A Knowledge-Based Perspective of the Distributed Design of Object Oriented Databases

Fernanda Baião Marta Mattoso Gerson Zaverucha

baiao, marta, gerson@cos.ufrj.br

Department of Computer Science - COPPE/UFRJ

PO Box: 68511, Rio de Janeiro, RJ, Brazil, 21945-970

Telephone: +55+21+590-2552, Fax: +55+21+290-6626

Abstract

The performance of applications on Object Oriented Database Management Systems (OODBMSs) is strongly affected by Distributed Design, which reduces irrelevant data accessed by applications and data exchange among sites. In an OO environment, the Distributed Design is a very complex task, and an open research problem. In this work we propose a knowledge based approach to the fragmentation phase of the distributed design of object oriented databases. In this approach, we will show a rule-based implementation of an analysis algorithm from our previous work and propose some ideas towards the use of Inductive Logic Programming (ILP) to perform a knowledge discovery/revision process using our set of rules as background knowledge. The objective of the work is to uncover some previously unknown issues to be considered in the distributed design process. Our main objective here is to show the viability of performing a revision process in order to obtain better and better fragmentation algorithms. We do not intend to propose the best fragmentation algorithm ever possible. We concentrate here on the process of revising a DDOODB algorithm through Knowledge Discovery techniques, rather than only obtaining a final optimal algorithm.

1 Introduction

Distributed and parallel processing on OODBMS may improve performance for non-conventional applications that manipulate large volumes of data. This is addressed by removing irrelevant data accessed by queries and transactions and by reducing the data exchange among sites [10], which are the two main goals of the Distributed Design of Databases.

Distributed Design involves making decisions on the fragmentation and placement of data across the sites of a computer network [16]. The first phase of the Distributed Design task in a top-down approach is the fragmentation phase. The fragmentation phase is the process of clustering in fragments the information accessed simultaneously by applications. In an object-oriented environment, the distributed design is a very complex task. First, because the semantic differences between relational and OO models inhibit a straightforward migration from existing relational distributed design algorithms to OO algorithms. Second, because it has to consider the existence of class methods and complex relationships (such as the "is-a" relationship), in addition to access from applications to complex objects and multiple relationships between classes. Third, because of OO operations access patterns: while relational operations are only *set oriented*, OO operations are *pointer based*, and therefore may have a dual nature involving both set operations (search over class extensions) and navigation (traversals) [12].

In our previous work [1,2], we have presented an analysis algorithm for assisting distributed designers in fragmenting object oriented databases. The algorithm performs an analysis on some useful information about the database schema and application frequencies in order to indicate the most adequate fragmentation technique (horizontal, vertical or mixed) for each class in the database schema.

In this paper, we show a rule-based implementation of our developed analysis algorithm that will be used as background knowledge when trying to discover a new revised algorithm through the use of a machine learning technique: Inductive Logic Programming (ILP) [14]. We intend to obtain a revised algorithm that may reflect important issues to the Distributed Design of Object Oriented Databases (DDOODB) that may be implicit, that is, not yet discovered by any of the proposed distributed design algorithms in the literature. In the present knowledge-based approach, we will represent the initial algorithm as a set of rules and perform a fine-tuning of it, thus discovering a new set of rules that will represent the revised algorithm. This new set of rules will represent a revised analysis algorithm that will propose optimal (or near to optimal) fragmentation schema with improved performance. In other words, we intend to perform Data Mining considering available database schema information as a test bed to produce optimal distributed database schema as an output.

The organization of this work is as follows: the next Section presents some definitions from the literature regarding the Distributed Design task and identifies some difficulties which motivated the use of a knowledge based approach for this problem. Section 3 shows the state of the art in the Distributed Design research area and in the ILP field. The analysis algorithm is described in Section 4. Section 5 discusses the use of ILP to revise the analysis algorithm. Finally, Section 6 concludes this paper.

2 Background

Distributed Design involves making decisions on the fragmentation and placement of data across the sites of a computer network [16]. In a top down approach, the distributed design has two phases: fragmentation and allocation. The fragmentation phase is the process of clustering in fragments the information accessed simultaneously by applications, and the allocation phase is the process of distributing the generated fragments over the database system sites. To fragment a class, it is possible to use two basic techniques: vertical fragmentation and horizontal fragmentation. In an object oriented (OO) environment, horizontal fragmentation distributes class instances across the fragments, which will have exactly the same structure but different contents. Thus, a horizontal fragment of a class contains a subset of the whole class extension. On the other hand, vertical fragmentation breaks the class logical structure (its attributes and methods) and distributes them across the fragments, which will logically contain the same objects, but with different structures. It is also possible to perform mixed fragmentation on a class, combining these two techniques. Horizontal fragmentation is usually subdivided in primary and derived fragmentation to address the relationship between entities: the primary horizontal fragmentation is applied on owner entities [9,16], while the derived fragmentation is applied on member entities according to the owner fragmentation.

The DDOODB is a very complex task. First, because the semantic differences between relational and OO models inhibit a straightforward migration from existing relational distributed design algorithms to OO algorithms. Second, because it has to consider the existence of class methods and complex relationships (such as the “is-a” relationship), in addition to application access to complex objects and multiple relationships between classes. Third, because of OO operations access patterns: while relational operations are only *set oriented*, OO operations are *pointer based*, and therefore may have a dual nature involving both set operations (search over class extensions) and navigation (traversals) [12].

This dual nature of access patterns has made object clustering a hard task to the OO distributed database designer. Usually, the components of a complex object are physically clustered together in the disk, while in the other hand a class with a large extension groups its own objects together. This may generate some conflicts to the OODBMS object clustering policy, and we believe that this object clustering policy is strongly related to the fragmentation process. While derived horizontal fragmentation privileges navigational access (from a complex object to its components), vertical fragmentation favors the class extension access and the use of class attributes and methods, by removing irrelevant data accessed by operations. Therefore, both fragmentation techniques should coexist for different classes in the distributed design. Also, there are some cases in which the best option is to perform horizontal and vertical fragmentation in a class simultaneously. We believe that algorithms that force all classes to have the same fragmentation policy (either horizontal or vertical), like the ones proposed in the literature, will end up having unsuitable fragmentation for classes having different access patterns, incurring in bad performance.

3 Related Work

Many researchers have worked on distributed design in the relational model, including ÖZSU and VALDURIEZ [16], and NAVATHE *et al.*[15]. In the same way, there are many works demonstrating the importance of the Distributed Design of Object Oriented Databases (DDOODB) to improve performance of non-conventional applications that manipulate large volumes of data [3,9,10,17]. However, despite the recent proposals for the DDOODB in the literature, a consensus is far from reachable about what is the best fragmentation technique (vertical, horizontal or mixed) to be applied in each class of the database schema. KARLAPALEM *et al.* [10] describe different aspects of a distributed object oriented database system that are critical to the distributed design process. The works from EZEIFE and BARKER [9] and SAVONNET *et al.* [17] propose algorithms for horizontal fragmentation of all classes in an OODB, while BELLATRECHE *et al.* [3], EZEIFE and BARKER [8] and MALINOWSKI [13] address only vertical class fragmentation for all classes in an OODB.

In our previous work [1,2], we have identified the need of an Analysis Phase in the class fragmentation process of object oriented databases. We have presented an algorithm for this Analysis Phase that was responsible for assisting distributed designers in choosing the best fragmentation schema, considering horizontal, vertical and mixed (horizontal and/or vertical) fragmentation of classes. The algorithm performs an analysis on some useful information about the database schema and application frequencies in order to indicate the most adequate fragmentation technique for each class in the database schema. By using it, the distributed designer will know, *a priori*, the most adequate fragmentation technique for each class, and will be able to choose his/her preferable algorithm to actually perform vertical and horizontal fragmentation in the corresponding classes to define fragments.

4 The algorithm for the Analysis Phase of the Class Fragmentation Process

This Section presents an overview of the whole fragmentation process of Object Oriented Databases (OODBs) illustrated in Figure 1, and describes the algorithm for the Analysis Phase of the Class Fragmentation Process that was proposed in our previous work [2].

The Analysis Phase algorithm considers issues about the database structure and user operations (obtained from the database designer or from the Global Conceptual Schema) to decide on the most adequate fragmentation strategy (horizontal and/or vertical) for each class in the schema. Information considered in this phase include **class** and **operation characteristics**:

For each class in the schema, the system needs to know:

- its relationships, their cardinalities and the referred classes of each;
- the existence or not of a collection of objects representing the class extension;
- the estimated size of the class extension (small, medium or large), compared to other classes in the schema;

For each operation running over the database, the system needs to know:

- the accessed class path, in order to classify it (extension or navigation operation);
- its estimated execution frequency, to determine its priority on the fragmentation process;

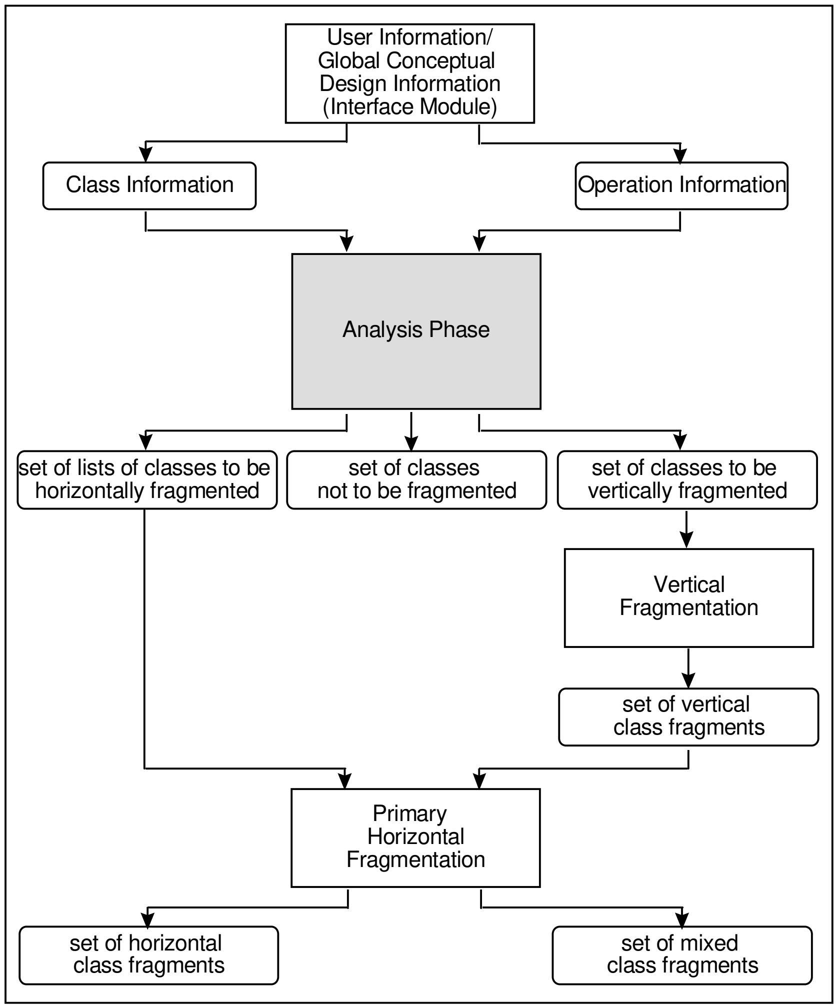

Figure 1. A framework for class fragmentation in Distributed Design of OODBs

The Analysis Phase algorithm proceeds as follows: operations are sorted in a descending way according to their execution frequency, thus priority is given to the most frequent operations. Operations are then considered one at a time, and for each operation the system chooses the best fragmentation technique to be assigned to the classes accessed by it (C(O)) according to its classification:

- if it is an extension operation, this class is included in the set of classes for vertical fragmentation Cv
- if it is a navigation operation, the list of classes C(O)) is included in the set of list of classes for horizontal fragmentation Ch

To help the decision between primary and derived fragmentation, the analysis algorithm includes in the horizontal set lists of related classes, instead of isolated ones. Those lists reflect the structure of the navigation paths accessed by the most frequent operations. However, when considering a navigation operation, there is the possibility of one of the classes in the list be already in C_h. To address this conflicting situation, the algorithm ensures that the lists inserted in the horizontal set do not contain any intersection among its non-root classes. When the algorithm tries to include in the horizontal set a non-root class C that already belongs to another list, then it proceeds to what we call **conflict analysis**. The conflict is solved based on the class clustering dependency classification, in the following way:

Given two navigational paths that conflicts at class C, (that is, when the classes before C in the paths are different), conflict analysis will decide which path will guide the fragmentation of class C. This decision is made according to the clustering dependency (as defined in [2]) between C and the classes of the two paths. Let N_1 and N_2 be two conflicting navigational paths accessed by two operations with a very close execution frequency:

$$N_1 = (C_i, C_j, \ldots, X, Y, \ldots, C_n) \text{ e } N_2 = (C_p, C_q, \ldots, Z, Y, \ldots, C_m)$$

For N_1 and N_2 to conflict, we have $X \neq Z$. The system decides if Y will be derived fragmented according to X or Z considering clustering dependencies from Y to X and from Y to Z. The priority of clustering dependencies is the following:

- non-shared dependent (**high priority**)
- shared dependent (**medium priority**)
- independent (**low priority**)

If clustering dependencies from Y to X and from Y to Z are the same, choose the navigational path of the most frequent operation.

Once the system has chosen for one of the paths (N_1, for example), N_2 will be broken, that is, Y and its following classes will be removed from it. The final navigational paths will be:

$$N_1 = (C_i, C_j, \ldots, X, Y, \ldots, C_n) \text{ e } N_2 = (C_p, C_q, \ldots, Z)$$

Those classes on the intersection of the horizontal and vertical sets (notice that only root classes in the horizontal set may be included in the vertical set) will eventually proceed to mixed fragmentation. In this case, the algorithm chosen for horizontal fragmentation will be performed on the vertical fragments of those classes.

5 Using Machine Learning Techniques for Theory Revision in the DDOODBs

Although the analysis algorithm has produced very good results when applied to some examples, as it will be shown in Section 5.2, some of its conjectures are only intuitive, since it is based on a set of heuristics obtained from experimental results, rather than from a formal theory. Examples of that are the arbitrary decisions made by the conflict analysis module to resolve conflicting situations as defined in the previous Section. Therefore, in order to obtain more reliable results, this work contributes by discussing a novel approach for revising our analysis algorithm through the use of a machine learning method - Inductive Logic Programming (ILP) [18]. This revision process is called Inductive Distributed Design of OODBs, and it will perform a fine tuning of our initial set of rules, thus discovering a new set of rules that will represent the revised algorithm. Some researchers have worked on proposals for inductive database design, but not in the same context. For example, BLOCKEEL and DE RAEDT [19] presented an approach for the inductive design of deductive databases, based on the database instances to define some intentional predicates. However, as far as we are concerned, performing theory revision in algorithms for Distributed Database Design is a novel approach in the area.

The idea of using knowledge-based neural networks (NN) to revise our background knowledge was first considered. There are in the literature many approaches for using NN in a theory revision process using propositional rules, such as KBANN [20] and CIL^2P [21]. However, due to the existence of function symbols in our analysis algorithm (such as lists) that could not be expressed through propositional rules, we needed a more expressive language, such as first-order Horn clauses. Since the representation of a more expressive language (such as first-order Horn clauses) in NN is still an open research problem, we decided to work with another machine learning technique - Inductive Logic Programming (ILP) [18].

According to MITCHEL [14], the process of ILP can be viewed as automatically inferring PROLOG programs from examples and, possibly, from background knowledge. In [14], it has been pointed out that machine learning algorithms that use background knowledge, thus combining inductive with analytical mechanisms, obtain the benefits of both approaches: better generalization accuracy, smaller number of required training examples and explanation capability.

Our main objective here is to show the viability of performing a revision process in order to obtain a better fragmentation algorithm. We do not intend to propose the best fragmentation algorithm ever possible. We concentrate here on the process of revising a DDOODB algorithm through Knowledge Discovery techniques, rather than on a final product. As a matter of fact, we could apply this knowledge-based approach to the DDOODB problem in any of the available fragmentation algorithms from the literature.

The resulting algorithm performance will depend not only on the quality of the background knowledge, but also on the quality of the examples considered in the training phase, as in conventional Machine Learning algorithms. Therefore, we needed a set of validated fragmentation schema with good performance when applied on a distributed database. However, such a set of optimal fragmentation schema is not available in the literature, since it is a current research topic. We decided to work on some scenarios already used as examples in the literature.

5.1 A Rule-Based implementation of the Analysis Phase algorithm

In our DDOODB problem, we used our analysis algorithm from [2] as our background knowledge. The overall structure of our set of rules is on Figure 2. We have implemented our algorithm as a set of first-order Horn clauses. Some of these clauses are illustrated in Figures 3 and 4. This set of rules constitutes a very good starting point for the ILP process to obtain the revised algorithm. The predicate chooseFragmentationMethod will represent the target predicate in our revision process. But we still needed the set of training examples to proceed a theory revision of our background knowledge.

Our set of training examples is being derived from several works in the literature. We are extracting from each selected work two sets of facts, one representing the initial database schema and another representing the desired fragmentation schema. It is important to notice that, as we did not have as many available examples in the literature as it would be desired, the background knowledge will play a major role in the ILP learning process.

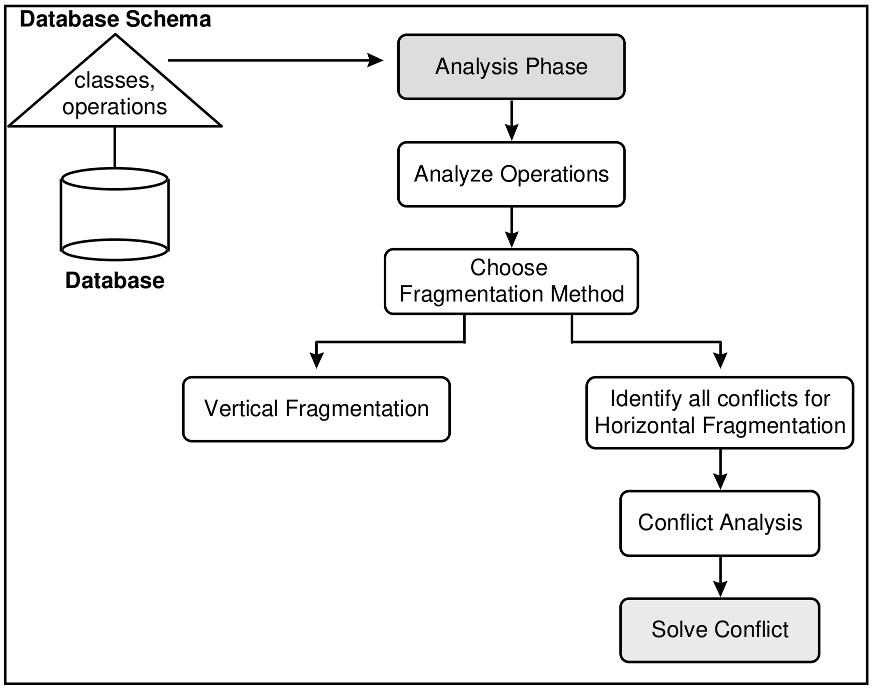

Figure 2. The overall structure of our set of rules for the class fragmentation

```
% ------------------------------------------------------
% analysisPhase( DataFile+, NewCh-, NewCv- )
% This will load input data from the file "DataFile"
% (such as the schema classes, the operations and their
% accessed classes) and build two sets of classes (NewCh,
% NewCv) representing respectively the classes to be
% horizontally and vertically fragmented.
% ------------------------------------------------------
analysisPhase( DataFile, Ch, Cv ) :-
   write( '************** BEGIN *****************' ), nl,
   consult( DataFile ),
   findall( X, operation( X, _ ), Operations ),
   bubbleSortDescending( Operations, OrderedOperations ),
   analyzeOperations( OrderedOperations, [], [],
                      TempCh, TempCv ),
   list_to_set( TempCh, Ch ),
   list_to_set( TempCv, Cv ),
   abolish(class/1),
   abolish(cardinality/2),
   abolish(relationship/4),
   abolish(operation/2),
   abolish(accessedClasses/2),
   abolish(accessFrequency/2).
```

Figure 3. The starting point of our analysis algorithm as a set of rules

```
% ----------------------------------------------------------
% chooseFragmentationMethod( Oi+, Ch+, Cv+,
%                            NewCh-, NewCv- )
% NewCh and NewCv will receive modified versions of Ch
% and Cv (actually, only one of them will be modified) to
% reflect the chosen fragmentation method for the accessed
% classes of operation Oi.
% ----------------------------------------------------------
chooseFragmentationMethod( Oi, Ch, Cv,
                           Ch, [RootClass|Cv] ) :-
   classification( Oi, extension ),
   accessedClasses( Oi, [RootClass|_] ),
   write( 'accessed classes ' : [RootClass|_] ), nl,
   cardinality( RootClass, large ),
   \+ isIn( RootClass, Cv ),
   write( RootClass:'will be vertically fragmented' ), nl.
chooseFragmentationMethod( Oi, Ch, Cv,
                           NewCh, Cv ) :-
   classification( Oi, navigation ),
   accessedClasses( Oi, ClassPath ),
   write( 'accessed classes ' : ClassPath ), nl,
   write( ClassPath:'will be horizontally fragmented,
                         if possible' ), nl,
   operation( Oi, F ),
   assert( accessFrequency( [ClassPath], F ) ),
   write( 'removing All Conflicts...' ), nl,
   removeAllConflicts( Ch, Cv, ClassPath, NewCh ).
chooseFragmentationMethod( _, Ch, Cv, Ch, Cv ).
```

Figure 4. A rule to choose the most adequate fragmentation technique for the classes accessed by an operation

5.2 An Example

In this Section, our proposed algorithms for the distributed design of OODBs are applied on an example extracted from [22]. This example illustrates a very common scenario in current organizations. We present the output of the application of our algorithm in this example, and evaluate the indicated fragmentation techniques for each class against some performance comments from the original work.

5.2.1 The Company Database Schema

Figure 5 shows a rule representation for the Company database schema as originally described in [22]. The original relations were adapted to address the object oriented data model. All the necessary information for the Analysis Phase of the Fragmentation Process is available in this representation, which will be used to build an example for our revision process.

```
/*****************************************************
DDOODBMS Example
Extracted from: Elmasri, R., Navathe, S., "Fundamentals of
                Database Systems", The Benjamin/Cummings
                Publishing Company, Inc., 1989, pp.619-622
*****************************************************/
class( employee ).
class( department ).
class( project ).
class( location ).
class( dependent ).

cardinality( employee, large ).
cardinality( department, small ).
cardinality( project, large ).
cardinality( location, small ).
cardinality( dependent, large ).

relationship( employee, department, worksIn, '1' ).
relationship( department, employee, holds, 'N' ).
relationship( employee, project, worksOn, 'N' ).
relationship( project, employee, isCarriedOutBy , 'N' ).
relationship( employee, employee, isManagedBy, '1' ).
relationship( employee, dependent, has, '0,N' ).
relationship( dependent, employee, dependsOn, '1' ).
relationship( department, location, isLocatedAt, '1' ).
relationship( location, department, holds, 'N' ).
relationship( project, location, isLocatedAt, '1' ).

operation( employeesWhoWorkInDepartmentAtLocation, 100 ).
operation( projectsControlledByDepartmentAtLocation, 100 ).
operation( mainEmployeeInformation, 85 ).
operation(insuranceInformation, 85 ).

accessedClasses(employeesWhoWorkInDepartmentAtLocation,
               [location, department, employee] ).
accessedClasses(projectsControlledByDepartmentAtLocation,
               [location, department, project] ).
accessedClasses(mainEmployeeInformation, [employee] ).
accessedClasses(insuranceInformation,[employee, dependent].
```

Figure 5. A rule representation for the Company database schema: input for the Analysis Algorithm

5.2.2 Applying the algorithm

Figure 6 presents the output of the application of our Analysis algorithm in the Company Database Schema. The final lists of classes Ch and Cv are, respectively, [[location, department, project],[location, department, employee]] and [employee].

```
?- analysisPhase('navathe.pl',Ch,Cv).
************* BEGIN ****************
navathe.pl compiled, 0.00 sec, 2,652 bytes.
Ch =[]
Cv =[]
Analyzing Operation :
             projectsControlledByDepartmentAtLocation
Going to Choose Fragmentation Method for classes accessed
             by
:projectsControlledByDepartmentAtLocation
accessed classes :[location, department, project]
[location, department, project]:will be horizontally
             fragmented, if possible
removing All Conflicts...
Ch =[[location, department, project]]
Cv =[]
Analyzing Operation
:employeesWhoWorkInDepartmentAtLocation
Going to Choose Fragmentation Method for classes accessed
             by :employeesWhoWorkInDepartmentAtLocation
accessed classes :[location, department, employee]
[location, department, employee]:will be horizontally
                          fragmented, if possible
removing All Conflicts...
Ch =[[location, department, project],
     [location, department, employee]]
Cv =[]
Analyzing Operation :insuranceInformation
Going to Choose Fragmentation Method for classes accessed
                          by :insuranceInformation
accessed classes :[employee, dependent]
[employee, dependent]:will be horizontally fragmented, if
possible
removing All Conflicts...
Ch =[[location, department, project],
     [location, department, employee],
     [employee, dependent]]
Cv =[]
Analyzing Operation :mainEmployeeInformation
Going to Choose Fragmentation Method for classes accessed
                          by :mainEmployeeInformation
accessed classes :[employee|_G882]
employee:will be vertically fragmented

Ch = [[location, department, project],
      [location, department, employee],
      [employee, dependent]]
Cv = [employee]

Yes
?-
```

Figure 6. Applying our Analysis algorithm in the Company Database Schema

5.2.3 Evaluating the Resulted Distributed Database

Considering performance comments from [22], we may affirm that our analysis algorithm has produced a fragmentation schema that improves performance. By applying the results from our algorithm, the distributed designer will perform primary horizontal fragmentation on class Location, and derived fragmentation of classes Department (according to class Location), Project (according to class Department), Employee (according to class Department) and Dependent (according to class Employee). Also, class Employee will be vertically fragmented (thus resulting in mixed fragmentation).

Intuitively, the reader may see that the first three traversals (the one that takes information for the Employees who work in a Department located at a specific Location; the one that takes information for the Projects controlled by a Department located at a specific Location; and the one that takes dependent information from Employees for insurance purposes) will have their performance improved, since they will perform a direct access to only one fragment each, and this will reduce communication overhead during their execution. Also, the last query, that will access main employee information (such as the attributes name, security number and salary) will benefit from vertical fragmentation that will probably be performed in this class in the vertical fragmentation phase of the DDOODB. This benefit will result from the elimination of irrelevant data accessed by it.

6 Conclusions

Distributed and parallel processing on OODBMS may improve performance for non-conventional applications that manipulate large volumes of data. This is addressed by removing irrelevant data accessed by queries and transactions and by reducing the data exchange among sites [10], which are the two main goals of the Distributed Design of Databases.

Distributed Design involves making decisions on the fragmentation and placement of data across the sites of a computer network [16]. Since Distributed Design is a very complex task in the context of the OO data model, we have developed an analysis algorithm to assist distributed designers in the fragmentation phase of OO databases. The analysis algorithm decides the most adequate fragmentation technique to be applied in each class of the database schema, based on some heuristics.

In this paper, we have presented a knowledge-based approach to the DDOODB problem. Our developed analysis algorithm is implemented as a set of rules and will be used as background knowledge when trying to discover a new revised algorithm through the use of ILP [18]. In our approach, we will perform a fine-tuning of our initial algorithm (represented as a set of rules), thus discovering a new set of rules that will represent the revised algorithm. This new set of rules will represent a revised analysis algorithm that will propose optimal (or near to optimal) fragmentation schema with improved performance. In other words, we intend to perform Data Mining considering available database schema information as a test bed to produce optimal distributed database schema as an output.

We have presented an example of applying the proposed set of rules using a Company Database Schema from [22], and the fragmentation schema obtained. The resulted fragmentation schema offers a high degree of parallelism together with an important reduction of irrelevant data.

We have presented the main ideas embedded in a novel approach for refining the analysis algorithm through the use of a machine learning method - Inductive Logic Programming (ILP). This approach (called Inductive Distributed Design of OODBs) performs a knowledge discovery/revision process using our set of rules as background knowledge. The objective of the work is to uncover some previously unknown issues to be considered in the distributed design process.

Our main objective was to show the viability of performing a revision process in order to obtain better and better fragmentation algorithms. We do not intend to propose the best fragmentation algorithm ever possible. We concentrate here on the process of revising a DDOODB algorithm through Knowledge Discovery techniques, rather than on a final product.

Although we have addressed the problem of class fragmentation in the DDOODB context, an important future work is the use of the same inductive learning approach in other phases of the Distributed Design (such as the allocation phase), as well as in the Database Design itself, possibly using another data models (relational or deductive). Also, the resulting fragmentation schema obtained from our revised algorithm may be applied to fragment the database that will be used in the work of PROVOST and HENNESSY [23], which proposes the use of distributed databases in order to scale-up data mining algorithms.

References

[1] Baião, F., Mattoso, M., 1997, "A Mixed Fragmentation Strategy for Distributed OO Databases", In: Proc. of The Second Workshop on CSCW in Design, Bangkok, Thailand, November, pp. 42-48

[2] Baião, F., Mattoso, M., 1998, "A Mixed Fragmentation Algorithm for Distributed Object Oriented Databases", to appear in: *Proceedings of the 9th International Conference on Computing and Information*, Winnipeg, Canada, June

[3] Bellatreche, L., Simonet, A., Simonet, M., 1996, "Vertical Fragmentation in Distributed Object Database Systems with Complex Attributes and Methods". 7th International Workshop on Database and Expert Systems Applications (DEXA'96) , Zurich, Switzerland

[4] Carey, M., De Witt, D., Naughton, J., 1993, "The 007 Benchmark". In: Proc of 1993 acm sigmod Int. Conf. on Management of Data, vol 22, n° 2, Washington DC, pp. 12-21

[5] Chakravarthy, S., 1997, Panel on: Bridging the Gap between Cooperative Information Systems and Database Systems, Second IFCIS International Conference on Cooperative Information Systems (CoopIS'97), South Caroline, USA

[6] Chen, Y., Su, S., 1996, "Implementation and Evaluation of Parallel Query Processing Algorithms and Data Partitioning Heuristics in Object Oriented Databases", Distributed and Parallel Databases, v. 4(2), pp. 107-142

[7] Cluet, S., Delobel, C., 1992, "A General Framework for the Optimization of Object-Oriented Queries". In: Proceedings of the 1992 ACM SIGMOD, vol. 21, issue 2, San Diego, California, pp. 383-391

[8] Ezeife, C., Barker, K., 1994, Vertical Class Fragmentation in a Distributed Object Based System, Technical Report 94-03, Department of Computer Science, University of Manitoba

[9] Ezeife, C., Barker, K., 1995, "A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System", Distributed and Parallel Databases, vol. 3, n° 3, pp. 247-272

[10] Karlapalem, K., Navathe, S., Morsi, M., 1994, "Issues in Distribution Design of Object-Oriented Databases". In: Özsu, M. et. al (eds), Distributed Object Management, Morgan Kaufman Publishers

[11] Lima, F. Mattoso, M., 1996, "Performance Evaluation of Distribution in OODBMS: a Case Study with O2" In: Proc IX Int. Conf on Parallel & Distributed Computing Systems, France, pp.720-726

[12] Maier, D., Graefe, G., Shapiro, L. et al., 1994, "Issues in Distributed Object Assembly". In: Özsu, M., Dayal, U., Valduriez, P. (eds), Distributed Object Management, Morgan Kaufmann Publishers

[13] Malinowski, E., 1996, Fragmentation Techniques for Distributed Object-Oriented Databases, MSc. Thesis, University of Florida

[14] Mitchell, T., 1997, Machine Learning, McGraw-Hill Companies, Inc

[15] Navathe, S., Karlapalem, K., Ra, M., 1995, "A Mixed Fragmentation Methodology for Initial Distributed Database Design", Journal of Computer and Software Engineering, vol. 3, n° 4

[16] Özsu, M., Valduriez, P., 1991, Principles of Distributed Database Systems, New Jersey, Prentice-Hall

[17] Savonnet, M., Terrasse, M., Yétongnon, K., 1996, "Using Structural Schema Information as Heuristics for Horizontal Fragmentation of Object Classes in Distributed OODB", In: Proc IX Int. Conf on Parallel & Distributed Computing Systems, France, pp. 732-737

[18] Lavrac, N. and Dzreroski, S., 1994, Inductive Logic Programming: Techniques and Applications, Ellis Horwood.

[19] Blockeel, H, de Raedt, L., 1996, "Inductive Database Design" In: Proc of the Int. Symposium on Methodologies for Intelligent Systems (ISMIS'96)

[20] Towell, G., Shavlik, J., 1994, Knowledge-Based Artificial Neural Networks, *Artificial Intelligence*, 70 (1-2), pp. 119-165.

[21] Garcez, A., Zaverucha, G. and Silva, V. N., 1997, "Applying the Connectionist Inductive Learning and Logic Programming System to Power System Diagnosis," In: Proceedings of the IEEE International Conference on Neural Networks (ICNN-97), Houston, Texas, USA, Vol.1, pp.121-126.

[22] Elmasri, R., Navathe, S., 1989, Fundamentals of Database Systems, The Benjamin/Cummings Publishing Company, Inc.

[23] Provost, F., Hennessy, D., 1996, "Scaling-Up: Distributed Machine Learning with Cooperation", In: Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96), Portland, Oregon