An Intelligent Data Retrieval Assistant

Jussi Stader and Robert Inder [1]

AIAI-TR-129

July 1993

To be Presented at Expert Systems 93, Cambridge, UK, December 1993

Artificial Intelligence Applications Institute (AIAI)
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

[1] previously at AIAI, now at Human Communication Research Centre (HCRC)
The University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW

Abstract

Recognising and gathering relevant data is often a very time-consuming part of an expert's work. Databases can make the necessary information available, but current access methods require database skills which few experts have. Bridging this gap between experts and the information they require can greatly improve the experts' effectiveness.

We describe a knowledge-based approach which allows users to browse databases without requiring database skills. A model of the domain is used to determine the interface to the system. It transforms the database structure and its contents into information the user can readily appreciate. Familiar terms and concepts can be used to formulate queries, to control display of information, and to navigate through the database. Knowledge about the domain is also used for adapting the interface to the nature of the information thus making interaction with the system more effective. Fuzzy matching techniques further support the user in specifying what information is required and they provide a more intuitive access to the data.

A system embodying these ideas is being developed, and has already been applied to a number of different databases in different domains.

Contents

1 Introduction

Expert Systems technology is often perceived as a means for reproducing the decision-making behaviour of an expert, either for giving advice to less experienced staff, or for automating a decision entirely. However, there is a substantial number of experts in many disciplines who carry out highly skilled or creative tasks using large amounts of data. A large part of their work is not concerned with actually making or evaluating decisions, but with obtaining the information needed to do so. Their knowledge of this data—their ability to say what kinds of data may be relevant to a problem and where it might be found—is in itself a substantial body of expertise, and acquiring and using that expertise can consume much of the experts' time.

Increasingly, the information these experts need is being held on computers. While this is potentially a great step forward, a significant problem remains (see Figure 1).

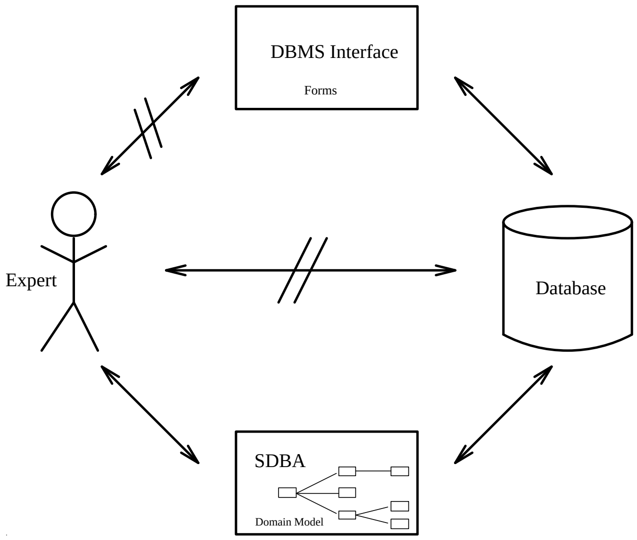

Figure 1: Communication Problems in Information Gathering

Interacting directly with a database requires database-specific knowledge and skill which many domain experts do not have, and either will not or cannot spend time acquiring. Equally, while system developers can provide user-friendly front-ends—e.g. a "Forms" interface—these only handle cases where the data structures are comparatively simple and likely queries and paths through it can be predicted. Unfortunately, creative thought processes tend to be unpredictable, and the result of one query will often determine the nature of the next.

This paper describes work on applying knowledge based systems not to the task of making decisions, but to the subordinate task of finding and presenting the relevant data.

The next section describes in more detail the problem of bringing together experts and the data they require. Section 3 describes an approach for providing a tool which bridge the gap between the two. In section 4 we give a brief description of the Smart Database Access tool (SDBA) which has been implemented at AIAI to demonstrate the approach. Finally, section 5 gives a summary and an outline of the potential for further extensions of the approach.

2 The Problem

The value of data, and the importance of ensuring that it is conveniently available, motivates much of the use of computers. Storing and retrieving data is an archetypal application, and falling storage costs and rising display capabilities are together allowing an increasingly large range of data to be held on-line.

The majority of those who use data do so for routine purposes: warehousemen load lorries, students attend exams, customers check and pay their bills. These people have fixed data requirements, and thus their needs are straightforward. However, many people carry out complex or creative tasks on the basis of substantial amounts of data. Initially, the work reported here focussed on geologists concerned with using exploration and production data to assess the likely value of looking for oil in one area or another, but there are many others who have generally similar data access requirements: e.g. senior managers, system designers, researchers of almost any kind.

Much of the expertise of these people lies in their knowledge of the data they are using—its location, the way to analyse it, the limitations on its accuracy or applicability—coupled with their ability to interpret it, and to see its significance to the activities and objectives of the organisation. As they do so, they often want to see additional data to continue or support the train of thought triggered by the data they have already retrieved. This makes it impossible to anticipate their data requirements, and thus to prepare interfaces to present the data they will need: the experts need to have the ability to access the information they require directly and control its display themselves.

Unfortunately, expertise in data doesn't make a database expert. Knowledge of what data is held is not enough to understand how it is held. Many factors interact to determine the structure of a database, including storage space, access time, compatibility with other systems, data integrity, control over access and the need to support many users simultaneously and reliably. As a result, database structures seldom reflect the structures of the real world, or the way experts think about the information the databases contain and it is often difficult to appreciate the organisation of information in databases. This matters, because the most flexible data access systems that are currently widely available – based on the SQL family – require knowledge of the structure of the database. Moreover, SQL is a

```
SELECT well_base.name, well_base.depth, well_location.country
FROM   well_base, well_location
WHERE  well_location.region = 'NORTH SEA CENTRAL GRABEN'
  AND  well_base.status = 'COMPLETED'
  AND  (well_base.result = 'O' OR well_base.result = 'OG')
  AND  well_base.key = well_location.key
```

Figure 2: An example SQL query

formal (programming) language: an example query is shown in Figure 2. For many who are uncomfortable with things mathematical, this is enough to render these tools beyond their reach. Those who are undeterred will quickly find that it is usually possible to read a query and guess what it is supposed to do, but that formulating queries themselves requires detailed knowledge of the exact form of the query language and the structure and contents of the database. Moreover, possibly unintentional variations in queries result in very different answers, particularly where missing data is involved. Together these factors mean that using a language like SQL to formulate a query which will produce the right results requires more knowledge and skill than many experts will be willing or able to acquire.

There are two "traditional" solutions to this problem: producing huge printouts, and employing a database specialist to retrieve data as and when required. Both have obvious drawbacks, and the ideal solution is to improve database access mechanisms so that no specialist skill is needed. The gap between experts and data must be bridged by tools that let the experts query the data in their own terms: user oriented access mechanisms must replace the current database oriented access mechanisms.

There are a number of developments in database technology that address some of these issues—for instance object-oriented database systems, which support data structures that reflect the world more closely than the tables and columns of relational database systems, and deductive databases. However, the gap between database issues and user needs remains, so that while these systems may be *better* than current ones, their user friendliness still leaves a lot to be desired. Moreover, these approaches are still new, and the effort of transferring existing data, let alone software or skills, is a substantial barrier to their adoption. These problems are particularly acute where data or software is being maintained elsewhere, because updates would continue to be issued for the old databases.

A better approach would be to improve the usability of database systems by providing tools that shield the user from their structure and technology. Current tools of this sort are mostly based on forms which are predefined by database specialists and can be used to query the database and display results. They provide good support for areas where there are standard, routine ways of accessing a database, where queries can be predicted and paths through the data are simple. However, these tools and application-oriented access facilities are usually not flexible. The user's requests must be anticipated by the system developer, which restricts the user to common paths of inquiries. They are not flexible enough to allow for creative data browsing where the result of one query determines the next query.

By analysing the needs of database users, one can identify a number of desirable features for a tool to support them. It should:

- embody the experts' view of the world and allow them to query databases and display data in their own terms, without reference to the internal operation of the database (or even databases) involved,
- support formulating queries and viewing results, actively assisting the user in obtaining and displaying the desired information,
- facilitate incremental query refinement, making it easy to build on previous queries and to access information related to current results – it should make it easy to navigate through the database and not restrict users' ability to pursue a line of thought,
- handle imprecise queries to allow users to formulate queries on the basis of an incomplete or uncertain idea of what is required,
- minimise the effect on the user's query of the incompleteness or inaccuracy that will inevitably be present in the data, and
- keep track of the history of the user's interaction with the system, helping them to re-trace their reasoning or explore in a different direction,

The next section describes, from a user's point of view, a tool which is intended to provide all these features. For a more technical description of the tool see [Stader, Inder and Chung].

3 The SDBA Approach

Our approach to database browsing is similar to the interaction of an expert with a good assistant: the expert specifies what is needed in the form of a high level query and the assistant collects all relevant items performing the details of information access, puts them into a "bag", and hands the bag to the expert. The expert can then pick items from the bag and inspect them in more detail and from different angles.

This approach frees the user from the details of how to access information in databases – the assistant/system does this for the user. It allows the user to interact with the system at a suitable level of abstraction and in familiar terms. It also splits the traditional query specification into two stages: first users specifies what is of interest and then they decide how to view relevant information and what aspects of information to examine.

3.1 Views of the World

Throughout all the user's interactions with the system—formulating queries, inspecting information or navigating through the data available—information is referred to in terms of a domain model which is built using knowledge acquired from domain experts. The structure and operation of the database(s) are completely hidden from the user.

The domain model is object-oriented consisting of objects and their attributes (parameters), and the possible relationships between objects. Attributes are organised into a type hierarchy and information about attribute values can be represented, including normal values, possible values, and possible units. For example, modeling petroleum geology there are objects like wells, areas, and sedimentary horizons; attributes of a well include its total depth, its location and its operator; there is a relationship between wells and areas: a well "is in" an area and an area "contains" wells.

While experts cannot be expected to be database specialists, they *can* be expected to know their area of expertise, and to relate to abstract models of that area. Thus the domain model is a suitable vehicle for interaction. Objects, attributes and relationships replace database concepts like tables, columns and joins between tables. The domain model is used to structure information and to determine the interface to the system in order to hide structural details of the database, but it also provides the knowledge needed to support the user during interaction.

3.2 Query Specification

Viewed in object-oriented terms, a database query has two parts: a specification of the objects of interest, and an indication of which attributes of those objects are relevant. In the SQL example in Figure 2 the objects of interest are completed wells in the Central Graben of the North Sea which have produced oil or oil and gas—the WHERE part of the query. The relevant attributes of each of these wells is its name, depth, and location—the SELECT part of the query.

These two parts of the query can be handled separately. Initially, the user specifies which objects are of interest by providing constraints (the WHERE part). This we call query specification. During this stage users do not need to think about which aspects of the objects they want to view. The decision of which aspects of the relevant objects to view is postponed until the objects have been collected (see 3.3).

Users formulate queries by specifying constraints on attributes of objects. Different constraint editors are available for different attribute types so that each constraint can be specified in the most appropriate way. For example, in order to constrain the depth of a well the editor best suited is a number editor which supports the specification of numeric values or ranges, while the well result is best specified using a menu editor which allows the user to select one of the possible results. Attributes whose values can be organised into a classification scheme or taxonomy are best edited using tree editors that reflect the hierarchical organisation of values. Examples of more specialised constraint editors are a map on which the user can select an area as a constraint for locations, and diagrams with "hot spots" for identification can be used to specify features. The most general constraint editor is the string editor for free text entry.

In addition to selecting the most suitable editor for the constraint specification the system supports the user with the help of knowledge about possible and normal values of attributes. This information is used to help avoid constraints that cannot have, or are unlikely to have, matching values in the database.

Finally, initial settings for constraint editors are provided to make the constraint specification process more efficient. Previous constraints or normal values are used for initialisations where appropriate.

3.3 Display of Information

Once the objects that match the specified query have been identified and collected into a bag they are ready for inspection. The approach to data display is similar to query specification. For editing constraints there are many editors and the most appropriate one is chosen according to the type of the attribute to be edited. For displaying data there are different display types, called "viewers". Again, there are general purpose viewers like tables, and more specialised viewers like maps, crossplots or diagrams. Each viewer shows different aspects of the (collection of) objects. Choosing a viewer determines which data to display which automatically specifies the SELECT part of conventional database querying.

Looking at the results of queries in an object-oriented way there are two types of results: sets of objects and individual objects. Along this line viewers can be divided into two groups: viewers for an individual object and bag viewers. Viewers for an individual object show details of the object as a form, map, diagram etc. Bag viewers show information on sets of objects typically in the form of summaries or distributions, e.g. in the form of tables, charts, graphs, histograms, crossplots, or maps. For all viewers the data values that are being displayed are formatted according to their type and according to their function and position in the viewer. Units of data values are also taken into account and displayed if appropriate.

Which viewer is most suitable for displaying information depends on the user's interest, but also on the type and amount of information to be displayed. Tables are generally useful for displaying summaries of objects, whereas maps are only suitable if locations are of interest. Distribution viewers like graphs and histograms are useful to show aspects of large collections of objects but given only 3 value points a graph is unlikely to provide any valuable insights. Some distribution viewers are only suitable if numerical information is highly relevant.

For example, if the system has collected all completed wells that are between 3000 and 4000 feet deep, this collection can be viewed as a map showing their locations. Or the set could be summarised in a table showing such details as the depth or result of the well (although, ideally, not their status, since they are all, by definition, `COMPLETED`. The depth of the wells can be plotted in a graph or a histogram. Displaying individual wells different aspects may be of interest like technical details about depth, location and result or administration details like the wells' operators and license numbers. Graphical details like the path of a well could be viewed as a diagram while location details could again use a map display.

The system can actively support the user in choosing an appropriate viewer. The type and the number of objects in a bag and the user's interest are the main criteria for deciding how suitable different possible viewers are for display, but the context of the query specification and the constraint editors that were used may also give an indication. The system can choose the most appropriate viewer as an initial display. If the user needs different or more information, additional viewers can be opened selecting from a choice of possible viewers ranked by how suitable they are. This makes choosing a viewer simple and flexible and provides a very efficient way of displaying information.

3.4 Fuzzy Matching

Query specification is further supported by fuzzy matching facilities. Matching in databases is performed by looking for values that match the specification exactly. Fuzzy matching is used to soften this "all or nothing" approach to matching. It addresses issues of uncertainty, which can arise either because the user does not know how best to describe the required set of objects, or because the data itself is inaccurate. Vague or unreliable data in the database can make that data inaccessible and thus cause other information to be missed. Fuzzy matching techniques make it possible to deal with the such data and thus exploit all available information by returning not only items that match the specification exactly, but also near misses, i.e. items that are similar to the specification. This provides the means to relax constraints and thus gives users more flexibility in specifying queries and more control over the number of results a query produces.

For example, a well's result has the following possible values: "dry", "gas", "'gas shows", "oil", "oil shows", "oil and gas", and "oil with gas shows". In a query for oil wells result values of "oil" are exact matches, but "oil and gas" and "oil with gas shows" are nearly as good because these wells all produce oil. The result "gas" is fairly similar because it also indicates hydrocarbon production. "Oil shows" is not that bad either because it may indication oil in the vicinity of the well. "Gas shows" and "dry" are pretty bad matches. If a query for oil wells returns too few matches the user can relax the specification "result oil" to "result very like oil" which would translate to oil production. This can be further relaxed to "result like oil" for hydrocarbon production or even to "result quite like oil" to include "oil shows". Gradually relaxing a constraint this way makes the system cast the net wider and wider in the search for matches in the database.

Some query languages allow wildcard matching. For example, in SQL the constraint value "oil%" will match all values in the database beginning with "oil". This concept of similarity is based on similar texts only. It can cover similarities in names: "Brown" is similar to "Browne" and "Brownie". However, what if we decide that it is also similar to "Black"? Furthermore, wildcard matching in query languages does not determine *how* similar texts are: "Brownie" matches "Brown" just as well as "Brownington" or even "Brown is a colour".

For a detailed discussion of fuzzy matching in databases see [Chung and Inder]. For details about the underlying concepts of fuzzy reasoning and fuzzy set theory see [Schmucker, Gaines]

3.5 Navigation

A main action in database browsing is to move from a set of data to other data which is of interest in the current context. Many databases allow these connections to be made but they are difficult to use because they are provided in terms of artificial links which do not reflect their meaning to the user.

In our approach the simplest form of navigation is the process of zooming from a collection of objects into details of a single object. However, this form of navigation is restrictive, it can only be used to provide more details about a query's results. For more flexible navigation relationships between objects are used. Thus the domain model provides two types of meaningful connections between information:

- the concept of objects having attributes is used to connect all information directly relevant in the context of a single object. This way information is clustered around objects.

- Relationships between objects are used to provide paths between different clusters of information. The user can use these paths to navigate the database. For example, a well is in an area (wells and areas are related via "is in"), and thus it is easy to move from information about a specific well to information about the area that the well is in. From there moving back to *all* wells the area contains or moving on to a neighbouring area is equally easy.

3.6 Other Features

SDBA also provides a number of other services which help smooth the expert's access to the data.

Quite often, there is more than one way of finding or deriving a certain piece of information, either within a single database or across different databases. Having a database independent domain model for user interaction decouples the information from its location and alternative sources can be taken into account resolving conflicting data and integrating complementary data.

Following lines of thought to navigate through the information is supported by different ways of incremental query specification. The user can go back to queries and modify them to obtain exactly the right set of results. Additionally, most viewers can be used as query environments which facilitates building further queries on currently displayed information (see also [Inder and Stader]).

The danger of losing a line of thought by being side-tracked is alleviated by keeping track of the user's interaction with the system throughout a session. A graphical summary of the logical steps the user has taken is provided and can be used to go back to previous queries or results or to cut off lines of investigation that are no longer needed. This enables users to return to the main investigation from a side track or to explore different options. It also reminds users of the intentions of queries and the meaning of results.

Some aspects of the system can be customised or extended by the user to adjust the system to the user's needs and to give more flexibility. A set of user preferences is taken into account which parameterises things like in which units values are displayed, how to treat and display missing values in the database, and preferences for viewers. There are editors for specifying and modifying the contents of viewers which gives users immediate and direct control over what information is displayed and how it is presented. Users can also define their own objects and concepts either in terms of other objects, attributes, and relationships or as new concepts. These new concepts can be used to extend the domain model or to adapt it to the user's view of the world.

4 The SDBA System

In this section there is a brief description of the SDBA demonstrator which was implemented at AIAI. For a detailed description see [Stader, Inder and Chung].

The overall architecture of SDBA is summarised in Figure 3 The user interacts with the system in a point-and-click style through a graphical user interface. All interaction is driven by the user.

The system kernel is responsible for transforming user interaction into queries, translating between different internal representations of queries, translating be-

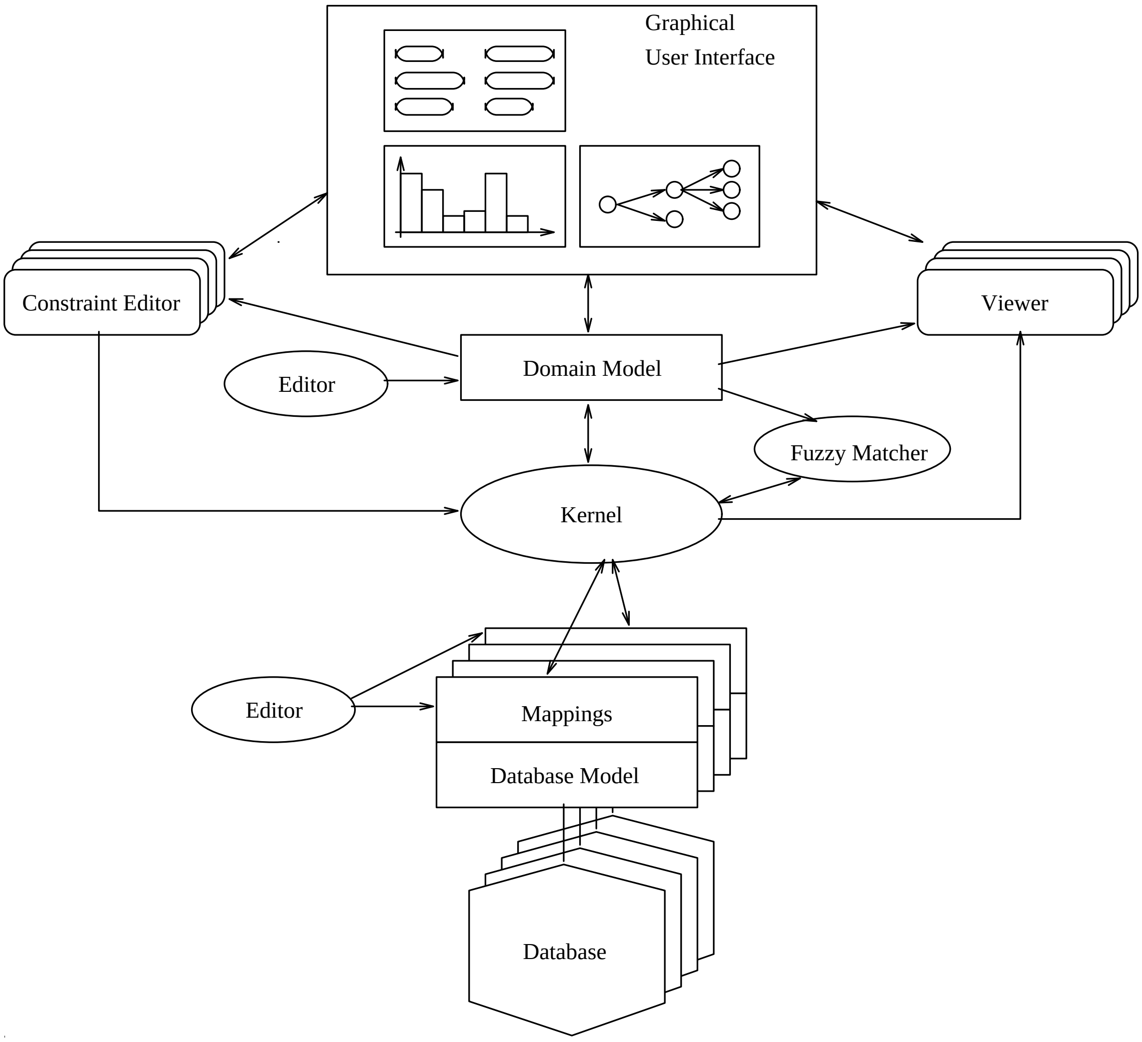

Figure 3: The SDBA Architecture

tween domain terms and database terms, performing unit conversions, producing queries suitable for execution by a database management system, preparing information for display, etc. The fuzzy matching module assists in query generation and in the assessment of results.

SDBA uses an object system for its knowledge representation. This contains

- the domain model (objects, attributes, relationships),
- the database model (tables, columns, value types),
- a collection of constraint editors,
- a collection of viewers,
- mappings between domain structures and database structures, and
- translations between database values and domain terms.

The last two contain information about how to bridge the gap between the domain model and the database model.

Specifying domain and database dependent information is supported. The domain model is put together and modified with the help of HARDY, a graphical hypertext editor developed at AIAI (see [Smart]). The models are specified by drawing diagrams from which specifications for the object system are generated automatically. The database model can be extracted from the database automatically if the database management system provides data dictionary information. The mappings between the domain model and the database model are specified using a set of point and click style editors.

The ideas for the SDBA approach originate in the PEXES project which was carried out jointly by Simon Petroleum Technology (SPT) and AIAI [Inder and Wells]. It was partly funded by Petroleum Science and Technology Institute (PSTI). The objective was to produce a system that could find analogues of (parts of) geological basins for petroleum exploration. A database browser was implemented as a first step to support the petroleum geologist in this task. During the PEXES project, the browser was connected to three different geological databases (two containing information about geological basins and one about wells). SDBA was developed by AIAI as a generalisation and an enhancement of the PEXES browser. The ideas of that project were refined and abstracted to produce a domain-independent database access tool. SDBA has since been used as a demonstrator for accessing databases in the fields of chemistry (test data) and ornithology (bird observation data provided by Lulu D. Stader, University of Stirling).

5 Conclusion and Future Work

The approach described in this paper and the SDBA demonstrator show that the gap between experts and their data can be bridged by a tool that provides intelligent support for database browsing. The tool assists users by :-

- performing detailed data access from high level query specifications,
- helping users to formulate high level queries through specialised constraint editors,
- providing different views on data which suit the user's need in the current context,
- handling uncertainty in query specifications,
- ensuring that queries adequately handle inaccurate and missing data,
- facilitating incremental query specification in order to get the right results,
- providing structure for information and paths for navigation,
- supporting the use of viewers as query environments to follow lines of thought,
- keeping track of the interactions during a session to remind users of their intentions, and
- unifying multiple sources of information.

The underlying concept for interactions between users and SDBA is the "bags and viewers" metaphor which is easy to understand and to use.

The main features of the SDBA approach are that it

- reflects the user's view of the world,
- is flexible to use,
- does not require any knowledge of database technology, and
- deals with incomplete and inaccurate information.

SDBA has potential for extension to provide further assistance, for example:

True Query by Example is a type of query where a description of an object is used to find other objects like it. This is a useful mechanism in areas where analogues are of interest. For example, in geology describing what is known about a relatively unexplored basin the geologist can find similar basins about which more is known. Information about these better explored basins can then be used to estimate information about the new basin which can help to determine whether or not it is likely to contain hydrocarbons.

Analogue finding is based on the fuzzy matching described in section 3.4 but whole objects are compared instead of attribute values. Comparing objects is done by comparing values but there are additional issues like which criteria to use, i.e. which attribute values to compare, and how to combine them.

Analysis of query results can be performed by the system based on its domain knowledge. The system can determine common features of objects, or it can point the user to unusual objects thus assisting the user in making sense of information.

Decision Support can be given if some of the user's task is known to the system. This support can range from suggesting what information to look at to formulating queries and displaying information on the user's behalf. How much support can be given depends on how much of the user's task is known and how predictable the strategies are.

References

[Chung and Inder] Chung, P.W.H. and Inder, R.
Handling Uncertainty in Accessing Petroleum Exploration Data In *Proceedings of EuroCAIPEP 1991*, Paris, France. October 1991. Also appears, in extended form, in of the *Revue de L'Institut Francais de Petrole*, Vol 47, No. 3, May/June 1992. Also available as AIAI-TR-104.

[Gaines] Gaines, B.R.
Foundations of Fuzzy Reasoning Int. J. Man-Machines Studies, vol 8, pp 623-668, 1976.

[Inder and Stader] Inder, R. and Stader, J.
Bags and Viewers: A metaphor for Intelligent Database Access Technical Report, AIAI-TR-127, 1993.

[Inder and Wells] Inder, R. and Wells, B.
PEXES*: Combining Knowledge and Data in a Tool for the Explorationist* Proceedings of *1991 Conference on Artificial Intelligence in Petroleum Exploration and Production (CAIPEP '91)*, College Station, Texas, 1991, pp 99-106. Also available as AIAI-TR-93.

[Schmucker] Schmucker, K.J.
Fuzzy Sets, Natural Language Computations, and Risk Analysis Computer Science Press, 1984.

[Smart] Smart, J.A.C.
User Manual for HARDY AIAI, Edinburgh 1993.

[Stader, Inder and Chung] Stader, J. and Inder, R. and Chung, P.W.H.
Transforming Databases for Experts Technical Report, AIAI-TR-128, 1993.