Knowledge Discovery in Object-Oriented and Active Databases *

Jiawei Han[§] Shojiro Nishio[†] Hiroyuki Kawano[‡]

[§] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6 (han@cs.sfu.ca)

[†] Department of Information Systems Engineering, Osaka University, Osaka 565, Japan (nishio@ise.eng.osaka-u.ac.jp)

[‡] Department of Applied Math. and Physics, Kyoto University, Kyoto 606, Japan (kawano@kuamp.kyoto-u.ac.jp)

Abstract

Knowledge discovery in databases (or data mining), which extracts interesting knowledge from large databases, represents an important direction in the development of data- and knowledge- base systems. With fruitful research results on knowledge discovery in relational databases and the emerging trend in the development of object-oriented and active database systems, it is natural to investigate knowledge discovery in object-oriented and active databases. This paper overviews the mechanisms for knowledge discovery in object-oriented and active database systems, with an emphasis on the techniques for generalization of complex data objects, methods, class hierarchies and dynamically evolving data, and on the integration of knowledge discovery mechanisms with production control processes.

1 Introduction

With rapid growth in the amount of information stored in databases, the development of effective and efficient tools for *knowledge discovery in databases* (*KDD*, or *data/knowledge mining*) has become an increasingly important task in both database and machine learning researches [21, 7, 22].

In the past several years, fruitful research has been conducted on knowledge discovery in relational databases, with some experimental systems constructed and tested in large databases [7, 11, 19, 10]. With the emerging trend in the development of new database systems, such as object-oriented and active databases, for advanced database applications [12, 3], it is important to extend KDD techniques to such new database systems.

Object-oriented data models and systems [2, 3, 12] embody rich data structures and semantics in the construction of complex databases, such as complex data objects, class hierarchies, property inheritance, methods and active data, etc. This not only brings the power and flexibility to the system but also adds complexity to the implementation techniques, including the development of knowledge discovery mechanisms [14].

Active database techniques react to the changes of environments or production processes automatically by referencing database data and triggering appropriate actions. With the help of knowledge discovery techniques, general regularities of a dynamic system could be discovered and the conditions and actions of a rule could be specified at a concept level higher than that stored as primitive data. Therefore, it is interesting to examine the integration of knowledge discovery methods with active database techniques.

Similar to knowledge discovery in relational databases, many kinds of knowledge rules, such as characteristic rules, discriminant rules, data evolution regularities, conceptual clusters, and certain interesting value-dependent patterns can be discovered in object-oriented and active databases. Furthermore, knowledge discovery will benefit generalization on complex data objects, schema formation and schema evolution in OODBs [23], generalized triggering and step-by-step refined discovery in active databases, etc.

Figure 1 outlines how a database system reacts to the changing of an environment by incorporation of knowledge discovery and active database techniques: First, the status of an environment is collected into a database by a data sampling process. Second, a knowledge discovery process extracts characteristics of current status, stable rules, evolution rules, etc. from the database and stores them into a knowledge base. Third, an active database examines the current status and other system behaviors and triggers some prespecified actions to change the environment, which in turn

*The research of the first author was supported in part by the grants from the Natural Sciences and Engineering Research Council of Canada and the Centre for Systems Science of Simon Fraser University, that of the second author was supported in part by the Ministry of Education, Science and Culture of Japan under Scientific Research Grant-in-Aid, and that of the third author was supported in part by a scholarship from the Ministry of Education, Science and Culture of Japan and the work was done during his visit to Simon Fraser University.

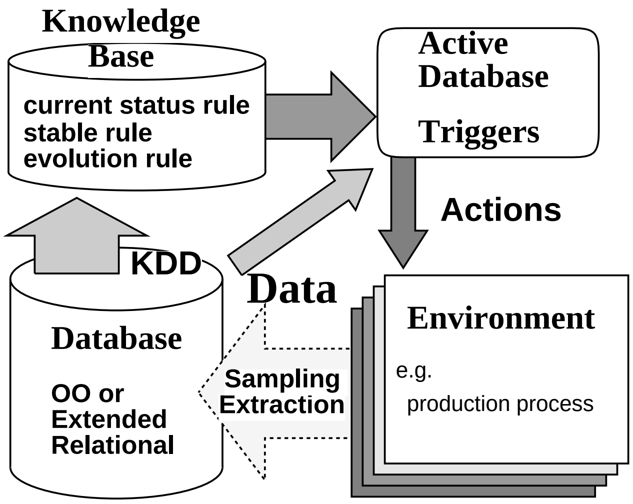

Figure 1: Knowledge Discovery in OO and active DBs.

serves as new sources of data for databases.

This paper presents a general overview on knowledge discovery in object-oriented and active databases. Because of the broad scope of the topic, the paper is descriptive in nature, with rigorous treatment referenced to the bibliography.

The paper is organized as follows. In Section 2, we examine the generalization of complex data objects in OODBs. In Section 3, the control of generalization in OODBs is studied. In Section 4, knowledge discovery in active databases and its applications are discussed. The study is summarized in Section 5.

2 Generalization in OODBs

An OODB organizes a large set of complex data objects into classes which are in turn organized into class/subclass hierarchies with rich data semantics. Each object in a class is associated with (1) an object-identifier, (2) a set of attributes which may contain sophisticated data structures, set- or list- valued data, class composition hierarchies, multimedia data, etc. and (3) a set of methods which specify the computational routines or rules associated with the object class.

To facilitate the development of KDD mechanisms in OODBs, it is important to implement efficiently a relatively small set of generalization operators on which a large set of possible generalization operations can be constructed.

Formally, the generalization of a component p of an object O_i can be written in an abstract way as $Gen(O_i.p)$, where Gen is an abstract object generalization operator which can be transformed into a concrete operation based on the role of the component and the specific learning requirement. Moreover, the generalized component of an object can be further generalized by applying Gen again, which could be the same or different generalization operators compared with the one applied in the last generalization. If the generalization is performed by applying the same sequence of generalization operators on a component p of O_i, we should have

$$Gen^{n-1}(Gen(O_i.p)) \quad \equiv \quad Gen(Gen^{n-1}(O_i.p)).$$

For example, a person P_1's address can be generalized from a detailed address, such as the number of a street, into a higher leveled one, such as a street block, then a street, a district, a city, a province, a country, etc. when necessary by applying the generalization operator Gen several times, such as $Gen(Gen(\cdots Gen(P_1.address)\cdots))$.

An object in an OODB is described by a set of attributes and a set of methods. The attribute value of an object could be a character, a fixed length character string, a numerical value, a structure, a class composition hierarchy, a set-valued or list-valued data, or unformatted data, such as text, map, image, voice or other forms of multimedia data. The generalization of different and complex kinds of attribute values is a challenging task.

In this section, we examine the generalization of different components of complex objects in an OODB, including object identifiers, unstructured and structure values, class composition hierarchies, spatial and multimedia data, inherited and derived data, methods specified by rules or procedures, etc.

2.1 Generalization of object identifiers

One of the essential components of an OODB is *object identifier* which uniquely identifies objects. It remains unchanged over structural reorganization of data. At the first glance, it seems to be impossible to generalize an object identifier. However, since objects in an OODB are organized into classes which in turn belong to certain class and subclass hierarchy, the generalization of objects may refer to their corresponding class and subclass hierarchy. Therefore, an object identifier can be first generalized to its corresponding lowest subclass names which can in turn be generalized to a higher level class/subclass name by climbing up the class/subclass hierarchy.

2.2 Generalization on single attribute values

Single valued, numerical and non-numerical data are the most popularly encountered attribute values in databases.

Generalization on non-numerical values may rely on the available concept hierarchies specified by domain experts or users. Concept hierarchies represent necessary background knowledge which directs the generalization process. Different levels of concepts are often organized into a taxonomy of concepts. The concept taxonomy can be partially ordered according to a general-to-specific ordering. The most general concept (corresponding to *level 0*) is the null description (described by a reserved word "*ANY*"), and the most specific concepts correspond to the specific values of attributes in the database. Using a concept hierarchy, the discovered rules can be represented in terms of generalized concepts and stated in a simple and explicit form, which is desirable to most users.

A conceptual hierarchy could be given by users or experts, stored or partially stored as data in a database, specified by some generalization meta-rules, such as deleting the street number from a street address, etc., being derived from the knowledge stored elsewhere, or being computed by applying some rules or algorithms, such as deriving "*British Columbia ⇒ Western Canada*" from a geographic map stored in a spatial database, etc. Also, it could be defined on a single attribute or on a set of related attributes, and it could be in the shape of a balanced tree, an unbalanced tree, a lattice or a general DAG.

Furthermore, a given hierarchy may often need to be modified or refined dynamically based on user's learning requirements and/or database statistics. For example, if the learning requirement is to analyze the birth place of the *students* of Simon Fraser University, the level 1 concepts could be: {*B.C, other_provinces_in_Canada, foreign*}; however, if it is to analyze the birth place of the *faculty* of the University, the appropriate level 1 concepts could be {*North_America, Europe, Asia, other_countries*}. Both concept hierarchies can be obtained by dynamic adjustment of a given concept hierarchy based on the analysis of the statistical distribution of the relevant data sets.

Generalization on numerical attributes can be performed similarly but in a more automatic way by the examination of data distribution characteristics [6]. In many cases, it may not require any predefined concept hierarchies. For example, the ages of the graduate students in a university can be clustered into several groups, such as {*below 23, 23-26, 26-30, over 30*}, according to a relatively uniform data distribution criteria or using some statistical clustering analysis tools. Appropriate names can be assigned to the generalized numerical ranges, such as {*very young, young,* $\cdots$} by users or experts to convey more semantic meaning.

2.3 Generalization on structured data

Complex structure-valued data, such as set-valued and list-valued data and data with nested structures, are common in OODBs. They can be generalized in several ways in order to extract interesting patterns from such data sets.

A set-valued attribute may be of homogeneous or heterogeneous types. Typically, a set-valued data can be generalized in two ways: (1) generalization of each value in a set into its corresponding higher level concepts, or (2) derivation of the general behavior of a set, such as the number of elements in the set, the types or value ranges in the set, the weighted average for numerical data, etc. Notice that in the case of set-valued attribute generalization, the generalization operator $Gen(p_k)$ indicates that the input p_k can be a set of values and the output "$p_{k-1} = Gen(p_k)$" may also be a set of values. Moreover, the generalization can be performed by applying different generalization operators to explore alternative generalization paths. In this case, the result of generalization must be a heterogeneous set.

For example, the *hobby* of a person is a set-valued attribute which contains a set of values, such as {*tennis, hockey, chess, violin, nintendo*}, which can be generalized into a set of high level concepts, such as {*sports, music, computer_games*}, or into 5 (the number of hobbies in the set), or both, etc. Moreover, a *count* can be associated with a generalized value to indicate how many elements are generalized to the corresponding generalized value, such as {*sports(3), music(1), computer_games(1)*}, where *sports(3)* indicates *three kinds of sports*, etc.

A set-valued attribute may be generalized into a set-valued or a single-valued attribute; whereas a single-valued attribute may also be generalized into a set-valued one if the "hierarchy" is a lattice or the generalization follows different paths. Further generalizations on such a generalized set-valued attribute should follow the generalization path of each value in the set.

A list-valued or a sequence-valued attribute can be generalized in a way similar to the set-valued attribute except that the order of the elements in the sequence should be observed in the generalization.

Set- and list-valued attributes are simple structure-valued attributes. In general, a structure-valued attribute may contain sets, tuples, lists, trees, records, etc. and their combinations. Furthermore, one structure can be nested in another structure at any level. Similar to the generalization of set- and list-valued attributes, a general structure-valued attribute can be generalized in several ways, such as (1) generalize each attribute in the structure whereas maintain the shape of the structure, (2) flatten the structure and generalize on the flattened structure, (3) remove the low-level structures or summarize the low-level structures by high-level concepts or aggregation, and (4) return the type or an overview of the structure.

2.4 Aggregation and approximation as a means of generalization

Besides concept tree ascension and structured data summarization, aggregation and approximation should be considered as an important means of generalization, which is especially useful for generalization of attributes with large sets of values, complex structures, spatial or multimedia data, etc.

Take spatial data as an example. It is desirable to generalize detailed geographic points into clustered regions, such as business, residential, industry, or agricultural areas, according to the land usage. Such generalization often requires the merge of a set of geographic areas by spatial operations, such as spatial union, or spatial clustering algorithms. Approximation is an important technique in such generalization. In spatial merge, it is necessary not only to merge the regions of similar types within the same general class but also to ignore some scattered regions with different types if they are unimportant to the study. For example, different pieces of land for different purposes of agricultural usage, such as vegetables, grain, fruits, etc. can be merged into one large piece of land by spatial merge. However, such an agricultural land may contain highways, houses, small stores, etc. If the majority land is used for agriculture, the scattered spots for other purposes can be ignored, and the whole region can be claimed as an agricultural area by approximation. The spatial operators, such as *spatial_union, spatial_overlapping, spatial_intersection*, etc., which merge scattered small regions into large, clustered regions can be considered as generalization operators in spatial aggregation and approximation.

2.5 Generalization on multimedia data

A multimedia database may contain complex text, graphics, images, maps, voice, music, and other forms of audio/video information. Such multimedia data are typically stored as sequences of bytes with variable lengths, and segments of data are linked together for easy reference. Generalization on multimedia data can be performed by recognition and extraction of the essential features and/or general patterns of such data.

There are many ways to extract the essential features or general patterns from segments of multimedia data. For an image, the size and color of the contained objects or the major regions in the image can be extracted by aggregation and/or approximation. For a segment of music, its melody can be summarized based on the approximate patterns that repeatedly occur in the segment and its style can be summarized based on its tone, tempo, major musical instruments played, etc. For an article, its abstract or general organization such as the table of contents, the subject and index terms frequently occurring in the article, etc. may serve as generalization results. In general, it is a challenging task to generalize multimedia data to extract the interesting knowledge implicitly stored in the data. Further research should be devoted to this issue.

2.6 Generalization on inherited and derived properties

OODBs are organized into class/subclass hierarchies. Some attributes or methods of an object class are not explicitly specified in the class itself but are inherited from its higher level classes. Some OODB systems may allow the properties to be inherited from more than one superclass (called *multiple inheritance*) when the class/subclass "hierarchy" is organized in the shape of a lattice. The inherited properties of an object can be derived by query processing in the OODB. From the knowledge discovery point of view, it is unnecessary to distinguish which data are stored within the class and which are inherited from its superclass. As long as the set of relevant data are collected by query processing, the knowledge discovery process will treat the inherited data in the same way as the data stored in the object class and perform generalization accordingly.

Method is another important component of OODBs. Many behavioral data of objects can be derived by application of methods. Since a method is usually defined by a computational procedure/function or by a set of deduction rules, it is difficult to perform generalization on the method itself unless the generalization of the method is clearly understood by application programmers and is coded as a new method which directly performs the required generalization. In general, the generalization on the data derived by method application should be performed in two steps: (1) deriving the task-relevant set of data by application of the method and, possibly, also data retrieval; and (2) performing generalization by treating the derived data as the existing ones.

2.7 Generalization on class composition hierarchies

An attribute of an object may be composed by another object, and some of whose attributes may be composed in turn by other objects, thus forming a class composition hierarchy. Generalization on a class composition hierarchy can be viewed as generalization on a set of (possibly infinite, if the nesting is recursive) nested structured data.

In principle, the reference to a composite object may traverse via a long sequence of references along the corresponding class composition hierarchy. However, in most cases, the longer sequence of references is traversed, the weaker semantic linkage between the original objects and the referenced composite objects is obtained. For example, one attribute "*vehicles_owned*" of an object class "*student*" could refer to another object class "*car*" which may contain an attribute "*auto_dealer*", which may refer to its "*manager*" with an attribute "*children*". Obviously, it is unlikely to find any interesting regularities between a student and his/her car's dealer's manager's children.

Therefore, generalization on a class of objects should be mainly performed on its own descriptive attribute values, methods, etc. with limited references to (and therefore generalization on) its composed object hierarchy. That is, in order to discover relatively interesting knowledge, generalization should be performed only on the composite objects closely related to the currently focused class(es) but not on those which have only remote and rather weak semantic linkages.

3 Control of Generalization Processes

Task-relevant data can be viewed as examples for learning processes. Undoubtedly, *learning-from-examples* [16, 9] should be an important strategy for knowledge discovery in databases. Most *learning-from-examples* algorithms partition the set of examples into *positive* and *negative* sets and perform *generalization* using the positive data and *specialization* using the negative ones [16]. Unfortunately, a database does not usually store negative data explicitly, and thus no explicitly specified negative examples can be used for specialization. Therefore, a database induction process relies mainly on generalization, which should be performed cautiously to avoid over-generalization.

A knowledge discovery process applies a sequence of generalization operators to a set of data to generate a new set of data. A set of data can be viewed as a class from the view of OODB. The class which is fed into a generalization operator for generalization is called the *working class*, $\mathcal{W}$, with the initial one (the set of task-relevant data) called the *initial working class*, $\mathcal{W}_0$. Notice that such a class could be obtained by querying on a set of classes of objects under some specified query condition. The class generated by the application of a generalization operator is called the *resulting class*, $\mathcal{R}$.

An attribute-oriented induction method, developed in the study of knowledge discovery in relational databases [10], can be extended to OODBs. The method is briefly outlined as follows. First, a set of generalization operators are selected and applied to an attribute in the working class without considering the inter-relationships among different attributes before the concepts in each attribute are generalized to a desired level. The reason to ignore the inter-relationships among different attributes at an early stage of generalization is that such inter-relationships, if considered at an early stage, would have to be expressed at an undesirably low level in large numbers. This cannot lead to elegant generalized rules to be expressed at a high level in concise terms. Attribute-oriented induction, which considers the attribute inter-relationships only when the data has been generalized into a relatively small set, will substantially reduce the computational complexity of a database learning process.

3.1 An attribute-oriented induction method

In general, we have the following basic techniques for attribute-oriented induction [10] in OODBs.

Technique 1 (Data focusing) *Generalization should be performed only on the set of data which are relevant to the learning request.*

Technique 2 (Fine granularity generalization) *Generalization should be considered on the smallest decomposable components (or attributes) of the relevant data objects.*

Technique 3 (Attribute removal) *If there are a large set of distinct values in an attribute in the working class, but there is no generalization operator on the attribute, the attribute should be removed from the working class.*

Technique 4 (Attribute generalization) *If there are a large set of distinct values in an attribute in the working class, but there exist a set of generalization operators on the attribute, a generalization operator should be selected and applied to the attribute at every step of generalization.*

As a result of generalization, different objects may be generalized to equivalent ones where two (generalized) objects are **equivalent** if they have the same corresponding attribute values without considering their *object identifiers* and a special internal attribute **count**, which registers the number of objects in the initial working class that are generalized to the object in the current resulting class. Notice that a *generalized object*, though has its own new object identifier in an OODB, is a carrier of the general properties of a set of initial objects because the original object identifier has been generalized into a class or superclass name. The *count* accumulated in the generalized class incorporates quantitative information in the learning process.

Technique 5 (Count propagation) *The count of a generalized object should be carried to its further generalized object, and the count should be accumulated when merging equivalent objects in generalization.*

Technique 6 (Attribute generalization control)

Generalization on an attribute a_i is performed until the concepts in a_i is generalized to a desired level, or the number of distinct values in a_i in the resulting class is no greater than a prespecified threshold.

Notice that *the threshold which controls the attribute generalization* is called **attribute threshold** which is usually a small number (often between 2 to 10) that can be specified explicitly by a user/expert or be built in the system as a default.

Remark 1 *The above generalization techniques are correct and necessary for the extraction of generalized rules from databases.*

Rationale. Technique 1 is obvious since only the task-relevant set of data need to be studied. Technique 2 follows the principle that the smallest decomposable components should be considered in generalization in order to avoid over-generalization. An attribute-value pair represents a conjunct in a generalized rule, the removal of a conjunct eliminates a constraint and thus generalizes the rule. If there is a large set of distinct values in an attribute but there is no generalization operator for it, the attribute should be removed. Thus, we have Technique 3 which corresponds to the generalization rule, *dropping conditions*, in *learning-from-examples* [16]. The generalization of an attribute value using a selected generalization operator makes the object covers more cases than the original one and thus generalizes the concept. Thus, we have Technique 4 which corresponds to the generalization rule, *climbing generalization trees*, in *learning-from-examples* [16]. Technique 5 is based on the merge of identical tuples. Technique 6 is based on the desirability of representation of each attribute at its desired level. □

The application of a database generalization operator on a *working class* results in a more general *resulting class*, which in turn becomes the working class in the next round of database generalization. Such a generalization process proceeds until the concept in every attribute in the resulting class has reached to a desired concept level, or the number of distinct values in every attribute is no greater than its attribute threshold. The generalized resulting class so obtained is called a **prime generalized class**.

3.2 Generalized rule extraction

Since the above induction process enforces only attribute generalization control, the prime generalized class so extracted may still contain a relatively large number of generalized objects. Two alternatives can be developed for the extraction of generalized rules from a prime generalized class: (1) further generalize the class to derive a **final generalized class** which contains no more objects than a prespecified *class threshold*, and then extract the *final generalized rule*; and (2) directly extract **generalized feature table** and present feature-based multiple rules.

Alternative 1 is based on the following Technique 7. The interestingness of the final generalized rule relies on the selection of the attributes to be generalized and the selection of generalization operators. Such selections can be based on data semantics, user/expert preference or instructions, execution history, processing efficiency, etc. When it is not easy to make a selection, one may proceed along several generalization paths in parallel until later stage because promising regularities may be discovered by pursuing different generalization paths.

Technique 7 (Class generalization control) *Generalization on a prime generalized class is performed until the number of distinct generalized objects in the resulting class is no greater than a prespecified class threshold.*

Alternative 2 takes the set of generalized objects and maps them into a generalized feature table [10]. Based on the generalized feature table, multiple generalized feature-based rules can be presented.

3.3 Generalization algorithms

Similar to the studies of KDD in relational databases, the above discussion can be summarized into the following generalization algorithm which extracts generalized characteristic rules in an OODB based on a user's learning request, where a *characteristic rule* is an assertion which characterizes the concepts satisfied by all or a majority number of the examples in the task-relevant data set.

Algorithm 1 (Basic A-O induction in OODB) *Discovery of a set of generalized characteristic rules based on a user's learning request by attribute-oriented induction.*

Input. (i) An OODB DB, (ii) a set of concept hierarchies or generalization operators on attributes a_i, and (iii) T, a *class threshold*, and T_i, a set of *attribute thresholds* for attributes a_i.

Output. A *characteristic rule* based on the learning request.

Method.

1. Derive the *initial working class*, $\mathcal{W}_0$, by collecting the set of task-relevant data based on the learning request.

2. Derive the *prime generalized class*, $\mathcal{R}_p$, by performing attribute-oriented induction according to the techniques 2 – 6.

Note: Generalization on each attribute a_i can be implemented efficiently by (1) collecting the distinct a_i values in the working class, (2) computing the minimum desired level L, and (3) generalizing the attribute to this level L by replacing each value in a_i's with its corresponding superordinate concept in H_i (the concept hierarchy for a_i) at level L.

3. Presentation of generalized rules either from a final generalized class (by further generalization) or from a generalized feature table (by feature mapping). □

Step 1 of the algorithm is essentially an OODB query whose processing efficiency depends on a particular query processing algorithm. The worst-case time complexity of Steps 2 & 3 in Algorithm 1 is $O(n \log(n))$ where n is the number of data objects relevant to the learning request [17]. Therefore, it is a fairly efficient algorithm for knowledge extraction in large databases.

The above technique (for learning characteristic rules) can be extended to the discovery of other kinds of rules in OODBs, such as discriminant rules, data evolution regularities, approximate rules, etc., where a **discriminant rule** is an assertion that discriminates a concept of the class being learned (called the *target class*) from other classes (called the *contrasting classes*); **data evolution regularity** reflects the general trend of changes in data contents in the database over time; whereas an **approximate rule** is the rule which represents the characteristics of a *majority* number of facts in the database, which is especially useful when databases contain noisy data and exceptional cases.

Here we briefly outline how to extend the method to the discovery of discriminant rules. To distinguish two sets of objects with different properties, it is necessary to first collect the task-relevant objects into two classes: the *target class* and the *contrasting class*, and then perform generalization *synchronously* on the two classes. Once the concepts are generalized to the *same* high level, general behaviors can be compared between the two classes and the discriminative behaviors can be extracted as discriminant rules.

Another important issue is the control of a discovery process. When every relevant attribute has been generalized to a relatively high concept level (especially when it reaches the desired concept level), the selection of attributes for further generalization becomes subtle. Criteria for selection of one attribute for generalization instead of another should be based on the concern of data and query semantics, user-preference, efficiency, etc. For example, if certain attribute becomes a focus of study, the selection of that attribute for progressive generalization should often be conservative in order to observe its regularity with the changes of other attributes.

3.4 Applications of KDD in OODBs

Knowledge discovery in databases may disclose interesting relationships and regularities of data in databases, help understanding database contents and data evolution regularities, and facilitate intelligent query answering and semantic query optimization.

Moreover, since object-oriented database consists of rich and complex structures, and such structures may evolve constantly in the life span of a database system, an important application of knowledge discovery in OODBs is at *schema formation* and *schema evolution* in object-oriented databases [23].

By knowledge discovery, a characteristic rule may partition a database into several different cases, and a discriminant rule may distinguish one class of data from others. This often forms a basis for partitioning data into different classes. Furthermore, there could be different schemes of class partitioning, and the judgement of which one is more preferable than others could be based on the simplicity of the generalized rules and the coverage of one scheme of class partitioning vs. others.

4 Integration of Knowledge Discovery and Active Database Technologies

Many large, automated systems generate, store and reference voluminous system status and control data and react promptly to the changes of a dynamic environment. Such kind of systems may require the integration of the technologies of active databases and knowledge discovery in databases.

First, in such a dynamic environment, data are generated rapidly, continuously, dynamically and in huge volumes. It is often unrealistic to store a *complete* set of raw data in the limited amount of memory of a database system and dynamically analyze and manage the data. This forces people either to abandon the dynamically generated, huge amount of data or to dump the generated data to tapes without timely analysis. By doing so, it is difficult to grasp the current status of a dynamic environment and react promptly to changes.

Second, most of the data/information in a dynamic environment are presented at low, primitive levels. There may not exist clear and concise relationships or regularities expressible by low level primitives. Moreover, human operators and programmers may like to analyze system conditions and express the control primitives at a relatively high level, comprehensible by human operators. There is a gap between the low-level processing data and high level control primitives.

Third, process control and system management in a dynamic environment often require *prompt, real-time*, and *intelligent* reactions in response to situation changes in the environment. The off-line data/pattern analysis is often too slow to meet the real time requirement. Furthermore, even if the correct patterns or regularities can be discovered, real-time decision must be made *promptly* and *automatically* in order to react correctly and timely to the changing environment.

These challenges motivate the integration of knowledge discovery and active database technologies in the following aspects.

1. *The collection of a large amount of data generated rapidly, continuously, and dynamically in an environment* can be handled by a data sampling technique which samples interesting pieces of information dynamically and systematically.
2. *The discovery of clear and concise relationships or regularities among the collected data*, can be handled by a knowledge discovery technique which performs efficient and effective data generalization to discover useful knowledge or regularity from the collected information [18, 21, 7].
3. *The prompt, real-time, and intelligent reactions to the changes of the environment*, can be dealt with by application of active database technology [4, 15, 8] for automatic and prompt reaction and control of the environment.

4.1 Learning current status, stable and evolution rules in dynamic environments

In a dynamic environment, the status of a system changes with time. It is important for a knowledge discovery process to extract the general characteristics of the present status and the stable/changing data. Therefore, the following three kinds of rules need to be discovered, and the discovered rules should be stored selectively in an active database and be applied appropriately to the control of the system.

1. **current status rules.** A *current status rule* summarizes the general characteristics of a set of sampled data at the present time which satisfies certain criteria, such as, the characteristics of the traffic flow on a highway at the present time.
2. **stable rules.** A *stable rule* describes the general characteristics which remain stable over a period of time, such as, the rule that helps find out the heavy traffic on a highway constantly or periodically.
3. **evolution rules.** An *evolution rule* describes the general characteristic of a set of patterns which evolve over several periods of sampling time, such as, how the highway traffic changes drastically over the past several sampling times.

The algorithm for the discovery of characteristic rules has been examined in the last section, which can be applied directly to the discovery of the current status rules. With minor extensions to the algorithm, it can be applied to the discovery of stable rules and evolution rules in a dynamic environment. Since a stable rule describes some general characteristics of stable data/patterns in a dynamic environment, whereas an evolution rule describes the regularities of changing data/patterns in the environment, it is important to extract data which remain stable or change with time.

In a dynamic environment, it is difficult to avoid minor fluctuations. Thus it is normal to have some minor differences between the measurements at different sampling times as long as they are within a certain allowed range of fluctuations. Sometimes, it becomes subtle whether some fluctuations should be considered as "stable data" or "changes". System specifications, processing history and general statistics should be used to judge a classification and determine whether the system should pay attention to such status changes and react properly.

For a knowledge discovery process, it is important to distinguish stable data from the changing data and generalize them to certain high level to extract general characteristics. After categorization and collection of stable and changing data, the data generalization and knowledge discovery processes are similar to the learning of characteristic or discriminant rules discussed in the last section.

4.2 Intelligent reactions to dynamic environments

To react intelligently to dynamic environments, the active database techniques and knowledge discovery processes can be integrated in the following five aspects:

1. **regularity extraction**.
2. **regularity updates**.
3. **knowledge-assisted active rule specification**.
4. **dedicated knowledge discovery**.
5. **generalized triggering**.

These aspects are further analyzed below.

First, regularity extraction needs the integration of both techniques. A large number of rules which summarize the current status or the stable and evolving regularities of a system can be extracted by a knowledge discovery process. However, some of these discovered regularities could be less interesting or redundant to the system. An active database method may act as knowledge discovery initiator which triggers a knowledge discovery process based on the *importance* or *freshness* of the knowledge to the system. The importance is related to some critical or sensitive aspects of an environment, such as the potential crisis of a production process, the critical condition of a chemical reaction, etc.; whereas the freshness is related to whether similar knowledge is already in the system. Furthermore, if the dynamic data is too large to be stored in a database but constantly monitoring is more preferable than data sampling, the database may store data summaries (extracted by knowledge discovery) at a level slightly higher (thus less voluminous) than the primitive data.

Second, it is important to verify, modify or invalidate the existing generalized rules stored in the rule base in a dynamic environment. Such a regularity updating task can also be performed by integration of knowledge discovery and active database technology. When a rule is discovered by a knowledge discovery process, the rule could be in one of the following cases: (1) it may enrich an existing rule by consolidating it in an extended period, or extending its condition or conclusion, (2) it may invalidate an existing rule because of the changed condition or conclusion, (3) it may not be interesting or fresh enough for inclusion in the rule base, or (4) it may violate certain integrity constraints, thus need to invoke some warning messages or perform appropriate actions. Knowledge rule verification, modification, invalidation or other appropriate actions can be specified as actions of triggers in an active database, which can be invoked when certain conditions are detected in the knowledge discovery process.

Third, the integration of the both techniques may facilitate knowledge-assisted active rule specification. It is often necessary to express active rules at the concept level higher than the primitive data in dynamic environments for comprehension and debugging by human programmers/operators. Such specification needs the help of knowledge discovery process. Moreover, the specification of appropriate conditions and actions should be based on the analysis of general characteristics of the current system, the stable and evolving regularities, and the execution history of the active rules. Obviously, such specification, refinement, and assessment of the appropriate conditions and actions of active rules need the application of knowledge discovery tools.

Fourth, dedicated knowledge discovery need the use of both knowledge discovery and active database mechanisms. In general, data sampling and knowledge discovery can be classified into *general* and *dedicated* processes. The former is adopted for *regular* environment checking and knowledge discovery; whereas the latter need to be invoked for a dedicated, detailed, frequent, and specialized data sampling knowledge discovery when certain condition happens. For example, when a production environment reaches a critical condition, an emergency data collection and knowledge discovery process should be invoked for close observation. Such an invocation of a dedicated sampling and discovery process can be performed by specifying conditions and actions for a dedicated process using the active database technology. When some unusual situation was detected by a knowledge discovery process, more focused and refined knowledge discovery can be initiated, and such a process can be refined progressively based on the discovered results. Such progressively refined processes can be specified by active database rules.

Finally, the integration may facilitate generalized triggering. Since the general status and evolution regularities of a system can be discovered by a knowledge discovery process and summarized at a high concept level, the conditions and appropriate actions of an active rule can be specified at a high level to communicate with both human and the system. Moreover, the actions of an active rule can also be specified at a high concept level. Appropriate mappings can be performed to transform high level actions into low level primitives to trigger the detailed actions and update the environment.

5 Conclusions

In this paper, we examined knowledge discovery in object-oriented and active databases and studied the extension of the mechanisms for knowledge discovery in relational databases towards such new generation databases. Several important directions have been emphasized in this study, including generalization of complex data objects and multimedia data, generalization control mechanisms, interaction of knowledge discovery with active database methods and control processes, etc. The availability of generalization operators and knowledge discovery tools will substantially enhance the power and increase the flexibility of data and knowledge-base systems.

As an emerging field, knowledge discovery in database has not only attracted wide attention in research communities [21, 7, 22] but also shown its high promise in industrial applications. A number of knowledge discovery systems/prototypes have been constructed and experimented on medium to large databases, such as DBLEARN [10], INLEN [11], KDW+ [19], EXPLORA [13], FortyNiner [24], Opportunity Explorer [1], Datalogic/R [25], Knowledge-Seeker [5], etc. With increasing R&D activities on KDD, many technically mature KDD systems will emerge in the near future.

Most of the recent developments are focused on knowledge discovery in relational databases. There are still many research issues on knowledge discovery in complex databases. The construction of efficient and effective generalization operators for complex structured or unstructured data, such as hypertext and multimedia data, the construction of a multi-resolution model for object-oriented and active databases [20] for browsing database contents and answer interesting queries at high concept levels are interesting topics for future research. Finally, software development and experimentation should be performed on the proposed mechanisms to verify and improve the proposed techniques and compare them with other related, promising proposals [11, 18, 13, 24].

References

[1] T. Anand and G. Kahn. Oppotunity explorer: Navigating large databases using knowledge discovery templates. In *Proc. AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 45–51, Washington DC, July 1993.

[2] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database systems manifesto. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 223–240. Elsevier Science, 1990.

[3] F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System: The story of O2*. Morgan Kaufmann, 1992.

[4] C. Beeri and T. Milo. A model for active object-oriented database. In *Proc. 17th Int. Conf. Very Large Data Bases*, pages 337–349, Barcelona, Spain, Sept. 1991.

[5] B. de Ville. Applying statistical knowledge to database analysis and knowledge base construction. In *Proc. 6th Conference on Artificial Intelligence Applications*, pages 30–36, Santa Barbara, CA, 1990.

[6] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

[7] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, 1991.

[8] N.H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model and implementation. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 327–338, Vancouver, Canada, August 1992.

[9] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[10] J. Han, Y. Cai, N. Cercone, and Y. Huang. DBLEARN: A knowledge discovery system for databases. In *Proc. 1st Int. Conf. on Information and Knowledge Management*, pages 473–481, Baltimore, Maryland, Nov. 1992.

[11] K. A. Kaufman, R. S. Michalski, and L. Kerschberg. Mining for knowledge in databases: Goals and general description of the INLEN system. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 449–462. AAAI/MIT Press, 1991.

[12] W. Kim. *Introduction to Object-Oriented Databases*. The MIT Press, 1990.

[13] W. Kloesgen. Problems for knowledge discovery in databases and their treatment in the statistics interpreter explora. *Int. J. Intell. Syst.*, 7:649–73, 1992.

[14] M. Manago and Y. Kodratoff. Induction of decision trees from complex structured data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 289–306. AAAI/MIT Press, 1991.

[15] D.R. McCarthy and U. Dayal. The architecture of an active database management system. In *Proc. 1989 ACM SIGMOD International Conference on Management of Data*, pages 215–24, Portland, OR, June 1989.

[16] R. S. Michalski. A theory and methodology of inductive learning. In Michalski et. al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 83–134. Morgan Kaufmann, 1983.

[17] S. Nishio, H. Kawano, and J. Han. Knowledge discovery in object-oriented databases: The first step. In *Proc. AAAI-93 Workshop on knowledge discovery in databases*, pages 186–198, Washington, DC, July 1993.

[18] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.

[19] G. Piatetsky-Shapiro and C.J. Matheus. Knowledge discovery workbench for exploring business databases. In *7(7)*, pages 675–686, 1992.

[20] R.L. Read, D.S. Fussell, and A. Silberschatz. A multi-resolution relational data model. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 139–150, Vancouver, Canada, Aug. 1992.

[21] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *Comm. ACM*, 34:94–109, 1991.

[22] M. Stonebraker, R. Agrawal, U. Dayal, E. Neuhold, and A. Reuter. DBMS research at a crossroads: The vienna update. In *Proc. 19th Int. Conf. Very Large Data Bases*, pages 688–692, Dublin, Ireland, Aug. 1993.

[23] S. B. Zdonik. Incremental database systems: Database from the ground up. In *Proc. 1993 ACM-SIGMOD Conf. Management of Data*, pages 408–412, Washington, DC, May 1993.

[24] R. Zembowicz and J.M. Żytkow. Testing the existence of functional relationship in data. In *Proc. AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 102–111, Washington DC, July 1993.

[25] W. Ziarko, R. Golan, and D. Edwards. An application of Datalogic/R knowledge discovery tool to identify strong predictive rules in stock market data. In *Proc. AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 93–101, Washington DC, July 1993.