present\n");
  return gphp;
};
```

```
/************/
function build_links (root: Sau_Node) : Sau_Graph;

function body build_links (root: Sau_Node) : Sau_Graph {

  o2 University uv = new University;
  o2 Resource rs; o2 set (Resource) rss;
  o2 School sc = new School; o2 set (School) scs;
  o2 Department dp = new Department; o2 set (Department) dps;
  o2 Research rc; o2 set (Research) rcs;
  o2 Faculty fc; o2 set (Faculty) fcs;
  o2 Student st; o2 set (Student) sts;

  o2 Sau_Graph gph = new Sau_Graph;

  gph->nodes = set (root);


  if (root->content->class_of == uv->class_of) {
    o2query (scs, "element (select x.Schools from x in Universities\
                    where x == $1)", root->content);

printf ("in build_links\n");
    for (sc in scs) {
      o2 Sau_Node nsc = new Sau_Node;
      o2 Sau_Link uv_sc = new Sau_Link;

printf ("in build_links\n");
      nsc->content = sc;

      uv_sc->drawing = list();
      uv_sc->label = "school";
      uv_sc->from = root;
      uv_sc->to = nsc;

      gph->nodes += set (nsc);
        gph->links += set (uv_sc);
    }
  }

  if (root->content->class_of == sc->class_of) {
    SchoolEntry = (o2 School) root->content;
    o2query (dps, "select x from x in SchoolEntry.Departments");
printf ("after query\n");
    for (dp in dps) {
      o2 Sau_Node ndp = new Sau_Node;
      o2 Sau_Link sc_dp = new Sau_Link;

      ndp->content = dp;

      sc_dp->drawing = list();
      sc_dp->label = "dept";
      sc_dp->from = root;
      sc_dp->to = ndp;

      gph->nodes += set (ndp);
         gph->links += set (sc_dp);
    }
  }

  if (root->content->class_of == dp->class_of) {
    DepartmentEntry = (o2 Department) root->content;
    o2query (sts, "select x from x in DepartmentEntry.Students");

    for (st in sts) {
      o2 Sau_Node nst = new Sau_Node;
      o2 Sau_Link dp_st = new Sau_Link;

      nst->content = st;

      dp_st->drawing = list();
      dp_st->label = "school";
      dp_st->from = root;
      dp_st->to = nst;

      gph->nodes += set (nst);
         gph->links += set (dp_st);
    }
  }

  gph->origin = tuple (x: 0, y: 0);
  gph->real_size = tuple (width: 300, height: 400);
  gph->screen_size = tuple (width: 700, height: 700);

  return gph;
};
```

REFERENCES

Bruza, P. D., and Th. P. van der Weide. 1990. Two Level Hypermedia - An Improved Architecture for Hypertext. Database and Expert Systems Applications: Proceedings of the International Conference in Vienna, Austria, 1990.

Conklin, Jeff. Hypertext: an Introduction and Survey. IEEE Computer.

Delisle, Norman, and Mayer Schwartz. 1986. Neptune: a Hypertext System for CAD Applications. 1986 ACM.

Faloustos, Christos, Raymond Lee, Catherine Plaisant, and Ben Shneiderman. 1990. Incorporating String Search in a Hypertext System: User Interface and Signature File Design Issues. Hypermedia. Vol. 2, No. 3, 1990. Taylor Graham Publishing, London, UK.

Gallagher, Leonard, Richard Futura, and P. David Stotts. 1990. Hypermedia. Increasing the Power of Hypertext Search with Relational Queries. Vol 2 No1 1990.

O2 Technology. 1993. The O2 User Manual. Version 4.4. Released December 1993.

Rambaugh, James, Michal Blaha, William Premerlan, Frederick Eddy, and William Lorensen. 1991. Object-Oriented Modeling and Design. Prentice Hall, Englewoods Cliffs, NJ 07632. 1991.

Schatz, Bruce R., and Michael A. Caplinger. 1989. Searching in a Hyperlibrary. 1989 IEEE.

Sowa, John F.

1993a. Relating Diagrams to Logic. Conceptual Graphs for Knowledge Representation: First International Conference on Conceptual Structures, ICCS'93, Quebec City, Canada, August 4-7, 1993.

1993b. Logical Foundations for representing object-oriented systems, Journal of

Experimental and Theoretical Artificial Intelligence (JETAI).

Tanaka, Katsumi, and Qing Qiang. 1992. Two-Level Schemata and Generalized Links for Hypertext Database Models. Proceedings of the 2nd Far East Workshop on Future Database Systems, Kyoto Japan, April 26-28, 1992.

Tompa, Frank W. M. 1989. A Data Model for Flexible Hypertext Systems. ACM Transactions on Information Systems, 7(1):85-100, January 1989.

Wang, B., and P. Hitchcock. 1991. *Intersect*: A General Purpose Hypertext System Based on an Object Oriented Database". Datatabase and Expert Systems Applications: Proceedings of the International Conference in Berlin, Federal Republic Of Germany, 1991.