Towards Process-Oriented Tool Support for Knowledge Discovery in Databases

Rüdiger Wirth[1], Colin Shearer[2], Udo Grimmer[1], Thomas Reinartz[1], Jörg Schlösser[3], Christoph Breitner[3], Robert Engels[4], and Guido Lindner[1]

[1] Daimler-Benz AG, Research & Technology, F3S/E,
P.O. Box 2360, 89013 Ulm, Germany
[2] Integral Solutions Ltd.,
Berk House, Basingstoke, Hampshire, RG214RG, United Kingdom
[3] Univ. of Karlsruhe, Institute for Program Structures and Data Organisation,
76128 Karlsruhe, Germany
[4] Univ. of Karlsruhe, Institute AIFB,
76128 Karlsruhe, Germany

e-mail: wirth@dbag.ulm.DaimlerBenz.COM

Abstract. Knowledge Discovery in Databases (KDD) is currently a hot topic in industry and academia. Although KDD is now widely accepted as a complex process of many different phases, the focus of research behind most emerging products is on underlying algorithms and modelling techniques. The main bottleneck for KDD applications is not the lack of techniques. The challenge is to exploit and combine existing algorithms effectively, and help the user during all phases of the KDD process. In this paper, we describe the project CITRUS which addresses these practically relevant issues. Starting from a commercially available system, we develop a scaleable, extensible tool inherently based on the view of KDD as an interactive and iterative process. We sketch the main components of this system, namely an information manager for effective retrieval of data and results, an execution server for efficient execution, and a process support interface for guiding the user through the process.

1 Introduction

Knowledge Discovery in Databases (KDD) is currently a hot topic in industry and academia. Unfortunately, the focus of research behind most emerging products is on underlying algorithms and modelling techniques. From a practical point of view, the main bottleneck for KDD applications is not the lack of techniques. The challenge is to exploit and combine existing algorithms effectively.

Today, KDD projects are typically approached in an unstructured, ad hoc manner. In many cases, the approach taken is heavily influenced by tools already available in the organisation, which may be many and disparate. The user will often embark on an initial analysis or modelling task; the results of this may cause him to try a new technique; that in turn may suggest restructuring the data and running a new analysis; and so on. In the worst case, this is almost a

blind search; the user stumbles from step to step, driven by results of previous stages and gut feel as to what is an appropriate thing to do next. For small scale, relatively simple projects, this approach may well bear fruit - the user stumbling onto a good result. But the larger and more complex the application, the lower the likelihood of success.

At Daimler-Benz we were (and still are) actively involved in many KDD application projects. In almost all of them the lack of a methodology and proper tool support lead to minor or even major problems. For instance, we spent a lot of time shuffling around data and transforming them for different techniques, setting up experiments, adjusting parameters, trying to keep track of the progress, recomputing results somebody else computed before and so on. In the end, there were typically wasted resources and unnecessarily long development times.

Additionally, the result of a KDD application depends to a large degree on the persons doing the work. Everybody has his own area of expertise but also weak points, and so we sometimes end up with sub-optimal solutions when the person working on the project was not an expert in the method which would have produced a better result.

Problems like the ones mentioned above are quite common in KDD. We identified the lack of a usable process model with proper tool support and a sophisticated multi-paradigm toolkit as the main causes for these problems.

In this paper, we describe the ongoing project CITRUS addressing these practically relevant issues. The development of CITRUS is driven by applications. Therefore, we first describe one of the applications which is used to concretise the requirements of CITRUS. Then, we outline a process model which forms the conceptual backbone of CITRUS. In this paper, we focus on three major components of the system, the information manager, the execution server, and the process support interface and user guidance, which will be described subsequently. Finally, we sketch further extensions and conclude.

2 An Application Example

In Wirth & Reinartz (1996), we introduced a multi-strategy approach for the 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 which, in certain aspects, behave like the whole population of cars at a later point in time? If yes, how can they be characterised by easily observable attributes? In the following, we call such sub-populations early indicator cars (EICs).

The EIC method consists of three major steps: In the first step, the fault profile of the whole population of cars at a certain time T_g is characterised. A fault profile is a vague term which could be defined in various ways. In any case, it will be described in terms of fault relevant attributes.

In the second step, EICs need to be selected. For this purpose, we compute the values for the fault attributes at an earlier time T_{eic}. Those cars which fit the fault profile of the whole population at T_{eic} are considered to be EICs.

In order to generate a characterisation of the EICs in the third step, we have to take into account that EICs need to be identified at production time or shortly after. Therefore, we cannot use attributes that relate directly to the fault profile. EICs need to be characterised by easily observable attributes, like type and configuration of a car or areas where it was sold. This requires the use of two separate sets of attributes. One set is related to the fault profile and the other set contains attributes that can be observed at production time.

There are many degrees of freedom. For each of these steps, different techniques can be applied. Then there are many parameters which affect the outcome. These parameters include the learning period, the attributes for the fault profile and the configuration, the times T_g and T_{eic}, and the parameters of the learning techniques.

While the feedback from the end users was very positive, the initial implementation was limited in various ways:

- 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. Thus, experimentation with different data sets and parameters was very costly.
- The same or a 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 an 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.
- The process is very complex. An ordinary user who wants to perform experiments with different techniques, attributes, or data sets was quickly overwhelmed with the decisions required from him. The user was not supported in these decisions.

All of these limitations are addressed in Citrus. In fact, the requirements from the EIC realisation are among the main driving factors for Citrus. Breitner et al. (1997) and Engels et al. (1997) address information management and user guidance issues, respectively, of Citrus in the context of the EIC method.

3 A Process Model of KDD

All components of Citrus are based on the view of KDD as a highly interactive and iterative process which requires substantial human effort and skills (cf.

Brachman & Anand, 1994; Reinartz & Wirth, 1995; Adriaans & Zantinge, 1996). Here, we outline a process model we are developing. A full description is beyond the scope of this paper.

The life cycle of a KDD project consists of nine phases. Each phase can be broken down into tasks with input, output, and activities. Moving back and forth between different phases and/or tasks is typically required. It depends on the outcome of each phase which phase, or which task of a phase, must be performed next.

The *requirements and feasibility analysis phase* aims at the precise definition of the project objectives and the data mining goals. The final outcome of the requirements and feasibility analysis is a set of recommendations for the further proceeding and the preparation of subsequent phases as a skeleton process plan.

In the *domain analysis phase* the relevant knowledge on the application domain, on the data, on the features, and on the environment is analysed and documented. At the end of this phase, the initial process plan is refined.

The *data access phase* is mainly technical and provides the data set and the required knowledge which are used for subsequent phases.

The *preparation phase* contains all tasks that need to be performed before the application of modelling and discovery techniques. This includes data and feature selection as well as transformation and cleaning of data for exploration and analysis tools. The preparation phase is typically iterative and usually the most time-consuming part of a KDD project. In particular, the concrete activities are highly dependent on the techniques that will be used for modelling and discovery.

In the *exploration phase*, the main objective is to get familiar with the data, discover first insights into the data, or to identify interesting subsets of data and features to form hypotheses for hidden information.

Both the preparation and exploration phase involve the frequent generation of data sets and information about these data sets. In practice, many data sets and intermediate results are computed several times. For documentation purposes, it is not easy to keep track of what had been done. Later, we show how proper information management can help to avoid these difficulties.

In the *application of modelling and discovery techniques phase*, various data mining algorithms are selected and applied, and their parameters are calibrated to optimal values. Typically, there are several techniques for the same data mining goal. Some techniques have specific requirements on the form of data. Therefore, another data preparation and transformation step is often needed.

In the *interpretation and evaluation phase*, the data mining results are interpreted in business terms and evaluated according to the success and evaluation criteria specified in the requirements and feasibility analysis phase. Shortcomings need to be identified and, typically, new experiments will be set up.

When the project results meet the success criteria, the results must be deployed. Depending on the requirements, the *deployment phase* can be as simple as generating a report or as complex as realising and integrating a data mining analysis environment. Frequently, the deployment consists of the integration of

a predictive model in an existing environment. Finally, the *experience documentation phase* ensures that all experiences of the project and of the deployment of the data mining results are reported.

For all phases of the process, we shall supply techniques. The process model encourages the user to think about the tasks before techniques are applied. From the computational point of view, the primary phases of the KDD-process are the preparation, the exploration, the modelling and discovery, and the interpretation phases. They consist of repetitive modelling of single data derivation processes (i.e., streams, see below), execution of the processes and subsequent interpretation of the results and comparing them to former results and processes.

The iterative modelling and execution of streams leads to a huge number of data sets and results which might be worth to be stored for later re-use for documentation or efficiency purposes, e.g., to avoid expensive re-derivation of data sets.

In the following sections, we outline the main components of CITRUS. The information manager will provide comfortable means for accessing data and results. The execution server optimises the execution of the streams. The process support interface and user guidance will support the user in planning, executing, and documenting the process.

4 CITRUS

4.1 Overview of CITRUS

In CITRUS, we have chosen to build on an existing commercial knowledge discovery tool, CLEMENTINE. Developed by Integral Solutions Ltd. (ISL), it is an environment which integrates multiple modelling and discovery algorithms with tools for data access, data manipulation and pre-processing, visualisation and reporting. All the facilities are accessed through a novel user interface based on visual programming.

The user selects, from palettes, icons representing techniques for solving tasks of the KDD process. The icons - also referred to as *nodes* - are connected to define data flows (*streams* in CLEMENTINE terminology); their attributes are edited through pop-up dialogues to define the details of the processing steps. For a more comprehensive description of CLEMENTINE see Khabaza & Shearer (1995) and Shearer (1996).

Figure 1 shows the architecture of CITRUS. It extends CLEMENTINE by integrating a broader range of algorithms and tools, by adding persistent management of information relevant to the KDD process (metadata, approaches taken, intermediate results obtained, etc.), by providing a high-performance execution server to improve scalability to extremely large data sets, by extending the user interface to offer process-oriented support and user guidance, and by providing interfaces to other specialised software packages, e.g., statistical packages.

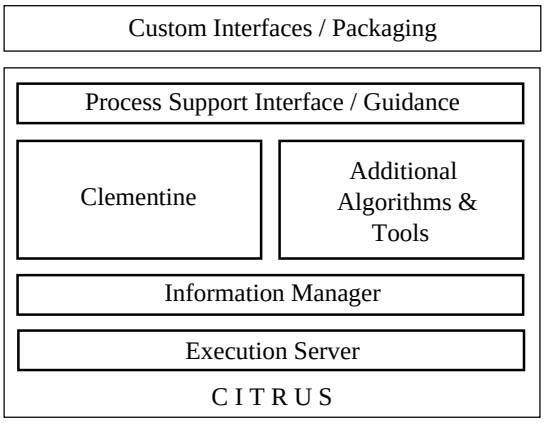

Fig. 1. The architecture of CITRUS.

4.2 Information Manager

The information manager has two principal goals: to support the modelling of the stream (especially the data selection and preparation tasks), and to offer means to retrieve and interpret data and results (cf. Breitner et al., 1997).

Support Modelling Today, the prevalent type of databases are relational database management systems (RDBMS). While the relational model undoubtedly has advantages in answering the needs of short online-transactional processing, handling of relational databases is painful when the relational schema grows. As major parts of streams deal with data selection and preparation tasks, the streams get very complex, causing the user to spend a long time sticking together nodes in an appropriate way. This will become even more difficult, if we assume the user is a domain expert rather than a database programmer.

The main reason is that the (relational) data model offers only little means for structuring the database. In order to ease the data handling and to cut down the time for modelling the streams, we propose to use an object-oriented data model as the representation of data. According to Brachman & Anand (1993) and Shoshani & Wong (1985) this is far more comprehensive and intuitive and leads to significantly simpler processes (Breitner et al., 1995). In contrast to the IMACS system (Brachman & Anand, 1993), we do not store the data physically in an object-oriented database system, since this causes considerable overhead for migrating the mostly relational data to the object-oriented representation. Instead, we only build an object-oriented schema.

The object-oriented schema is complemented by special operators, e.g., for aggregation and for feature construction. At execution time, the streams are mapped to their relational counterparts. This mapping from the object-oriented to the relational model is completely automatic without user interaction. The

object-oriented schema itself is (mostly automatically) constructed based on the relational schema during the domain analysis phase.

Retrieval of Results During a KDD process many streams are executed leading to a multitude of results. The retrieval of the data and results aims at supporting the user to answer following questions: What is the meaning of a data set? How has it been derived? How does it compare to other data sets? What is its relationship to other data sets? How have certain knowledge pieces been generated? What knowledge is available with respect to a certain data set?

In our approach, this is where the object-oriented schema comes in again. We use the object- oriented schema as the central information directory, which serves as an anchor for data, streams and results. The execution of each stream causes a trace in the schema, i.e., leads to a well-specified change. For instance, the derivation of a new attribute or the selection of a specific data set is reflected in the schema as a new attribute and a new data subset at a well defined place. Along with the new schema elements (here, attribute and subset) the derivation history is recorded in order to be able to reconstruct the stream at a later time. By using subsumption mechanisms repeatedly derived attributes and subsets are detected and, thus, only stored once; knowledge discovered for the same data set is, thus, connected to the same place (the data set) within the schema. Similarly, the information whether certain intermediate results have been materialised can be found at this well-defined location.

The object-oriented schema makes retrieval easy and comfortable and enables the automatic exploitation of intermediate results by the execution server and the process support interface. Together with the project plan generated by the process support interface, it provides a powerful mechanism to document the process and its results.

4.3 Execution Server

From a database point of view, a stream can be interpreted as a query against the database. With respect to the execution of such queries, most existing KDD-systems implement a strategy where at the beginning of the execution the data is entirely loaded into the system. This strategy is called *data-shipping* and is adequate for small data sets. However, in real- world KDD-applications the system is faced with huge data volumes and the streams consist of many data preparation and exploration operations. Since these operations are very data (memory) extensive, the data-shipping strategy quickly reaches its limits.

Therefore, we implemented an execution strategy which is called *query-shipping*. We assume the data is physically stored in a commercial RDBMS. These systems became widespread and are optimised for the management of giga and tera bytes of data. They also efficiently execute SQL queries, e.g., by using parallelism, sophisticated data organisation, and query optimisation. Since the data which is analysed is usually stored in a RDBMS it is reasonable to use a RDBMS for the management of data and efficient stream execution as well.

After mapping the streams created according to the object-oriented schema to their relational counterparts, this relational stream is partly exported to the RDBMS server in form of SQL queries. These queries are then processed efficiently and only a relatively small amount of data has to be sent back to the KDD-system, where the rest of the stream is executed. A similar query-shipping strategy is used by IDEA and INLEN (Kerschberg et al., 1992; Selfridge et al., 1996) as a basis to achieve scalability.

Additionally, we use two further optimisation approaches in CITRUS. First, the system support intelligent materialisation of derived data. Due to the iterative nature of the KDD process the same data sets are often produced several times. Instead of re-deriving these sets each time, substantial effort is saved if they are stored at the first time and reused afterwards. For this purpose, we use the object-oriented schema in two ways. First, the schema provides a semantic description of the data which can be used to decide if a materialisation can be used later. Second, we estimate the relevance of a materialisation by analysing the process history condensed in the schema. Further useful information about the most likely next steps of the process can be gained from a collaboration with the user guidance.

Second, we optimise streams. A potential drawback of the query-shipping strategy is that the relational database model offers only a limited functionality and streams rarely can be mapped to SQL statements completely. In such cases part of the stream functionality must be fulfilled by the KDD system. However, usually there exist multiple equivalent streams which can be mapped to SQL in different ways. In order to find the most efficient mapping we re-order and/or replace operations by different operations, which is quite similar to the query optimisation in database systems (e.g. Herzog & Schlösser, 1995). Particularly, when materialised intermediate results are taken into account, substantial performance gains are achieved.

Ideally, efficiency is a feature which should be provided automatically by the resulting system. The user should not need to be aware of any efficiency aspects during the KDD process. Therefore, we make an effort in CITRUS to let the system deide on the most efficient way to process required actions.

4.4 Process Support Interface and User Guidance

Since the KDD process contains many different phases and hence a lot of interacting tasks, the user of a process-oriented KDD tool needs guidance through the various stages of a KDD project (Engels, 1996). In CITRUS, the main purposes of user guidance are to set up and refine the project plan and to support the user in executing the plan. Since many techniques are available for the various phases and tasks of the KDD process, a user requires assistance in the selection and application of available techniques, in the interpretation and evaluation of results, and in the collection and documentation of final results.

The focus of user guidance is on support not on automation. We do not expect that executable streams are generated automatically from a problem description, or that techniques are automatically chosen and their parameters are

automatically set. The user is always in control of the process and user guidance is essentially a powerful help mechanism. However, the guidance module offers default settings and rules of thumb as often as possible.

From a user's point of view, a KDD project can be set up and executed in three ways. First, the user can link the available techniques in CITRUS and set up a stream from scratch. He selects CITRUS nodes from the palette of available techniques for each KDD task. Each node is accompanied by help descriptions about the functionality of the node and the tasks it is useful for. Thereby, obvious misapplications of single nodes can be prevented.

The input and output requirements of each node are automatically tested. If input is missing, the user is warned to create this input before applying the node and executing the entire stream. In addition, the tool also indicates when output is created but never used. If the user forgets an important task entirely, the user guidance module recommends to specify techniques for these tasks as well. For each technique, CITRUS suggests default settings depending on the inputs.

After an executable stream has been built and applied, the user guidance supports retain facilities. The user specifies descriptions of the problem solved for his stream or parts of his stream and for single nodes. Templates for such descriptions are automatically generated.

Second, the user can build on existing streams and adapt them to the current problem. The user can type in an informal problem description, and the user guidance module interactively retrieves the most similar projects and their streams. These streams are either actual streams that have been successfully applied in the past or pre-defined templates of generic streams for typical sequences of tasks. The guidance module then suggests the adaptation of a specific stream or the refinement of an abstract stream for the new project.

The third alternative is to iteratively decompose and refine a project plan until executable techniques become available to solve each resulting low-level task. This alternative is typically a combination of both building streams from scratch and re-using existing streams.

In the requirements analysis phase, a skeleton plan is defined which will be refined as more knowledge about the problem, the data, and the data mining goals becomes available. At the beginning of a project, the plan is basically a hierarchical decomposition of tasks. Plan refinement and task decomposition at lower levels are iterated until each task can be mapped to executable techniques.

For project plan decomposition and refinement, we use a partial planner that allows hierarchical problem decomposition and refinement. This enables us to represent abstract plans to a user, and guide him in specifying the remaining details. Intermediate abstract and partial plans represent the state of the KDD process and mirror the user's actions already performed and tasks to do in the rest of the project.

Project plans also serve another purpose which is extremely important but often neglected in KDD tools. The plans monitor what the user did for reports, history, and protocols. In combination with the information manager, the plan provides a context for the interpretation of results, helps to avoid repeated com-

putation of the same result, and helps to collect the interesting results at the end. This facility is very important for the EIC application where many experiments with different data sets and techniques have to be performed.

Finally, a project plan is an important input for the execution server. If it is known what the user is going to do next, the strategies for intelligent materialisation of intermediate results can be adjusted accordingly.

5 Conclusion

Most KDD research and development focuses on algorithm improvement. We believe this emphasis is wrong; to enable large-scale industrial KDD projects, it is more important to:

- integrate a comprehensive range of algorithms
- make the application of KDD scaleable to very large databases
- provide a process-based approach to KDD.

CITRUS is a process-based comprehensive KDD environment designed to meet these requirements. The development is well under way. The major components have been prototyped and their usefulness has been demonstrated. Of course, there remains a lot to be done. In particular, we need a still better understanding of the KDD process, how the various phases, tasks and techniques interact, what support a user needs, and how the execution server and the process support interface can interact most beneficially. The CITRUS prototype provides an excellent environment to study these questions; its use on a variety of ongoing projects within Daimler-Benz is producing valuable feedback on all these issues.

To date, most KDD developments have been driven by *technology push* rather than by *user pull*. Our work is motivated by the demands of current and future KDD applications in one of Europe's largest industrial organisations, and a perception of the issues faced by others tackling equivalent problems. As a result, unlike many technology-based KDD tools CITRUS is driven by application needs and places the user and his application problem in the centre of the KDD process.

Acknowledgements

We thank Julian Clinton for his various contributions to the work described in this paper.

References

1. Adriaans, P., Zantinge, D. (1996). *Data Mining*, Addison-Wesley.
2. Brachman, R.J., Anand, T. (1993). Integrated Support for Data Archaeology. *Proceedings 1993 AAAI Workshop on Knowledge Discovery in Databases*, pp. 197-212.
3. Brachman, R.J., Anand, T. (1994). The process of knowledge discovery in databases: A first sketch. *Proceedings 1994 AAAI Workshop on Knowledge Discovery in Databases*, pp. 1-12.
4. Breitner, C., Freyberg, A., A. Schmidt (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.
5. Breitner, C., Schlösser, J., Wirth, R. (1997). Process-Based Database Support for the Early Indicator Method, submitted.
6. Engels, R. (1996). Planning tasks for knowledge discovery in databases; performing task- oriented user-guidance, In *Proc. of 2nd International Conference on Knowledge Discovery and Data Mining*, Portland.
7. Engels, R., Lindner, G., Studer, R. (1997). A Guided Tour through the Data Mining Jungle, submitted.
8. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurasamy, R. (1996). *Advances in Knowledge Discovery and Data Mining*. Cambridge, MA: MIT Press.
9. Herzog, U., Schlösser, J. (1995). Global Optimization and Parallelization of Integrity Constraint Checks. *Proc. of the 7th International Conference on Management of Data (COMAD 95)*, pp.186-205.
10. 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.
11. Khabaza, T., Shearer, C. (1995). Data Mining with Clementine. IEE Colloquium on Knowledge Discovery in Databases, *IEE Digest* No. 1995/021(B), London.
12. Selfridge, P.G., Srivastava, D., Wilson, L.O. (1996). IDEA: Interactive Data Exploration and Analysis, *Intl. Conf. Management of Data (SIGMOD)*, pp. 24-34.
13. Shearer, C. (1996). User Driven Data Mining. *Unicom Data Mining Conference*, London.
14. Shoshani, A., Wong, H.K.T. (1985). Statistical & Scientific Database Issues, *Transaction of Software Engineering* (10), pp. 1040-1047.
15. Reinartz, T.P., Wirth, R. (1995). The need for a task model for knowledge discovery in databases. in: *Workshop notes Statistics, Machine Learning, and Knowledge Discovery in Databases*. MLNet Familiarisation Workshop, Heraklion, Crete, pp. 19-24.
16. Wirth, R., Reinartz, T.P. (1996). Detecting Early Indicator Cars in an Automotive Database: A Multi- Strategy Approach. *Proc. of the 2nd Int. Conference on Knowlege Discovery and Data Mining*, pp 76-81