Warehousing Structured and Unstructured Data for Data Mining

L.L. Miller and Vasant Honavar

Department of Computer Science
Iowa State University
Ames, IA 50011

Tom Barta

Dept. of Industrial Engineering
Iowa State University
Ames, IA 50011

ABSTRACT

More data, especially unstructured data, is available to users than ever. There is so much data available that it is difficult for users to make use of their data in its raw form. To handle the diversity of data types, we have designed and prototyped a multidatabase/warehouse system. The system has been especially designed to facilitate the interaction of structured and unstructured data. The system makes use of object oriented views. The main features of the view mechanism, especially as they relate to textual documents, are presented in the paper. The system is designed to take target documents either from large repositories or from the Web. Issues for both sources of documents are examined in the paper. The paper also looks at how the view approach allows the interaction between the data taken from structured (e.g., relational), semistructured (e.g., object oriented) and unstructured (e.g. text) data sources. The warehouse support provided by the system is briefly examined and the paper concludes by looking at our approach to data mining and how the system will operate in the complete environment.

INTRODUCTION

Today's computing environments are characterized by increasing heterogeneity, distribution, and cooperation. These characteristics increase the complexity of advanced applications that require integration and interoperation of heterogeneous and distributed hardware and software systems. On the other hand, the technology of modern computing industry has established the infrastructure to host complex applications that were implausible just a few years ago.

The topic of integrating data from multiple heterogeneous data sources has been extensively studied. However, the changing technology of the modern computing industry demands better and more realistic solutions. In this paper, we describe an extensible view system that supports integration of both structured and unstructured data sources in either the multidatabase or data warehouse environment.

The view system is an extension of the Zeus View System(ZVS) (Yen et al., 1994; Yen and Miller, 1995). The current view system has been extended to handle semistructured and unstructured data (e.g. text and images) in addition to structured data (e.g. relational databases). In addition changes have been made to enhance operations like composition of views.

The roots of our work are in the multidatabase environment. However, we see the notion of a data warehouse as being a natural outgrowth of a multidatabase system. Many of the underlying issues are the same. For example, the semantic mapping required to integrate data sources is a fundamental aspect of both environments. In our case, the notion of a warehouse became important as we looked into applying data mining to large collections of unstructured data (e.g. text and images). When we looked at the data warehouse literature for object oriented warehouses it was apparent that most of the existing work is aimed at either relational or dimensional models and would not be rich enough for our applications. Since we were interested in testing different data mining algorithms on the same collection of data, we needed an environment that was capable of creating and storing different "views" of the data set chosen for the tests.

It was clear that we would benefit from having the tools (in our case data mining tools) and the data in the same package.

The main contribution of this paper is the extension of the view system and the development of an object oriented warehouse based on the view system for applying data mining to unstructured data. The warehouse has been designed to allow both the search and the necessary transformations to be incorporated into the views that define the warehouse classes.

The paper has the following structure. The next section looks at some related work. The third section overviews our view structure. Section 4 looks at our approach to an object oriented warehouse and looks at our current prototype. Finally, Section 5 takes a look at our expectations for using the warehouse for data mining.

RELATED WORK

Warehouses and Multidatabases

Work on gaining access to data from heterogeneous sources can be found in both data warehouses and multidatabase literature. Both of the systems are designed to operate on heterogeneous data sources and provide a mapping of the data into a common structure. Over the years, there has been a great deal of work on multidatabases. A taxonomy of multidatabase systems is given in (Bright et al., 1992).

While data warehouses are becoming common place in industrial circles (Mattison, 1996), research literature on data warehouses has only recently appeared. The work has focused on using materialized views as a means of constructing the warehouse. The Stanford data warehouse project points out that object oriented views would be a useful way of accessing heterogeneous data sources (Hammer et al., 1995; Wiener et al., 1996), but their publications make use of the relational view. The majority of the published work has been focused on the self maintenance of materialized views (Baekgaard and Roussopoulos, 1997; Gupta, 1997; Huyn, 1996; Quass et al., 1996; Zhuge et al., 1995) and concurrency control (Kawaguchi et al., 1997a; Kawaguchi et al., 1997b).

Fundamental research in multidatabases includes modeling issues, mapping methodologies and semantic issues. The modeling issues center around the creation of a common data model for modeling integration and interoperation of multiple database systems. Sheth and Kalinichenko (1992) looked at information modeling issues in environments that range from multidatabase manipulations to multisystem applications. Barsalou et al.(1991) proposed an extensible meta level system in which the syntax and semantics of data models, schemas and databases can be uniformally represented. Su et al. (1993) proposed an object-oriented rule-based approach to providing a common data model that maps a local database schema to a global schema.

A great deal of activity has been focused directly on the task of developing multidatabase systems that allow transparent access to a variety of database environments. The work can be roughly divided into three approaches (Bright et al., 1992).

The first approach makes use of a global schema. It typically makes use of a common data model and a global language. The heterogeneous nature of the data is hidden from the user. Bright et al. (1992) list several approaches.

The second approach is the Multidatabase language approach. It places most of the burden on the users. Global users are aware of multiple data sources. Users make use of the common language to define how the data sources are integrated, transferred and presented.

The third approach is the use of federated databases. Each local database system maintains a partial global schema that contains only the global information used by local users. The local site works closely with a set of inter-related sites to set up the partial global schema. The set of inter-related sites form a federation. A formal definition of a federated database system can be found in (Heimbigner and McLeod, 1985).

Evolution of the View Concept

Due to the diversity of existing view concepts, it is hard to make a precise comparison between the various view concepts. However, we would like to give readers the idea of how view concepts have been used in different application domains and why we chose views as the basis of the $\mathcal{ZVS}$.

Conventional Views *The conventional notion of views is often used to describe derived data in order to achieve better protection and flexible access to shared information.* For example, relational views can be represented by non-procedural queries to describe derived data. Views have also been used in software engineering. For example, views in (Garlan, 1986) were used to let multiple tools share a common object base.

View Objects *In recent years, the view concept has been extended to utilize object-oriented techniques. This extension has been applied to a variety of application domains.* For example, the notion of view objects was introduced in (Wiederhold, 1986) to integrate the abstraction capabilities of programming languages and the conventional view concept of database systems. In (Premerlani et al., 1990), an Object-Oriented Relational DBMS was implemented and applied to a spectrum of applications. The view concept in (Shilling and Sweeney, 1989) is used to create an architectural building block for object-based software environments. In (Barsalou et al., 1991), object-based views are used to update relational databases. The issues of view update semantics are discussed and investigated in (Chen and McLeod, 1989; Scholl et al., 1991). *Numerous approaches have also emerged to encapsulate information and software systems by views.* For example, abstraction and view mapping capabilities are proposed in (Heiler and Zdonik, 1990) to support federation of heterogeneous software and databases. A sophisticated view mechanism is introduced in (Abiteboul and Bonner, 1991) to help restructure data or integrate databases. Object-oriented views are used to integrate heterogeneous information systems in (Czejdo and Taylor, 1992; Kaul et al., 1990).

AN EXTENSIBLE VIEW SYSTEM

We propose a view mechanism for integration of data from heterogeneous structured and unstructured data sources. In this section we discuss the view mechanism.

Overview

The view has abstract object semantics. It does not have a stored state nor does it provide any global resources. It is only used to describe available services and resources such that the information can be recalled by the view system to coordinate the sharing of the services and resources. We use an extensible object model (EOM) as an intermediate model to facilitate the representation of different data model constructs and application objects. We also extend the EOM to provide an integration model. The views are described by a view definition language (VDL) which provides portable syntax and semantics for view construction and sharing. The EOM and VDL create a uniform interface to hide the heterogeneity and distribution of participating systems. The view is created to define: how the local services and resources are exported, how the global services and resources are imported, or how the services and resources are constructed and presented. The view mechanism supports the creation and management of views, the processing of views and the coordination of the computations resulting from views.

An Extensible Object Model

We define an extensible object model (EOM) to provide a common framework for modeling real-world and conceptual entities in a variety of application domains. We adopt the terminology used in (OMG, 1992) to present the EOM. The EOM is composed of a core object model and a set of components. A component is a compatible extension of the core object model. A profile is a set of selected components. *A profile along with the core object model provide the modeling facility for a particular application domain.* Figure 2 shows the concepts of our EOM. The proposed core object model has been influenced by on-going standardization efforts (OMG, 1991; OMG, 1992) and research(Peters et al., 1992).

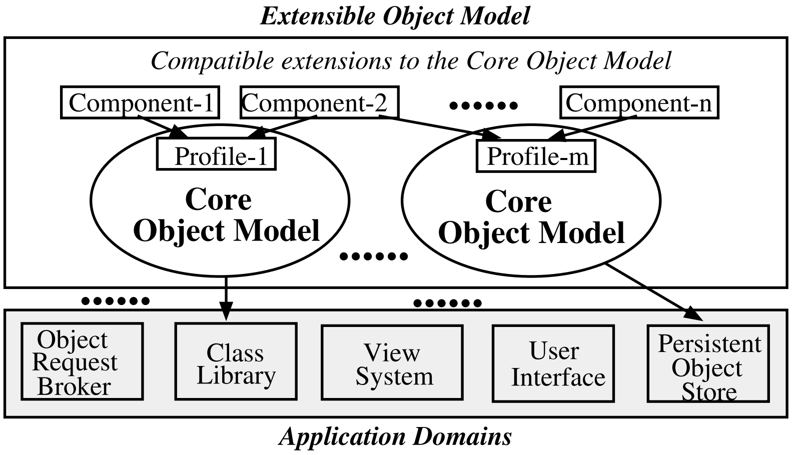

Figure 1: **The extensible object model (EOM).**

Object An object can model any real-world entity or conceptual entity. Each object has a unique identity which is immutable and persists as long as the object exists. An object has its state and behavior. The state of an object captures the information carried by the object. The behavioral semantics of an object are implemented by operations to change the state of the object.

Type Objects are instances of types. A type has a unique and immutable identity, an interface and a set of instances. The set of instances of a type is also called the extension of the type. The definition of the interface is also referred to as the type specification which defines the structure and behavior of the type.

Class A type specification may have several different implementations. The combination of a type specification and one of the implementations defined for that type is called a class.

Non-object Types Instances of non-object types are the atomic values like numbers, characters, strings, and so on. The set of non-object types is different in different application domains and can be specified in profiles. For example, Binary Large OBjects (BLOB) can be chosen as a non-object type for multimedia applications.

Type Hierarchy & Inheritance All types, except non-object types, form a type hierarchy. The growth of the type hierarchy is through subtyping. Subtyping creates the relationship between supertypes and subtypes. Inheritance is a mechanism for code reuse. The specification of subtypes can be inherited from supertypes through inheritance.

Type System & Object System We define a type system as a group of types which model a specific application domain.

The EOM is extensible in the sense that components can be added and then grouped by profiles to tailor to the requirements of existing and future application domains.

Mapping Methodology

A mapping methodology is required to ensure that data brought in from the data sources to the warehouse/multidatabase match the types needed for the warehouse/multidatabase applications. Supporting such a mapping is complicated by the continued growth of the number of new data types that must be considered. The mapping methodology must be able to handle data from files, various database management systems, document retrieval systems, as well as, other forms of semi or unstructured data. In addition it must be able to easily adapt to new data types. Our approach is to make use of the view object concept as the basis of the mapping methodology. We define the view object type as being an extension of the

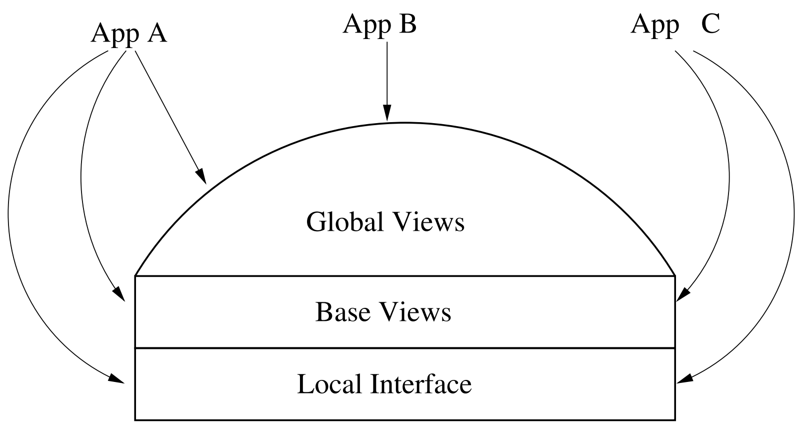

Figure 2: **Layered access to the three levels of views.**

object model (EOM). Our use of views is an extension of our work on the Zeus View Mechanism given in (Yen et al., 1994). The view type consists of public attributes, private attributes, a derivation section and a method section. The public attributes, private attributes, and method sections have the normal meanings. The derivations section is used to generate the public and private attributes of each object instance created through a view.

Three levels of views have proved useful in our system:

Local Interface The individual data sources are expected to have local control. The local interface is a view type object that is used by the local data administrator to provide a mechanism to make his/her data accessible to the warehouse/multidatabase environment. The local interface gives the local administrator the ability to present data as it is stored, or he/she can hide or convert(map) some of the data from the data source. To reduce the load on the local administrator, we assume that the primary functions will be simply providing access and hiding of sensitive data.

Base views The base view is used to do the necessary conversion of the data. The derivation section is used to provide the mechanism for both data selection and conversion. The base view can either be a user defined view or a system view. The only real difference is that the system base views are provided for expected conversions. Two examples of system base views are views that convert object instances to a set of relations and a a view that turns a set of textual data into a set of term vectors. As the set of system base views expands, user base views are only needed for new conversions.

Global views Whereas base views only make use of local interfaces, global views can be composed of either local interfaces, base views or other global views. The main reason for global views is to provide a higher degree of flexibility in creating different "views" of the data.

The composition algorithm allows global views to be based on any number of local interfaces, base views, or global views. For runtime simplicity, our current view system prototype maintains a table of precomposed views. It would be easy to incorporate the composition at runtime, but we have chosen to do the composition at the time the view is integrated into the system.

Figure 2 shows the layered access to the three view types. A given application can use any combination of the views.

Runtime Support

We have selected the Inter-Language Unification (ILU) package as the basis of our current prototype. ILU has been created by Xerox for distributed object management. While ILU is related to the CORBA (Common Object Request Broker Architecture), it is not CORBA compliant. An object request broker (ORB) provides the mechanisms by which objects transparently make requests and receive responses.

The runtime behavior of our views comes from a two-step mapping, view objects $\mapsto$ ILU objects $\mapsto$ programming-level objects. View objects are purely abstractions. ILU objects may encapsulate services, DBMSs, etc. Programming-level objects can be used in programming languages as program variables. The mappings between view objects and ILU objects are handled by the view system. The mappings between ILU objects and programming-level objects are either determined by the view system or applications. At runtime, application programs rely on programming-level objects to perform the computation. ILU objects have concrete runtime semantics which is provided in the forms of object invocation and method dispatching by object request brokers. View objects are purely high-level abstractions.

DATA WAREHOUSE

The material presented to this point can either be applied to a multidatabase system or to an object oriented warehouse. In this section we look at some of the issues that are particular to warehouses.

The warehouse in our approach consists of a set of classes where each class is a materialized object view. The resulting object classes are either virtual or imaginary classes using the terminology in (Abiteboul and Bonner, 1991). The views used to generate the materialized view can be at any level, i.e. local interfaces, base views or global views. The choice of view level depends on the amount of preprocessing and/or preselection that is being done to create the class for the warehouse.

Virtual classes are typically added to the warehouse by making direct use of the local interface. We denote these materialized views as virtual classes, since the local administrator has introduced the class to the system as a set of objects. Our usage of the virtual class is somewhat looser than that used by Abiteboul and Bonner (1991), since the actual data source may or may not be object oriented. For example, the actual data source may have been a set of text documents that the local data administrator mapped to the object data format.

Imaginary classes make use of either base views or global views. Global views are used to generate a materialized view, whenever it is necessary to use composition of views to generate the imaginary class. Base views are typically used to provide any desired mapping (preprocessing) or preselection of the data from the data sources.

There are several advantages to this approach. The main advantage of our approach over existing warehouse systems is the fact that the object format created by our view system has far richer semantics than current systems. We are able to make use of our views to provide different looks to the same set of data. This will be especially advantageous when we apply data mining tools to our warehouse. For example, with the view system it is possible to store a class that contains textual documents, while another class consists of weighted term vectors for the same set of documents. This means that we can use different tools to analyze the same data set. The warehouse provides an environment for making clean reliable data available for analysis.

The status of our warehouse at the moment of this writing is that a prototype has been implemented using the POET object oriented database. POET gives us a platform on which the data for the materialized views can be stored and the data mining tools can be integrated as part of the same system environment. The classes (materialized views) of the warehouse are populated by running our view system and capturing the resulting objects in our warehouse.

Figure 3 shows a block diagram of the current prototype. A user that wishes to create a materialized view for the warehouse first adds the view (object class) definition to the POET database (warehouse) and then executes an application using the view via the view system. The result of the application is then used to populate the materialized view. The instances in the materialized view are assigned "warehouse" object ids

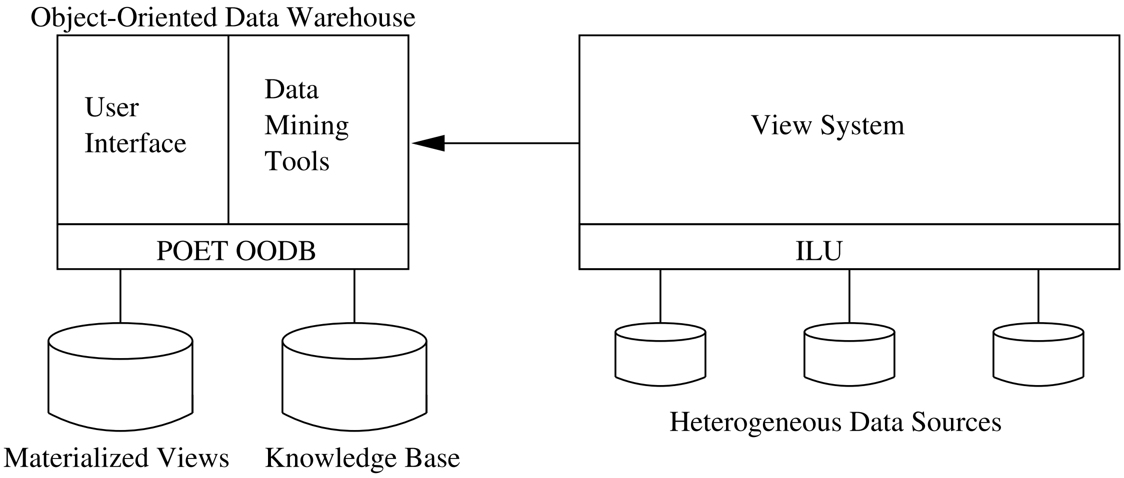

Figure 3: **The block design of the current prototype.**

by the POET database management system. Note that, in general, no object ids from an object oriented data source would be available outside of that local data source. Thus there is no conflict with the generation of new object ids for warehouse operations on the data.

Once the materialized view is available in the warehouse, the user can apply any of the data mining tools to the class (materialized view) that are appropriate for the structure of the object instances. Results from the data mining tools can either be presented to the user or stored in the knowledge base of the warehouse for future use with other data mining tools.

In future versions of our system we plan to develop a tighter coupling of the two sytems. We also expect that future versions of the warehouse will see improvements in the user interface and the availability of data mining tools (see the next section). We have also started to look at the use of intelligent agents as a way to reduce the amount of data that has to be moved in order to create the materialized views.

DATA MINING ISSUES

Our data mining experiments, aimed at heterogeneous data (e.g. textual data), are just in the beginning stages and only a few statistical tools are currently available in the warehouse. However,to give some closure to our presentation, we will conclude by looking at some of the issues we see ahead of us.

The view system and warehouse provide us with the necessary data and give us the ability to represent it in a variety of ways. This leaves us with a range of machine learning algorithms for data mining tools (Honavar and Uhr, 1994; Langley, 1995; Mitchell, 1997; Ripley, 1996). Some of the factors are (Honavar and Uhr, 1994):

- Overall objectives of the knowledge acquisition task: discovery of hidden regularities, pattern classification, approximation of an unknown function from examples, prediction, knowledge compilation, etc. For instance, decision trees or neural networks can extract the knowledge necessary to classify patterns in the domain of interest using the available data in the form of preclassified patterns.

- The nature and amount of a-priori domain knowledge that is available: Domain knowledge can often be used to reduce the computational effort and/or the amount of data that is needed for learning or to improve the quality of the acquired knowledge. Often, prior knowledge is available in a form (e.g., classification rules) that lends itself to gradual refinement using additional data (e.g., in the form of preclassified patterns).

- Choice of data and knowledge representation languages: Different families of machine learning algorithms assume different ways of representing data to be processed and the knowledge that is acquired. For instance, many popular data-driven inductive machine learning algorithms for pattern classification tasks assume that data (preclassified patterns) are represented as vectors of attribute values with an associated class name. Knowledge might be represented in the form of a classification or decision tree, a neural network, a collection of rules, a logic expression, etc. The computational effort needed and hence the feasibility of learning is a function of the data and knowledge representation languages. In particular, the knowledge representation language has to be expressive enough to describe knowledge in the domain of interest. However, too much expressive power can adversely affect the computational tractability of learning. The choice of the knowledge representation language may also be influenced by whether the results of knowledge acquisition need to be in a form that is easily understandable by humans. For instance, rules and decision trees may be easier to understand than knowledge represented in the form of a neural network (unless some rule extraction algorithms are used to summarize this knowledge). The choice of a machine learning algorithm to be used for knowledge acquisition is a function of both the data and knowledge representation languages.
- The amount as well as the quality of data that is available: If the data is available in a knowledge-rich form (e.g., definitions in a dictionary), then knowledge acquisition might involve little more than transforming the data directly into knowledge (e.g., rules) in a sufficiently expressive knowledge representation language. Other knowledge acquisition tasks might require the use of inductive learning to extract knowledge from data. The performance of inductive learning algorithms depends critically on the availability of both domain knowledge and adequate data (that is reasonably representative of the domain of interest). Domain knowledge, when available, can significantly improve the quality of the acquired knowledge. It can also reduce the amount of data and the computational resources needed for learning. If the domain is prone to noisy data or missing attribute values, the robustness of the learning algorithm in the presence of noise becomes important. If the domain lends itself to active data collection by the knowledge acquisition module (e.g., through selection of data items that are likely to be most informative given the current knowledge of the system), it is often possible to reduce the quantity of data that is needed.

The preceding discussion makes it clear that no single machine learning algorithm can solve the knowledge acquisition problem in every domain. However, machine learning does offer perhaps the most practical and cost effective approach to knowledge acquisition for a broad range of applications. Thus, automated knowledge acquisition and discovery from heterogeneous data sources calls for a system that supports the use of multiple machine learning paradigms and algorithms in a synergistic fashion. Furthermore, the system must be modular and extensible so that the individual knowledge acquisition modules using specific machine learning algorithms can be easily added, modified, or replaced with minimal effect on the rest of the system.

CONCLUSION

A view mechanism and an object oriented data warehouse based on the view mechanism has been presented. The warehouse will be used to conduct data mining experiments on semi and unstructured data.

REFERENCES

Abiteboul, S., and A. Bonner (1991). Objects and views. *ACM SIGMOD*, 238–247.

Barsalou, T., A. M. Keller, N. Siambela and G. Wiederhold (1991). Updating relational databases through object-based views. *ACM SIGMOD*, 248–257.

Baekgaard, L. and N. Roussopoulos (1997). Efficient refreshment of data warehouse views. Technical Report, Department of Computer Science, University of Maryland, URL: http://www.cs.umd.edu/TRs/authors /Nick_Rouss opoulos-no-abs.html.

Bright, M.W., A.R. Hurson and S.H. Pakzad (1992). A taxonomy and current issues in multidatabase systems. *Computer,* 25, 3, 50-60.

Chen, I. A., and D. McLeod (1989). Derived data update in semantic databases. *Very Large Data Bases*, 225–235.

Czejdo, B., and M. C. Taylor (1992). Integration of information systems using an object-oriented approach. *The Computer Journal*, 35, 5, 501–513.

Garlan, D. (1986). Extending IDL to support concurrent views. *ACM SIGPLAN*, 22, 11, 95–110.

Gupta, H. (1997). Selection of views to materialize in a data warehouse. To appear in the *Proceedings of the International Conference on Database Theory*, Athens, Greece.

Hammer, J., H. Garcia-Molina, J. Widom, W. Labio, Y. Zhuge (1995). The Stanford data warehousing project. In *IEEE Data Engineering Bulletin.*

Heiler, S., and S. Zdonik (1990). Object views: extending the vision. *IEEE Data Engineering*, 86–93.

Heimbigner, D., and D. McLeod (1985). A federated architecture for information management. *ACM Trans. on Office Information Systems*, 3, 3, 253-278.

Honavar, V. and L. Uhr (1994). Toward learning systems that integrate different strategies and representations. *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and L. Uhr (eds.), Academic Press, Boston.

Huyn, N. (1996). Efficient view self-maintenance *Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications*, Montreal, Canada.

Kaul, M., K. Drosten and E. J. Neuhold (1990). ViewSystem: integrating heterogeneous information bases by object-oriented views. *IEEE Data Engineering*, 2–10.

Kawaguchi, A., D. Lieuwen, I.S. Mumick, D. Quass, K.A. Ross. (1997). Concurrency control theory for deferred materialized views. To appear in *Proceedings of the 1997 ICDT Conference.*

Kawaguchi, A., D. Lieuwen, I. Mumick, D. Quass, K. Ross (1997). Concurrency control theory for deferred materialized views. To appear in the *Proceedings of the International Conference on Database Theory*, Athens, Greece.

Langley, P. (1995) *Elements of Machine Learning.* Morgan Kaufmann, Palo Alto, CA.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill, New York.

The Common Object Request Broker: Architecture and Specification, Object Management Group (1991). OMG Document 91.12.1.

OMG Architecture Guide Chapter 4: The OMG Object Model (1992). Object Management Group.

Mattison, R. (1996). *Data Warehousing:Strategies, Technologies and Techniques*, McGraw Hill, New York.

Peters, R. J., M. T. Ozsu and D. Szafron (1992). TIGUKAT: An object model for query and view support in object database systems, TR 92-14, U. of Alberta.

Premerlani, W. J., M. R. Blaha, J. E. Rumbaugh and T. A. Varwig (1990). An object-oriented relational database. *Communications of the ACM*, 99–109.

Quass, D., A. Gupta, I. S. Mumick, and J. Widom (1996). Making views self-maintainable for data warehousing. *Proceedings of the Conference on Parallel and Distributed Information Systems*, Miami Beach, FL.

Ripley, B.D. (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press, New York.

Scholl, M.H., C. Laasch and M. Tresch (1991). Updatable views in object-oriented databases. *Deductive and Object-Oriented Databases (DOOD)*, 189–207.

Sheth, A. and L. Kalinichenko (1992). Information modeling in multidatabase systems: beyond data modeling. *Int'l Conference in Knowledge Management*, 8–16.

Shilling, J.J. and P.F. Sweeney (1989). Three steps to views: extending the object-oriented paradigm, *OOPSLA*, 353–361.

Su, S., S. Fang, and H. Lam (1993). An object-oriented rule-based approach to data model and schema integration. *Technical Report, U. of Florida,*.

Wiederhold, G. (1986). Views, objects, and databases. *IEEE Computer*, 37–44.

Wiener, J.L. , H. Gupta, W.J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom (1996). A system prototype for warehouse view maintenance. *Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications*, Montreal, Canada, 26-33.

Yen, Cheng-Huang, L. Miller, and S. Pakzad (1994). The design and implementation of the *Zeus View System. 27th Hawaii International Conference on Systems and Sciences,* 206-215.

Yen, C.H. and L.L. Miller (1995). An extensible view system for multidatabase integration and interoperation. *Integrated Computer-Aided Engineering,* 2. 2. 97-123.

Zhuge, Y. , H. Garcia-Molina, J. Hammer, and J. Widom (1995). View Maintenance in a Warehousing Environment. *Proceedings of the ACM SIGMOD Conference*, San Jose, California.