A Formal Dynamic Schema Evolution Model for Hypermedia Databases

Bing Wang
Computer Science Department
University of Hull
Hull, HU6 7RX
United Kingdom
E-Mail: B.Wang@dcs.hull.ac.uk

Abstract

A hypermedia database schema is a meta-information which defines the structures of applications and objects. A dynamic schema evolution model is an abstract machine which describes changes of schemata. Unfortunately, because of the lack of formal data structures of hypermedia databases, it is very difficult to describe the dynamic changes of a hypermedia schema. The research presented in this paper proposes a formal approach to define the model of dynamic schema information. That is, a complete formal structure of a hypermedia database on the basis of previous research[WH93, WH95, Wan96, Wan97], and the schema operations defined for the schema evolution. The significant features of this research are, firstly, a formalized model to define the dynamic schema structure, secondly, a theoretical approach to manage the global schema of a distributed system, and finally, a design strategy to integrate the Web databases to support a wide range of application areas.

Keyword Formal Data Model, Multimedia, Dynamic Schema, Distributed Databases and Object Orientation.

1 Introduction

When a hypermedia application involves a large amounts of different types of multimedia information, it becomes more and more difficult to maintain the complex object types due to the lack of a powerful mechanism to define and maintain the structure of nodes and links. Consequently, a well-known problem in hypermedia, *the disorientation of hypermedia navigation*, appeared[Hal88, Hal91]. The essence of the disorientation is that the multimedia information types are unstructurally organized. The solution to this problem relies on the way of how to store information and how to describe relationships among nodes. Databases can be naturally chosen as the most suitable candidates for supporting complex hypermedia applications. This is because databases themselves are well structured for modelling a complex application structures. Since 90's, the author has focused on the data structure of linking databases with hypermedia to support complex application environments[HW92, Wan93, WH93, WH95]. Several enhanced hypermedia data models are developed[Wan96, Wan97]. In this paper, a further study of the dynamic hypermedia schema based on an extended hypermedia model[WH95] is discussed. In particular, we will illustrate the mechanisms of how to maintain the semantics of hypermedia application schemata in order to keep the consistency the node and link structures.

This paper is organized as follows. Section 2 reviews the semantic constructors of an extended hypermedia data model in order to illustrate the hypermedia schema structure which is described in section 3. Section 4 describes some related works and we conclude our discussion in section 5.

2 A Hypermedia Model

The dynamic hypermedia schema evolution model is set up on the basis of an existing extended hypermedia data model which is an object oriented data model capturing both semantics of hypermedia and object databases.[1] There three basic constructors in this model. They are *node constructor, composite node constructor* and *schema constructor.* These constructors are used as essential semantic building blocks for constructing a hypermedia application structure on the basis of object oriented databases. In the following review of this model, we will focus on the semantic aspects of these constructors and show how these constructors can formally present a hypermedia application.[2]

2.1 Relationships

The main problem of the current hypermedia systems to model an application structure is that they lack a powerful facility to reveal relationships among node. Thus, our extended hypermedia model focus on this aspect and reveal how the node structure can easily capture those semantic relationships among nodes.

In the conventional hypermedia approach, components of an application are stored in nodes and the nodes are connected by links. Because links are used simply to represent a fact that there exists a connection among nodes. It is very difficult for both designers and users to know the exact semantic relationship between nodes. Thus, it depends heavily on the human beings abilities to reveal relevant information behind nodes and links. It is therefore useful if a hypermedia node can automatically represent some semantic meanings of application structures.

There are two major relationship types between hypermedia components. They are part-of and inheritance relationships. In the extended hypermedia data model we developed, we define two node constructors to reveal such internal structure of a node.

Component Node Constructor

This is the first node constructor which reveals the part-of relationship between nodes. In general, a conventional hypermedia node is constructed by multimedia information. The multimedia data types within a node can be used as a either the destination or target of links. Thus, a node structure can be described by figure 1.

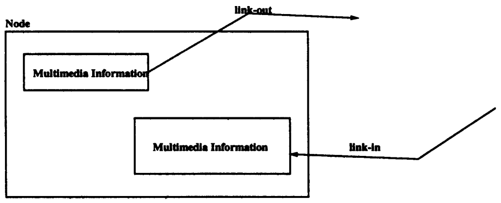

Figure 1: A conventional node structure

In the conventional approach, we cannot find out the exact semantic relationship between the node and its multimedia components. The node is a virtual object. It is only used as a concept to describe an information container. However, if we impose the part-of relationship between components and its container, we can modify the above picture into a more semantic node type as described in figure 2.

[1]This is a unified data model called HyperDB which was developed by the author in 1997[Wan97].

[2]A formal specification language Z is used to describe these constructors. Z is the formal specification language developed by the Oxford University in the late seventies. Now, it is widely used both in industry and academia.

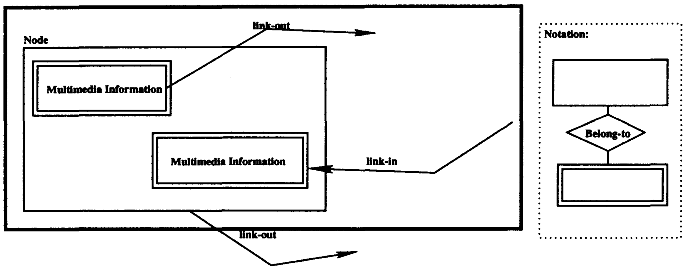

Figure 2: A semantic node structure

In figure 2, a significant difference between the conventional node (figure 1) and the new node is that a component is explicitly expressed as the *part* of a node. The node itself can be used not only as a container to group components but also a component for another node. The following Z schema exactly represents such semantic meaning.

$$
\begin{array}{|l}
\textit{Composite\_Node\_Constructor} \\
\hline
\textit{Node\_Constructor} \\
\textit{composite\_parts} : \mathbb{P}\, \textit{Multimedia} \\
\hline
\exists\, n : \textit{Node} \mid n = \textit{node} \bullet \textit{composite\_parts} = n.\textit{source}.\textit{media\_part} \cup n.\textit{destination}.\textit{media\_part} \\
\hline
\end{array}
$$

By using the above definition, a conventional hypermedia node is a special case of our definition. More importantly, using the Z schema definition to capture semantics of the node structure, we can further define the dynamic schema evolution model which can automatically maintain consistency of nodes and links.

Inheritance Node Structure

The inheritance property is the second important characteristic of existing applications. It reveals potential links between nodes. Most importantly, it organizes nodes in a structural way. From software engineering point of view, it reuses the sources of some software components. In the hypermedia design point of view, it imposes a semantic constraint between nodes. That is, the node **A** is a *super node* of the node **B**. By defining such semantic constraint among nodes, it guarantees that semantic operations such as *delete nodes, change nodes* and *create nodes* can be correctly executed when a defined application structure is evolved. The following picture illustrates the inheritance relationship between nodes.

The inheritance property defined above is different from that of object oriented model. An inheritance node inherits not only attributes of its super objects but also link structures defined within its super nodes. The following Z schema defines such scenario.

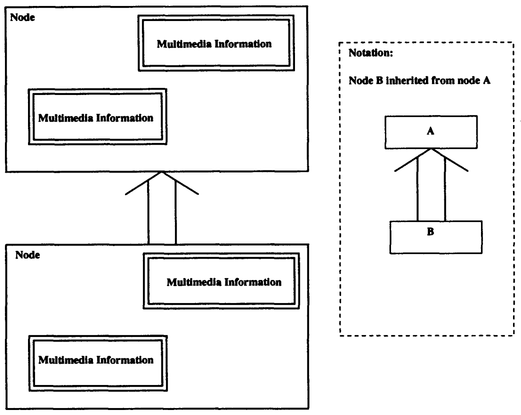

Figure 3: A semantic node structure

Node_Constructor

$$
\begin{array}{l}
node : Node \\
new\_media\_part : Components \\
new\_id' : ComponentID \\
super\_sub : \mathbb{P}(Components \times \mathbb{P}\ Components) \\
\hline
\operatorname{dom} super\_sub = 1 \\
\operatorname{ran} super\_sub = \{node.source.id\} \cup \{node.destination.id\} \\
\exists\, c : Components \mid c = new\_media\_part \bullet (c.media\_part = \{\} \Rightarrow \\
(c.id = node.source.id \lor c.id = node.destination.id) \land \\
(new\_media\_part = (c.media\_part \cup node.source.media\_part) \lor \\
new\_media\_part = (c.media\_part \cup node.destination.media\_part))) \\
\lor \\
(c.media\_part \neq \{\} \Rightarrow new\_media\_part.media\_part = c.media\_part \land \\
new\_id' = new\_media\_part.id \cup ComponentID)
\end{array}
$$

2.2 Schema

A hypermedia application is constructed on the basis of the above node constructors. The application is abstracted by the general purpose schema definition in the proposed approach. This is defined as follows.

Scheme_Constructor

$Node\_Constructor$
$Composite\_Node\_Constructor$
$node\_object : Node\_Type \nrightarrow ObjectType$

$\forall\, nodes : \mathbb{P}\, Node\_Type \bullet \forall\, n : nodes \bullet \exists\, o : ObjectType;\ N_1 : Node\_Constructor;$
$N_2 : Composite\_Node\_Constructor \mid n = InheritableNode\ N_1 \lor n = CompositeNode\ N_2$
$\bullet\ o = node\_object\ n$

The above schema specifies that all defined nodes must belong to the existing node types that have the corresponding object types supported by a database system. Since the purpose of the extended hypermedia data model is to use database facilities to model a hypermedia application structure, the above schema constructor provides users both database's and hypermedia's view to analyse and model an application structure. Figure 4 illustrates the relationships between a hypermedia node and a database schema.

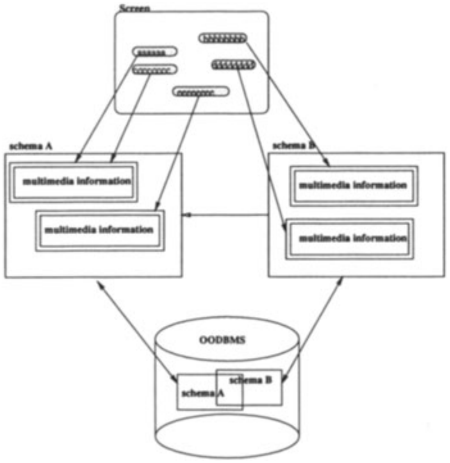

Figure 4: The relationships between nodes and schema

The schema of the extended hypermedia data model plays a role of structurally organize hypermedia information. The database is used as object store to maintain multimedia information. Thus, in order to get more semantics of an application schema, we need to define and use schema operators to combine or integrate different schemata which will discussed in the next section.

3 The Schema Operators

The dynamic evolution of a hypermedia database schema is a facility of managing hypermedia schema information when the hypermedia database is in operation. When a hypermedia application structure is defined, its semantic components and relationships among objects are automatically captured by the database. The dynamic schema evolution facility will use the information captured by the database to define a new application structure on the basis of the existing one. In order to formally describe such evolution, it is important to formalize a schema structure at the first stage as we have done in the previous section. In the second stage, what we need to do is to define schema operators which are essential for the dynamic schema evolution.

We have four basic operators for the schema evolution. In this paper, due to the limited length of the paper, we omit the discussion of the schema refinement. A complete description of the schema refinement will appear in the paper entitled *A dynamic Schema Structure for a Hypermedia Database.*[3]

3.1 Creating a Hypermedia Node

A node cannot be created without any semantic meanings. This means a hypermedia node can only be created as either a composite or an inheritance node. This is the potential constraint defined in HyperDB. In order to avoid creating an isolated node, we need two different functions to describe how to create a hypermedia node in HyperDB.

$$\begin{array}{|l}
\mathit{Create\_Composite\_Node} : \mathbb{P}\ \mathit{Components} \nrightarrow \mathit{Composit\_Node\_Constructor} \\
\hline
\forall\, c : \mathbb{P}\ \mathit{Components};\ s : \mathit{Schema\_Constructor} \bullet \\
\exists\, \mathit{new\_composite\_node} : \mathit{Composite\_Node\_Constructor};\ \mathit{new\_id} : \mathit{ComponentID}; \\
s' : \mathit{Schema\_Constructor} \mid s'.\mathit{Composite\_Node\_Constructor}.\mathit{composite\_parts} = \\
\mathit{new\_composite\_node}.\mathit{composite\_parts} \wedge c.\mathit{id} = \mathit{new\_id} \\
\wedge\ \mathit{new\_composite\_node} = s.\mathit{Node\_Constructor} \vee s.\mathit{Composite\_Node\_Constructor}
\end{array}$$

In the above function, it guarantees that whenever a node is created its components must be the one of existing nodes. This meets the original definition of the composite node constructor. The partial function is defined to establish the linkage between components and composite node type. This is a typical mathematic approach of connecting two object sets in Z specification. Its main purpose is to explicitly express the output of an action. More importantly, this function can be used to construct and refine a schema. In this definition, there are several pre-defined given sets. A given set is a mechanism used in Z in order to ignore detailed description of a data type structure. By using this facility, we can focus on the fundamental design issue of a model. The ComponentID is a given set which means the structure of the component identifier is ignored. The Node_Type is a free type structure defined in [Wan96]. This definition allows us to generalize the node type structure in our model. The second given set used in the above function is ObjectType which describes all possible object types supported by an object oriented. The similar definition of creating a inheritance node is defined as follows.

$$\begin{array}{|l}
\mathit{Create\_Inheritance\_Node} : \mathit{Node\_Type} \nrightarrow \mathit{Node\_Type} \\
\hline
\forall\, n : \mathit{Node\_Constructor};\ c : \mathit{Composite\_Node\_Constructor};\ s : \mathit{Schema\_Constructor} \bullet \\
\exists\, \mathit{new\_inheritance\_node} : \mathit{Node\_Type};\ \mathit{new\_obj} : \mathit{ObjectType};\ s' : \mathit{Schema}\ {}_{C}\mathit{onstructor} \bullet \\
s'.\mathit{node\_object} = s.\mathit{node\_object} \cup \{\mathit{new\_inheritance\_node} \mapsto \mathit{new\_obj}\} \\
\Rightarrow \mathit{new\_inheritance\_node} = \mathit{CompositeNode}\ c \vee \mathit{new\_inheritance\_node} = \mathit{InheritableNode}\ n
\end{array}$$

The above function specifies that an inheritance node is such a node that is generated from the existing node, which is a node with the type of either Composite_Node_Constructor or Node_Constructor.

By using these two function definitions, the first schema operator of creating a hypermedia node which has the semantics defined in HyperDB is defined as follows:

[3]A paper submitted to th Journal of Information Science

$$\begin{array}{|l}
\textit{Creating\_Node} \\
\Delta \textit{Schema\_Constructor} \\
\textit{node?} : \textit{Node\_Type} \\
\textit{components?} : \mathbb{P}\ \textit{Components} \\
\hline
\forall\ \textit{new\_node} : \textit{Node\_Type} \bullet \textit{new\_node} = \textit{Create\_Inheritance\_Node}\ \textit{node?} \lor \\
\textit{new\_node} = \textit{Create\_Composite\_Node}\ \textit{components?} \\
\hline
\end{array}$$

3.2 Finding a Hypermedia Node

Before a schema is evolved, we need to know if a node is existed. This means that we need to find out a specific node according to our needs. First, we specify the existence status of a node. A set is defined to represent two different status.

$$\textit{Existence\_Status} ::= \textit{Existence} \mid \textit{Nonexistence}$$

On the basis of this set, a schema operator to indicate if a node is existed is defined as follows:

$$\begin{array}{|l}
\textit{Report\_Node\_Status} \\
\textit{node\_status!} : \textit{Existence\_Status} \\
\hline
\textit{node\_status} = \textit{Existence} \\
\hline
\end{array}$$

Furthermore, we define a schema operator to check if a node is absolutely existed in the information basis. This is:

$$\begin{array}{|l}
\textit{Examining\_Node} \\
\textit{Schema\_Constructor} \\
\textit{found\_node!} : \textit{Node\_Type} \\
\hline
\textit{found\_node!} \in \operatorname{dom} \textit{node\_object} \\
\exists\ \textit{I\_node} : \textit{Node\_Constructor};\ \textit{c\_node} : \textit{Composite\_Node\_Constructor} \bullet \\
\textit{found\_node!} = \textit{InheritableNode}\ \textit{I\_node} \lor \textit{found\_node!} = \textit{CompositeNode}\ \textit{c\_node} \\
\hline
\end{array}$$

The above schema operator guarantees that the found node must be a node defined in the corresponding application structure and it is a valid node by imposed the node type constructor (Node_Type)[Wan96].

In a more general situation, an evolution of a hypermedia application schema of successfully finding a node can be described as:

$$\textit{Find\_Hypermedia\_Node} \mathrel{\widehat{=}} \textit{Examining\_Node} \land \textit{Report\_Node\_Status}$$

In this schema operator definition, we have shown the simple way of how to define a schema evolution by using the first order logic connectives. However, in most object oriented schema evolution approaches, there are many complex operators are defined in order to check or change an existing object. The advantage of our approach is that we rely only on the first order logic to connect schema and to infer a new schema operator such as Find_a_Hypermedia_Node defined above. That is why we only need to define four basic schema operators in our approach.

3.3 Add Operator

The third general schema operator in the hypermedia schema evolution is the add operator. When a schema is changed, it implies that either a new node is inserted into the new application structure or a node is deleted from the existing application structure in order to keep semantic consistency between node types defined in the new application structure. Thus, we need to consider two different situation. They are, adding a node according to the need and delete a node according to the requirements. We first discuss how to add a node when a schema is changed. Next section will illustrate the situation of deletion.

- *A complete new node is inserted*

  In this situation, a new node is created without any input information such as Creating_Node schema defined above. This is because a node is created due to the specific requirements of a application structure. Such schema operator is defined as follows.

  $$\begin{array}{|l}
  \textit{Insert\_New\_Node} \\
  \hline
  \Delta \textit{Schema\_Constructor} \\
  \hline
  \forall\ \textit{a\_new\_node}, n : \textit{Node\_Type} \bullet \exists\ c : \mathbb{P}\ \textit{Components} \bullet \\
  \textit{a\_new\_node} = \textit{Create\_Composite\_Node}\ c \\
  \vee\ \textit{a\_new\_node} = \textit{Create\_Inheritance\_Node}\ n \\
  \hline
  \end{array}$$

  This schema says that all newly inserted nodes are automatically inserted into the new application schema without any input information. That means, if it a composite node, a corresponding components are generated. Otherwise, a inheritance node is generated.

- *A new node is inserted on the basis of the existing one*

  In the second situation, the new generated node is based on an existing node. This means that the new node will inherit all link structures defined in the old node. We re-define link definitions defined in HyperDB in order to define this schema. All the predicate parts of link schema remain the same, but they are defined as generic definition.

  1. *Navigation Link Type*
     A navigation link type is a functional link to link instances of a node type. In the HyperDB approach, an instance link is a non-semantic link. That means it only physically links two instances. This is just a browsing function defined on the source and the target of nodes. Its generic form is defined as follows.

     $$\begin{array}{|l}
     \textit{Navigation\_Link}[M_1, M_2] \\
     \hline
     \textit{target}_1, \textit{target}_2 : \textit{Node\_Type} \\
     \textit{navigation\_link} : \mathbb{P}(M_1 \times M_2) \\
     \hline
     \forall\ \textit{nodes} : \mathbb{P}\ \textit{Node\_Type} \bullet \exists\ n_1, n_2 : \textit{nodes};\ N_1 : \textit{Node\_Constructor}; \\
     N_2 : \textit{Composite\_Node\_Constructor} \mid n_1 = \textit{InheritableNode}\ N_1 \vee \\
     n_1 = \textit{CompositeNode}\ N_2 \vee n_2 = \textit{InheritableNode}\ N_1 \vee \\
     n_2 = \textit{CompositeNode}\ N_2 \bullet N_1.\textit{node.source.media\_part} = \textit{target}_1 \wedge \\
     N_2.\textit{node.destination.media\_part} = \textit{target}_2 \\
     \Rightarrow \textit{target}_1 = \textit{navigation\_link}\ \textit{target}_2 \\
     \hline
     \end{array}$$

     This schema says that whatever nodes they are, a navigation link is a connection between the source and the target of a node.

  2. *Connection Type*
     A connection between node types is a high level relationship type which specifies a user's defined semantics between ᵐc le types. The *generalization/specialization*

and *part-of* relationships are already revealed by the definitions of node constructors themselves. However, it is difficult to represent the following semantics between two entities:

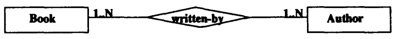

Figure 5: A *written-by* relationship between two entities

We need a specified relationship exactly represent a semantic meaning between two node types. This is achieved by the connection type defined by the following schema.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Node\_Type\_Connection}[\textit{Node}_1, \textit{Node}_2]} \\
\hline
\textit{connection} : \mathbb{P}(\textit{Node}_1 \times \textit{Node}_2) \\
\hline
\forall\, \textit{node\_types} : \mathbb{P}\, \textit{Node\_Type} \bullet \exists\, n_1, n_2 : \textit{node\_types};\ N_1 : \textit{Node\_Constructor}; \\
N_2 : \textit{Composite\_Node\_Constructor} \mid n_1 \neq \textit{InheritableNode}\ N_1 \lor \\
n_2 \neq \textit{CompositeNode}\ N_2 \bullet n_1 \mapsto n_2 \\
\hline
\end{array}$$

The above schema expresses that if two node types are not inheritable and composite node types, there must be a named semantic relationship defined between them. This further implies that no isolated node types exist in the hyper-space. Thus, a complete schema operator of adding a new node on the basis of existing nodes can be defined as follows.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Insert\_Based\_Node}} \\
\hline
\textit{nodes} : \textit{Node\_Type} \rightarrowtail\!\!\!\!\rightarrow \textit{ObjectType} \\
\textit{link}_1 : \textit{Navigation\_Link}_{[\textit{Multimedia},\textit{Multimedia}]} \rightarrowtail\!\!\!\!\rightarrow \textit{ObjectType} \\
\textit{link}_2 : \textit{Node\_Type\_Connection}_{[\textit{Node\_Type},\textit{Node\_Type}]} \rightarrowtail\!\!\!\!\rightarrow \textit{ObjectType} \\
\hline
\operatorname{dom} \textit{nodes} \in \mathbb{P}\, \textit{Node\_Type} \\
\operatorname{dom} \textit{link}_1 \in \mathbb{P}\, \textit{Navigation\_Link}_{[\textit{Multimedia},\textit{Multimedia}]} \\
\operatorname{dom} \textit{link}_2 \in \mathbb{P}\, \textit{Node\_Type\_Connection}_{[\textit{Node\_Type},\textit{Node\_Type}]} \\
\langle \operatorname{ran} \textit{nodes}, \operatorname{ran} \textit{link}_1, \operatorname{ran} \textit{link}_2 \rangle\ \mathsf{partition}\ \textit{ObjectType} \\
\hline
\end{array}$$

Finally, as we illustrate before, we use the basic logical connectives to describe the add schema operator as follows.

$\textit{Add\_Node} \mathrel{\widehat{=}} \textit{Insert\_New\_Node} \lor \textit{Insert\_Based\_Node}$

3.4 Delete Operator

This the final schema operator for evolution purpose. It is a complex issue to delete a node. This is because a deleted node may have relationships with other nodes. In most cases, the deletion operation is executed on the basis of the assumption that all the related nodes are automatically deleted as well. The following delete operator schema expresses such situation.

$$\begin{array}{|l}
\textit{Delete\_Node} \\
\hline
\Delta \textit{Schema\_Constructor} \\
\textit{Examining\_Node} \\
\textit{Report\_Node\_Statusdeleted\_node?} : \textit{Node\_Type} \\
\hline
\forall s : \textit{Schema\_Constructor} \bullet \\
\exists \textit{results} : \textit{Node\_Type};\ \textit{relevantnodes} : \mathbb{P}\,\textit{Node\_Type};\ s' : \textit{Schema\_Constructor} \mid \\
\textit{results} = \textit{found\_nodes} \land \\
\textit{results} = \textit{InheritableNode deleted\_node?} \cup \textit{CompositeNode deleted\_node?} \land \\
\textit{node}' = (\textit{relevantnodes} \cup \textit{results}) \lhd \textit{node} \\
\hline
\end{array}$$

This schema says that, firstly, if a node will be deleted, we find out all its relevant nodes; secondly, all the relevant nodes are automatically deleted without any conditions.

By using the above schema operators, can finally describe how a a new hypermedia application schema is defined by the following definition.

$$\textit{Define\_a\_New\_Schema} \mathrel{\widehat{=}} \textit{Creating\_Node} \lor (\textit{Finding\_Hypermedia\_Node} \land \textit{Add\_Node}) \lor (\textit{Delete\_Node} \land \textit{Add\_Node})$$

This simple algorithm illustrates a complex situation of the schema evolution. It abstracts a complex evolution strategy and provides a simple way to modify an existing application structure. This is the main advantage of this research.

4 Related Work

Orion[JKso87] is the earliest research of using invariants and rules to structurally organize the database schema information. A invariant is a constraint to maintain schemata consistency and rules are used to control invariants if a schema is changed. The schema propagation of Orion is through screening. A similar approach is used in the GemStone system design[PS87]. In later 80's, Zdonik proposed a framework for object version type[SZ86]. In this approach, their focus is on the object types' propagation. The schema operations are still similar to Orion.

In general, there are three methods used to deal with semantics of schema changes. They are screening, conversion and filtering. In the screen approach, a conversion program is automatically generated when evolving a database schema. Old object types are monitored and converted into the new types of new or changed schema. In the conversion approach, old object types are immediately changed according to the new schema definition. Finally, in the filtering approach, a set of new definitions are defined as filters to deal with the consistency between old and new object types, that is, object types belonging to the old schema are becoming a specific version types of the defined schema. Orion, GemStone and Zdonik's approaches use the combination of these techniques to explicitly enforce objects to coincide with the new definition of the schema. Because there is not a universal definition of object databases, it is difficult to say which technique is best. On the other hand, since the hypermedia node types are organized by using links, it is impossible to use the above techniques to coerce nodes to coincide with the new or updated application structures. The reason is that node types are pre-connected by unstructured links. Thus, when we use a general purpose object oriented database as the information base for supporting hypermedia, it is very difficult for us to keep consistency of both links and node types. The proposed research avoids the complex definition of hypermedia application structures by introducing two basic node types. These two node types plays two roles. First, they are the only existing object types. This means no other isolated node types will be allowed to exist. Secondly, they are used as abstract templets to represent all possible objects. Thus, it simplifies the complex definition of application structures. More importantly, this allows us to use limited schema operators to specify all the possible change among a schema.

Formal methods are currently being increasingly introduced and frequently y used in hypertext research. When compared with other research areas, studies on the formal specification of hypertext structures only began a few years ago. From 1988 onwards, several research results have been published. The earliest person to use mathematical methods to define formally the structure of a hypertext system was Garg[Gar89], The important contribution of Garg's work is the definition of the abstraction mechanisms — *aggregation* and *generalization* — which were constructed on the basis of the mathematical definitions of nodes and links. Garg's work shows that we are able to define the hypertext structure using set theory and first-order logic. In the 90's, formal specification languages such as VDM and Z have been introduced into hypertext research. Lange[Lan90] used VDM to define the hypertext structure. His work experiments with the use of an OODBMS to build a hypertext system, and he formally defines his hypertext data model in an object oriented way. Halasz and Schwartz[HM90] used Z to formalize the Dexter hypertext reference model. The Dexter model divides a hypertext system into three layers, these being the runtime, storage and within-component layers[HM90]. Dexter's specification focuses mainly on the storage layer, which is used to model the basic hypertext node/link structure. Dexter model defines hypertext operations clearly and unambiguously, and guarantees addressing for any hypertext component. It is also beneficial to us when defining operations, constraints and relationship types. From these approaches, we see that the formal specification of hypertext structures can provide the foundation for understanding the essentials of hypertext systems.

5 Conclusion

The dynamic schema evolution of a hypermedia database is a new research area in the hymerpedia area. The importance of this research is to reveal formally the linkage between a database and hypermedia, and in particular the schema facilities of a database to maintain complex relationships among application schemata. The significant contribution of this research are, firstly, it is the first time of using formal notations to describe a hypermedia schema structure and corresponding schema operations; secondly, it provides a sound basis of how to use object oriented databases to integrate hypermedia and multimedia; and finally, it offers a mechanism of how to automatically support schema generation and maintenance. Furthermore, the research presented in this paper can be used as one of fundamental data structures for integrating Web with object oriented databases. We need a global supporting tool to control and maintain thousands of Web information. The proposed data model meets such requirements and it can be used to precisely define the data structure of the new generation Web systems. To explore the potential power of this data model, a research project entitled the design of Web meta-information search engine is being done by the database research team at the University of Hull, U.K.

References

[Gar89] P. K. Garg. *Information Management in Software Engineering: A Hypertext Based Approach.* PhD thesis, Computer Science Department, University of Southern California, USA, 1989.

[Hal88] F. G. Halasz. Reflections on notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 7(7):836–862, July 1988.

[Hal91] F. G. Halasz. Seven issues: Revisited. *Hypertext'91 Keynote Talk*, December 18, 1991.

[HM90] F. G. Halasz and S. Maye. The dexter hypertext reference model. *Proceedings of the Hypertext Standardization Workshop*, pages 95–131, January 16-18, 1990.

[HW92] P. Hitchcock and B. Wang. Formal approach to hypertext system based on object oriented database systyem. *Information and Software Technology*, 34(9):573–592, September 1992.

[JKso87] J.Banerjee, W. Kim, and so on. Semantics and implementation of schema evolution in object oriented databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 311–322, San Francisco, May 1987.

[Lan90] D B Lange. A formal approach to hypertext using post-prototype formal specification. *Lecture Notes in Computer Science*, (428):99–121, April 17-21 1990.

[PS87] D.J. Penney and J. Stein. Class modification in the gemstone object oriented dbms. In *Proceedings of the International Conference on Object Oriented Programming: System, Language and Applications*, pages 111–117, Orland, FL, October 1987.

[SZ86] A.H. Skarra and S.B. Zdonik. The management of changing types in an object oriented database. In *Proceedings of the International Conference on Object Oriented Programming: System, Language and Applications*, pages 483–495, Portland, OR, September 1986.

[Wan93] B Wang. *Integrating Database and Hypertext to Support Documentation Environments.* PhD thesis, Computer Science Department, University of York, Heslington, York, Y01 5DD, U.K, 1993.

[Wan96] B. Wang. The design of an integrated information system. In Roland R. Wagner and Helmut Thomas, editors, *Database and Expert System Applications (DEXA'96), Lecture Notes in Computer Science No. 685*, pages 479–488, Zurich, Switzerland, September 1996. Springer-Verlag.

[Wan97] B. Wang. Toward a unified data model for large hypermedia applications. In Ab delkader Hameurlain and A Ming Tjoa, editors, *Database and Expert System Applications (DEXA'97), Lecture Notes in Computer Science No. 1308*, pages 142–152, Toulouse, France, September 1997. Springer-Verlag.

[WH93] B. Wang and P. Hitchcock. An object oriented database approach for supporting hypertext. In Colette Rolland, Francois Bodart, and Corine Cauvet, editors, *Advanced Information Systems Engineering (CAiSE'93), Lecture Notes in Computer Science No. 685*, pages 601–628, Paris, France, June 1993. Springer-Verlag.

[WH95] B. Wang and P. Hitchcock. *InterSect_DM*: a hypertext data model based on oodbms. *Information and Software Technology*, 37(3):573–592, March 1995.