Process-Based Database Support for the Early Indicator Method

Christoph Breitner, Jörg Schlösser
University of Karlsruhe
Institute for Program Structures and Data Organization
76128 Karlsruhe, Germany
{breitner,joerg}@ira.uka.de

Rüdiger Wirth
Daimler-Benz AG, Research and Technology F3S/E,
P. O. Box 23 60,
D-89013 Ulm, Germany
wirth@dbag.ulm.DaimlerBenz.COM

Abstract

In Wirth & Reinartz (1996), we introduced the *early indicator method*, a multi-strategy approach for the efficient prediction of various aspects of the fault profile of a set of cars in a large automotive database. While successful, the initial implementation was limited in various ways. In this paper, we report recent progress focusing on performance gains we achieved through proper process-based database support. We show how intelligent management of the information collected during a KDD process can both make the task of the user easier and speed up the execution. The central idea is to use an object oriented schema as the central information directory to which data, knowledge, and processes can be attached. Furthermore, it enables the automatic exploitation of previously stored results. Together with the query-shipping strategy, this achieves efficiency and scalability in order to analyze huge databases.

While we demonstrate the usefulness of our solution in the context of the early indicator method, the approach is generally applicable and useful for any integrated, comprehensive KDD system.

Introduction

In Wirth & Reinartz (1996), we introduced a multi-strategy approach for the efficient prediction of various aspects of the fault profile of a set of cars in a large automotive database. We addressed the following problem: Do there exist sub populations of cars, so called *early indicator cars* (EICs), which behave like the whole population of cars at a later point in time for certain aspects? If yes, how can they be characterized by easily observable attributes?

The EIC approach was prototypically implemented and the feed-back of the domain experts was positive. Nevertheless, the initial implementation was limited in various ways:

- The database access and the different techniques for the various steps were only loosely coupled. The prototype consisted of a multitude of database operations and algorithms connected by additional data transformation steps. In particular, most of the effort had to be spent on putting together operations for the pre-processing phase, which made experimentation with different data sets and parameters very costly.
- The same or similar results (e.g., data sets, distributions, etc.) were computed several times from the original large database. Since this involved many joins over several tables, re-computing again was very time-consuming. If the user wanted to avoid this re-computation, he had to store and manage the results himself. But then it was hard to keep an overview of the various data sets, the parameters used in constructing the data sets, and - most importantly - the results which were derived from the data sets.
- During the process of detecting EICs, documentation of the experiments and results had been neglected because of the considerable overhead and the lack of a adequate methodology. This made it very hard to remember which experiments had been carried out, which features had been computed, and which results or experience had been gained.

These issues are quite common in other KDD projects as well. Therefore, we started to develop a general purpose KDD system, CITRUS (Wirth et al., 1997), to address these and other issues. We chose the existing commercial knowledge discovery tool CLEMENTINE (Shearer, 1996) as a starting point. It integrates multiple data mining algorithms and tools for data access, visualization, and data preprocessing. To perfom a multi step discovery task, e.g., consisting of multiple preprocessing steps before a learning algorithm, the user simply has to connect corresponding nodes for the steps according to the data flow on the drawing pane, edit the parameters through pop-up dialoques, and start the execution of the resulting graph ("stream" in CLEMENTINE terminology).

Here, we report recent progress in CITRUS, focusing on the role database support can play to achieve more comfortable data handling, significant speed up, and improved documentation facilities. In the next two sections, we discuss how an object-oriented schema can simplify the modeling of streams and the management of results. In section 4 we describe the optimized execution of streams to achieve significant speed-up. Finally, we discuss the realization of the ideas presented in this paper and conclude.

Simplifying the Modeling of the Streams

Most of the difficulties with modelling the derivation processes arise during the data preparation phase, which is usually the most time consuming phase in a KDD project (cf. Famili et al., 1997). The necessary streams in our application scenario are very complex and difficult to model because of complex preprocessing of the data. Figure 1 depicts a stream fraction for computing some of the new features, namely the number of radiator faults which occurred within 365 days of operation, and the average costs, for a set of cars at a certain time (data set T_g). Altogether, more than 60 nodes were needed to compute all desired features.

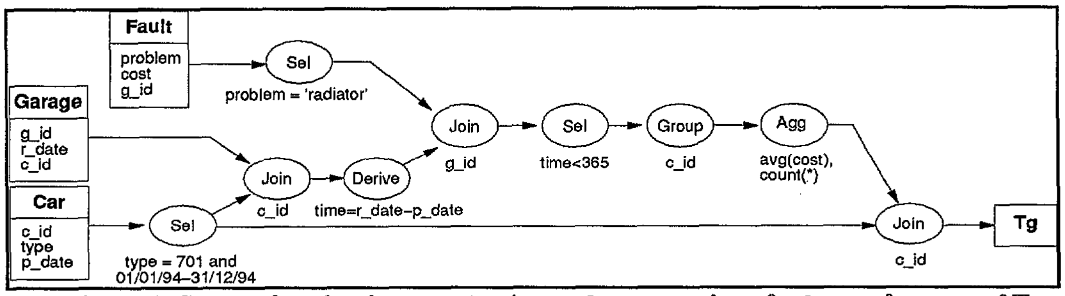


Figure 1: Stream fraction for the selection and construction of relevant features of T_g

We claim that an object-oriented data model together with a set of generic, set-oriented operators can simplify and speed up the modelling of streams significantly. The generic operators represent different ways to focus on relevant data, e.g, to select objects or attribute values which meet a given condition, or to combine multiple objects or attribute values. To follow the relationships between object types path-expressions can be used.

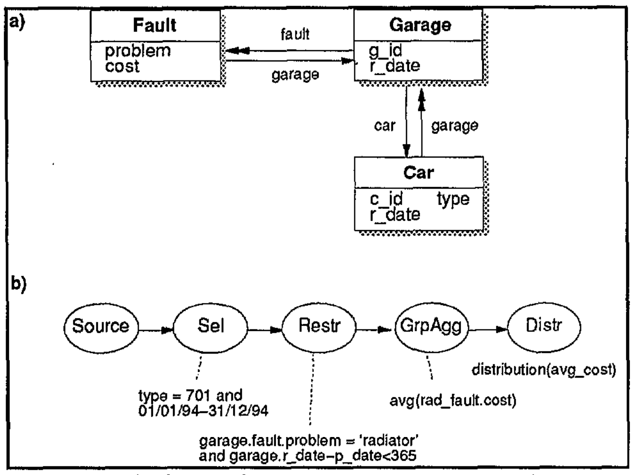


Figure 2: Object-oriented stream for computing new features for T_g

Figure 2a) and b.) show the object-oriented schema (arrows indicate complex attributes, double arrows represent multi-valued attributes), and the resulting object-oriented stream corresponding to figure 1. The stream consists of four nodes, a source-node which chooses the object-set Car as an initial data set, a select-node which specifies the car-type and the production period, a restrict-node which restricts the multi-valued attribute garage.fault to radiator problems which occurred within a given time of operation, as well as a group-aggregate-node which computes the number of faults and the average costs.

Storage and Access to Results

In order to manage all the information generated during the KDD process, e.g., data sets, attributes, streams, graphs, rule-sets, etc., we use the object-oriented schema as the *central information directory* to which data, knowledge and streams can be attached to. The evolution of the information directory is done completely automatically at execution time, with each operator leaving a trace within the schema. The direction of the trace and, thus, the place of the anchoring point for attaching information, is determined by using operator specific subsumption mechanisms.

We explain our solution with an example. Initially, the schema might be the one shown in figure 3a). At execution time the select-node is matched against subsets of the initial data set, namely Car. Since Car has no subsets, a new data set Car_701 is introduced as a direct subset of Car (b). Along with the new schema element the derivation history is recorded. This is done by attaching the derivation step - which includes the selection condition needed for the matching - to the subset relation. Then the restrict and the group-aggregate operator lead to the introduction of new attributes rad_fault and avg_c as well as operators (c,d). Finally, an additional distribution-node, which computes the distribution of the costs for radiator faults is mapped against the schema. Again, the matching fails and the schema is extended by elements which reflect the knowledge and the derivation step.

The mechanism of repeated subsumption and schema evolution makes it possible to reconstruct streams and provide a fairly detailed documentation of all the intermediate results. It also represents a powerful way to *retrieve* already discovered results by simply specifying a stream and mapping it against the schema. If the matching of the last node in the stream succeeds, the requested knowledge has already been discovered and can be returned without having to execute the stream on the raw data.

Optimized Execution of Streams

In demanding applications like EIC one is often faced with data sets which easily reach volumes of tens of giga bytes and streams consisting of many data (memory) extensive operations. Instead of requiring the data to be stored in object-oriented database systems (Breitner et al. (1995)), we rely on RDBMS as a powerful platform for storing and accessing data. Our efforts for an efficient execution strategy of streams is driven by two basic ideas: *query-shipping* and *automatic usage of materializations.*
Instead of loading the data entirely into the system for the execution (data-shipping) we use, similar to IDEA (Selfridge et al., 1996) and INLEN (Kerschberg et al., 1992), the functionality of the underlying RDBMS. For the execution, a stream is partly replaced by adequate SQL queries which can be out-sourced and answered efficiently by the RDBMS. To do this, object-oriented stream fractions must be mapped to relational counterparts. This is done by traversing the schema, and, for each derived object type or derived attribute, successively transforming the corresponding derivation operations to the relational model (e.g., path expressions are mapped to join sequences) until the resulting SQL-statement for the corresponding schema elements (stream) is based on existing tables.
During this mapping of a stream, profitable materializations of intermediate results from streams executed earlier are considered automatically. Like the above mentioned knowledge pieces, the information about the availability of such materializations is simply attached to the corresponding schema elements. Thus, to find the relevant ones for our example in Figure 3 we first look at the materialization information of Car_701 and then at the information of its - in this case unique - superset Car. If different mappings are possible the most efficient alternative will be chosen by estimating the execution costs.
Figure 4 shows a possible execution plan for our example if the result of the **Restr**-operator was materialized earlier in the table Car_701_rad_fault. IMPORT-DBMS contains the SQL-statement which will be sent to the RDBMS. The corresponding result is used as input for (an internally realized) DISTR-program which creates a distribution for the values of avg_c.

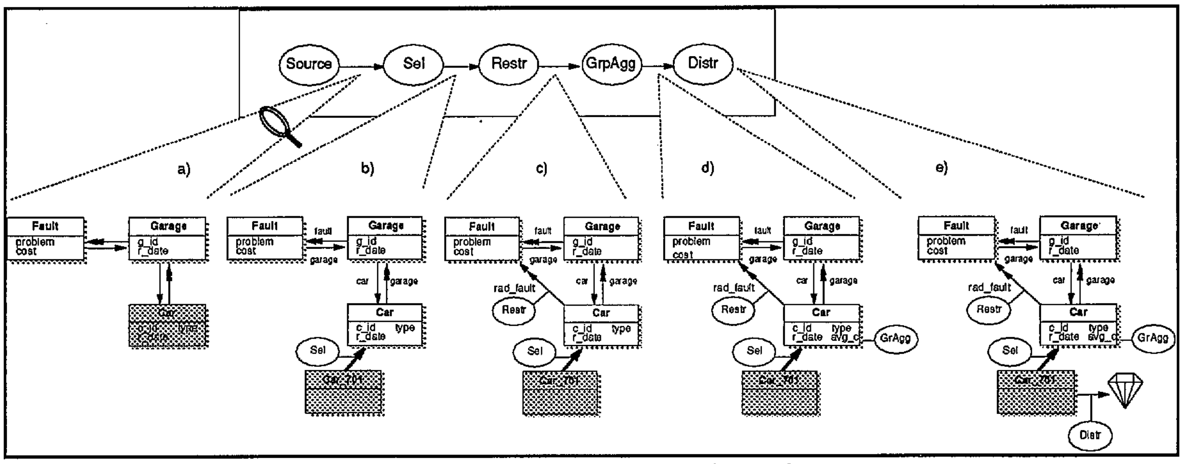
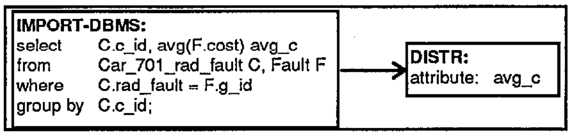


Figure 3: Mapping streams against the object-oriented schema


```
IMPORT-DBMS:
select      C.c_id, avg(F.cost) avg_c
from        Car_701_rad_fault C, Fault F
where       C.rad_fault = F.g_id
group by    C.c_id;

DISTR:
attribute:  avg_c
```
Figure 4: Execution plan for the example stream

Realization in Citrus

To realize the functionality mentioned in the previous sections, we extended CLEMENTINE by a new component, called *Information Management Component (IMC).* The typical information processing loop is as follows: First, a stream is modeled using the visual programming interface of CLEMENTINE which is then passed to the IMC where it is executed. Finally, the result is returned and presented to the user using appropriate visualizations of CLEMENTINE. The execution itself can be divided into three principal steps: schema mapping, plan generation and plan execution. Figure 5 shows the client-server architecture.
Schema mapping is done by the *schema manager*, and returns the schema objects, i.e., attributes, sets of objects, and knowledge pieces corresponding to the stream nodes. When streams with new process-nodes are to be executed,

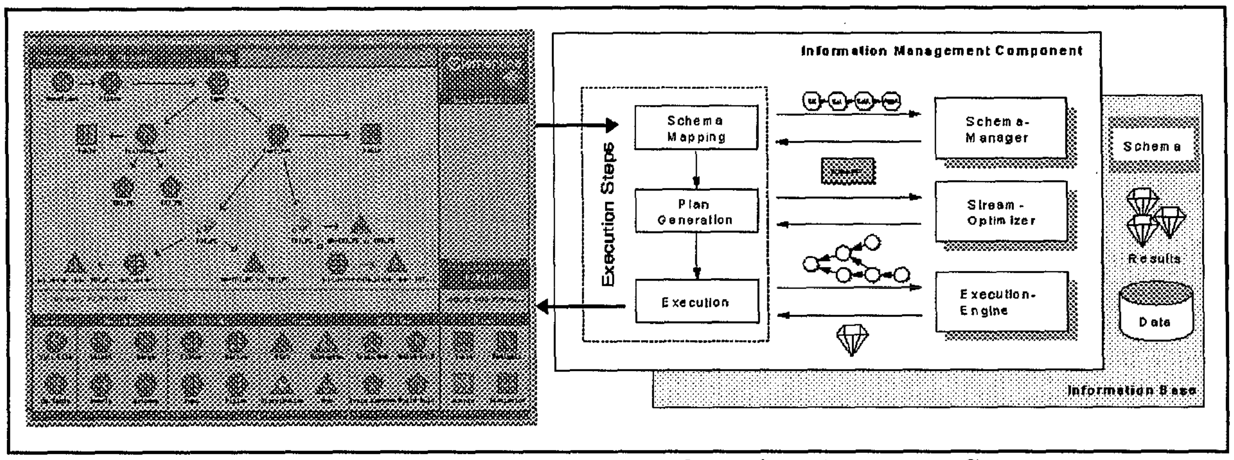


Figure 5: Visual Programming Interface and Information Management Component

the schema will evolve to cover newly derived attributes, object sets or results.

Following the schema mapping, the *stream optimizer* takes the stream and the corresponding schema objects as input and generates an optimized execution plan, which can contain both SQL-statements and external functions with no counterparts in RDBMS. In the simplest case the execution plan consists of a single 'load-file' instruction. This applies when the result has already been computed and stored during previous stream executions.

Last but not least, the actual plan execution is done by the *execution engine*, which takes an execution plan from the stream optimizer and executes it by using the query shipping strategy. The parts of the stream which have no SQL counterpart have to be executed by the execution engine by means of internal operators or external programs (e.g., like C4.5).

Concluding Remarks

In this paper we presented several approaches integrated into the IMC module of CITRUS to overcome many problems arisen due to the lack of an appropriate information management during a real world KDD process. We addressed these problems in the context a real application. Although the first prototype of the IMC is quite powerful a lot remains to be done. More sophisticated subsumption mechanisms and enhancements of the retrieval and documentation facilities are necessary. Furthermore, we are currently trying to automate the materialization of intermediate results to further shift the responsibility for execution efficiency from the user to the system. The idea is to analyze the process history condensed in the schema in order to estimate the future relevance of a data set, and manage the materializations on the basis of this measure.

References

Brachman, R.J., Selfridge, L., Terven, L., Altman, B., Halper, F., Kirk, T., Lazar, A., McGuiness, D., & Resnick, L. (1993). Integrated Support for Data Archaeology. *Proceedings 1993 AAAI Workshop on Knowledge Discovery in Databases*, pp. 197-212.

Breitner, C., Freyberg, A., & Schmidt, A. (1995). Toward a Flexible and Integrated Environment for Knowledge Discovery. *Workshop on Knowledge Discovery and Temporal Reasoning in Deductive and Object-Oriented Databases, KDOOD*, Singapore, pp. 28-35.

Famili, A., Shen, W-M., Weber, & R. Simoudis, E. (1997). Data Preprocessing and Intelligent Data Analysis. *Intelligent Data Analysis, Vol1, No.1*.

Kerschberg, L., Michalski, R.S., Kaufman, K.A., & Riberio, J.S. (1992). Mining for Knowledge in Databases: The INLEN Architecture, Initial Implementation and First Results. *Journal of Intelligent Information Systems 1 (1)*, pp. 85-113.

Selfridge, P.G., Srivastava, D., & Wilson, L.O. (1996). IDEA: Interactive Data Exploration and Analysis, *Intl. Conf. Management of Data (SIGMOD)*, pp. 24-34.

Shearer, C. (1996). User Driven Data Mining. *Unicom Data Mining Conference*, London.

Wirth, R., & Reinartz, T.P. (1996). Detecting Early Indicator Cars in an Automotive Database: A Multi-Strategy Approach. *Proceedings of the 2nd Int. Conference on Knowledge Discovery and Data Mining*, pp. 76-81.

Wirth, R. Shearer, C., Grimmer, U., Reinartz, T.P., Schloesser, J., Breitner, C., Engels, R., Lindner, G. (1997). Towards Process-Oriented Tool Support for KDD, *Proc. of 1st Intern. Conference on Principles of KDD*, Springer Verlag.