# Interface Support for Data Archaeology

Loren G. Terveen
AT&T Bell Laboratories
600 Mountain Avenue, 2C-401
Murray Hill, NJ 07974-0636 USA
terveen@research.att.com

**Abstract**

*I describe the IMACS interface, which supports a type of interactive data exploration task called data archaeology. The interface facilitates users in performing this task using three key design principles: (1) combine power and ease of use, (2) provide direct support for integrated, iterative data exploration, and (3) assist users in managing their work over time. I show how these principles are relevant in the data archaeology task, describe how knowledge representation technology provides a foundation for an adequate support system, and illustrate in detail how the interface offers powerful support for data archaeology.*

**Keywords**: knowledge discovery, interactive data exploration, marketing, knowledge representation, reuse

## 1 Introduction

Databases have become ubiquitous throughout business, government, science, and other types of enterprise, leading organizations to view databases as sources of new and useful implicit knowledge. For example, a database of customers and purchases, originally used for billing and inventory management, may now be seen as a resource for predicting customer behavior and defining customer sub-groups, leading to more targeted, effective, and easily evaluated marketing campaigns. In a previous paper [7], we identified an approach to knowledge discovery in databases that we call *data archaeology*. The term emphasizes that this is a skilled process in which answers do not emerge in one pass, as full-blown nuggets, but rather evolve in an iterative, dialectic process that requires constant human intervention. This process resembles the highly skilled work of an archaeologist. This paper describes a system called IMACS (Interactive Marketing Analysis and Classification System) [7] that addresses these problems, focusing on the user interface.

While IMACS is a multi-faceted system – it uses knowledge representation technology to provide an expressive representation of data and to integrate data from multiple databases, employs novel techniques for translating from databases to a knowledge base, and features a custom query language – these aspects of the system are discussed only as necessary to support the explanation of the interface. The IMACS interface supports the data archaeology task using three key design principles:

- *Combine power and ease of use* – general purpose systems such as the UNIX operating system or the Common Lisp language provide their users a great deal of power; however, the price of this power is that they are difficult to learn and to use optimally. Direct manipulation graphical interfaces are easy to use, but often limited in their functionality. IMACS offers the power of a general purpose knowledge representation system and query language and the ease of use of a graphical, form-based interface.
- *Directly support the iterative, integrated data exploration that characterizes data archaeology.* One hallmark of the task is that the usefulness of a category that is formed in the analysis process only can be determined through further analysis. Thus, such "working categories" are treated as "first class" objects to which all the analysis operations of the interface can be applied. Further, IMACS provides integrated support for hypothesis formation and testing. IMACS makes it very easy for analysts to view data to form tentative hypotheses about the data, segment the data into categories that embody the hypotheses, then view the categories to test the hypotheses.
- *Support users in managing their work over time* – the importance of reuse has been widely recognized [2, 3]; in IMACS, analysts create reusable resources as a natural part of doing their analyses, and these resources subsequently can be exploited both by their creators and other analysts.

The remainder of the paper is organized as follows. I first discuss the task of data archaeology and show how current technology does not support it adequately. I next describe the role of knowledge representation technology in IMACS. The core of the paper comes next: I describe the task structure of the interface, show how the design principles enable adequate support for data archaeology, and illustrate these points with analysis scenarios. I conclude by discussing the status of the system, including how it is being used within AT&T, and areas for future work.



CIKM '93 - 11/93/D.C., USA

© 1993 ACM 0-89791-626-3/93/0011 ....$1.50

## 2 The Task Of Data Archaeology

As a human task, data archaeology requires appropriate support tools. In order to understand the type of support required, we first consider the characteristics of the task in more detail. Let us use the department store example to illustrate these characteristics. Suppose a department store maintains the following data:

- customer information including names, addresses, and credit information like current balance,
- inventory information about store items such as their department, price, and current quantity in stock,
- purchase information such as customer, item purchased, date purchased, and method of payment, and
- sale information, including dates and details of past and planned sales.

In general, the department store would want to understand and predict the behavior of classes of customers, the effect of sales on purchases, and possible correlations between purchases of different items or classes of items.

Specific questions the store might want answered include:

- Who are the biggest customers, and when during the year are they likely to buy?
- Are customers who buy from one department (for example, Electronics) also likely to buy from another department (for example, Sports)?
- Is a certain class, for example, a class of "steady customers", growing, shrinking, or staying the same?
- Are certain classes of customers more likely to respond to sales?
- Do customers who make most of their purchases during sales spend more money than other customers? Are they more likely to fall behind in their credit card payments?

Consider the process an analyst might go through in answering the last two questions concerning the behavior of sale customers. The key characteristics of such an iterative, archaeology-like process might include:

- *Useful patterns may be hidden within a large pre-defined class*, like the class of all customers. For example, it may contain a set of "Sale Customers", who make a large portion of their purchases during sales. This set can be extracted only by *segmenting* the set of customers using the appropriate parameter (such as purchase date) and parameter values (such as "between December 26th and 31st"). In general, there is no way to do this a priori, and such analysis therefore is beyond the reach of automatic data mining.
- *The usefulness of a category like "Sale Customers" can be determined only through additional analytic work*, e.g., statistical analysis, comparison to other categories, and visualization. Having segmented customers into "Sale Customers" and "Non-Sale Customers", one will want to check whether "Sale Customers" spend more money than Non-Sale Customers or are more likely to fall behind in their credit card payments. Forming and testing such generalizations requires viewing and comparing aggregate properties of analysis categories and properties of individuals in the categories. Such analysis might result in re-definition of a category; for example, "Sale Customers" might originally have been defined as those who make more than half of their purchases during sales, but if analysis of this category reveals no useful generalizations, the analyst might re-define "Sale Customers" as those who make more than a third of their purchases during sales, which may in turn lead to useful generalizations.
- *Analyses often build on previous analyses.* Certain categories and the queries that generated them may be useful in subsequent (related) analyses. For example, an analyst may decide that she is likely to re-examine the class of "Sale Customers" in future analyses and does not want to reproduce the work required to define it. The analyst also may want to track changes in the size and makeup of this class over time: as customers change their buying behavior and become "Sale Customers", the store may want to target them with certain mailings. The analyst also might realize that the query used to retrieve "Sale Customers" could be widely applicable if generalized; for example, it could retrieve customers who make most of their purchases in a certain department or during a particular season.

To summarize, the data archaeology task cannot be captured in simple automatic methods. Useful knowledge is constructed through a flexible process of interacting with the data to form, investigate, and revise hypotheses.

While current mainstream data management technology (relational databases and SQL) has raised the possibility of being able to do such analysis, it does not provide adequate support for data archaeology. First, there are problems in the *representation* of data – relational databases often are incomplete or incorrect, and integrity constraints typically are not enforced. Second, tools like SQL have serious limitations for data analysis:

- difficulties in *accessing* data – while SQL provides a relatively expressive query language, large queries are difficult to formulate and comprehend. A user must be intimately familiar with the structure and organization of the database in order to write effective queries. It is difficult to gauge the accuracy of the query result, that is, both whether you asked for the right thing and whether you received the result you expected.
- little or no support for *iterative investigation and exploration of data* – it is hard to further analyze results of a query to determine how useful the results are. It is hard to vary a query slightly to compare its results to the results returned by the original query.
- little or no support for *managing work over time* – first, there is no way for analysts to recognize and reuse common patterns in their queries. Second, limitations of SQL often lead analysts to write a query simply to extract all the data that might be of interest into a very large flat file and then use a general purpose programming language like AWK or C or a statistical package like SAS or S to analyze the data. In this scheme, a category such as "Sale Customers" is represented only as an SQL query or the data in a file that resulted from running the query. It is up to the analyst to keep track of the files and queries, there is no way to track migration into or out of the category over time, and,

in the latter case, the analyst will have to reproduce the query if she wants to analyze "Sale Customers" (or a similar category) in a subsequent analysis.

From the characteristics of the data archaeology task and the shortcomings of current tools, we can extract some requirements for an adequate support system:

- without sacrificing the power of a general purpose query language, the system must make it easy for analysts to express queries and comprehend results;
- it must be easy to form tentative segmentations of data, to investigate and explore these segments, and to re-segment quickly and easily; there must be a powerful repertoire of viewing methods, and these methods must be applicable to segments;
- the system must allow analysts to recognize and abstract common analysis patterns; it must be easy to apply and modify these patterns; the domain representation should be extendable by the addition of new categories formed from queries; there must be some way to track migration into and out of these categories over time.

## 3 The Role Of Knowledge Representation

We use knowledge representation (KR) technology as the basis for an adequate data archaeology support system. It provides a declarative, natural representation of an application domain. In the current application, the data comes from several databases; thus, we also use KR for database integration. We use the frame-based KR system CLASSIC [4, 6]. There are three kinds of formal objects in CLASSIC:

- *concepts* – structured descriptions of sets of objects formed by composing a limited set of operators (e.g., EXPENSIVE-ITEM might represent an ITEM whose price role is restricted to be greater than 500); concepts correspond to one-place predicates;
- *roles* – two-place predicates that relate individuals (e.g., item-purchased would relate a PURCHASE individual to an ITEM individual);
- *individuals* – objects in the domain of interest; individuals are given properties by asserting that they satisfy concepts (e.g., Joe-Smith is a CUSTOMER) and that their roles are filled with other individuals (e.g., the department of Nike-Air is Shoes).

CLASSIC objects are written in typewriter font, with concepts (ITEM) in uppercase, individuals (Joe-Smith) capitalized, and roles (item-purchased) in lowercase.

Two features of CLASSIC are of interest for the present discussion. First, the "schema" (concept hierarchy) can be extended dynamically. This is necessary because analysts must be able to define new concepts from categories in their analysis. Second, CLASSIC is the basis for a query language that is crucial in enabling the interface to meet the requirements we have stated.

The query language (QL) is a set-oriented language designed specifically for CLASSIC, although it borrows quite a bit from systems like TDL [5]. It includes constructs for defining sets using logical operators, role following syntax and conventions, and arithmetic functions. The interface uses the QL for data retrieval and for forming sub-sets (segments) of data. Some example QL expressions are:

*access* –
Nike-Air.current-price
*the current price of a* Nike-Air *shoe*
COUNT (Joe-Smith.purchases)
*the number of purchases made by Joe-Smith*
Joe-Smith.purchases.item.department
*the set of all departments in which Joe-Smith has made a purchase*
*segmenting* –
x in CUSTOMER where
x.amount-spent > 1000
*the set of* CUSTOMERs *that spent more than $1000*
x in Joe-Smith.purchases.item where
x.department = Electronics
*the set of* ITEMs Joe-Smith *bought from the* Electronics *department*

Technically, a set formed by a query is called a *collection*. The QL can operate on collections, e.g., where a concept like CUSTOMER could appear, so could a collection of CUSTOMERs resulting from a previous query. This enables the exploratory analysis necessary for data archaeology. The QL also can generate a CLASSIC concept definition from a collection so that it can be added to the knowledge base. This does not simply create a node in the knowledge base and store a set of individuals under that node; instead, the concept definition specifies the conditions under which an individual belongs to the concept. Thus, as the knowledge base changes, the individuals that satisfy the concept are automatically recomputed.

To apply IMACS to a domain, the users must construct a "domain model" that embodies their way of thinking about the domain. When expressed in CLASSIC, a domain model consists mainly of concepts and roles. Typically, the overall structure is taxonomic, reflecting categories and subcategories within the domain. Figure 1 shows part of the domain model for the department store example.

The nodes in the taxonomy are implemented as CLASSIC concepts, complex descriptions involving roles, role restrictions, and individuals. For example, consider the concept of a CUSTOMER. CLASSIC is used to specify what roles can apply to CUSTOMER individuals and to represent constraints on the fillers of the roles, e.g., that the purchases role of a CUSTOMER must be filled by PURCHASE individuals. CLASSIC also provides a "trap door" for specifying computations in the host language (Lisp or C); these are called test functions. Test functions can be associated with a concept to define properties of the concept that cannot be expressed in the CLASSIC concept definition language. In this domain model, the definition of SALE-PURCHASE uses a test function that examines the date of a purchase to see if the purchase occurred during a sale.

After a domain model has been defined, there must be a mechanism for populating it from the databases. This is an active area of research [1, 8, 9], and one that our project has explored in depth [7, 12]. However, for the purposes of this paper, what is relevant is that (1) we define general translation routines that specify mappings from the databases into CLASSIC, (2) we run the translation routines to create a file of CLASSIC commands, thus decoupling the KR system and the database, (3) the file then is loaded into CLASSIC to populate the domain model, and (4) we periodically create a file of incremental updates to the databases and load the file into CLASSIC.

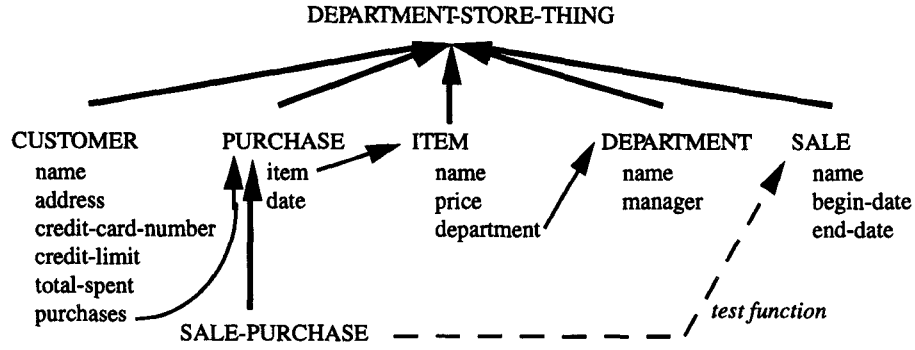


Figure 1: Department store domain model

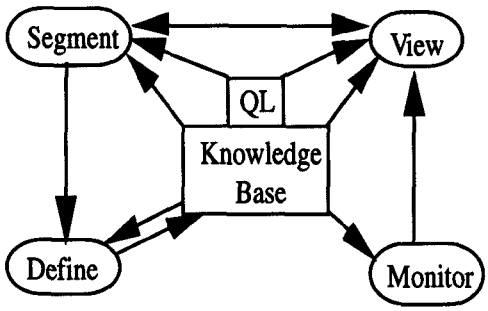


Figure 2: Task Structure for IMACS Interface

## 4 Interface Support For Data Archaeology

As shown in figure 2, the IMACS interface embodies an analysis of data archaeology into four constituent tasks (the arrows between the tasks illustrate typical patterns of activity but do not exhaust the possibilities):

- *viewing* data in different ways, including concept definitions, aggregate properties of concepts, tables of individuals, and graphs;
- *segmenting* data into subsets of analytic interest;
- *defining* new CLASSIC concepts from a segmentation;
- *monitoring* changes in the size and makeup of concepts that result from incremental updates from the databases.

In the remainder of this section, I illustrate how the interface supports each of the tasks with usage scenarios from the department store domain. Throughout this discussion, I emphasize the role and importance of the design principles listed in the introduction:

- combine power and ease of use,
- directly support integrated, iterative data exploration, and
- support users in managing their work over time.

### 4.1 Task 1: Viewing Data

An analyst views data first, to "get a feel for the data", e.g, to determine the attributes that characterize a customer, the average amount customers spend, or the amount spent by particular customers, and second, to formulate questions to be investigated, e.g., "Is there any correlation between the percentage of purchases customers make during sales and the total amount they spend?"

A necessary part of analyzing data is selecting characteristics of the data to view. For example, in looking at a table of customers, an analyst might wish to see the total amount spent, the number of purchases made, and the percentage of purchases that were made during sales. Whether a view is appropriate depends on the type of data being examined and the analytic task. The information an analyst might want to see for a particular object cannot be limited to data stored on roles of that object. For example, to determine the percentage of purchases a customer made during sales would involve accessing the value of the purchases role, determining which purchases were SALE-PURCHASEs, then dividing the number of sale purchases by the total number of purchases.

These considerations led to a design decision that all views should be driven from *templates*, declarative specifications of the data to be displayed, and that all such templates should be user-editable. (While we illustrate the use of templates for table views, similar remarks hold for other views.) Each template consists of a set of expressions in the QL and column headings. Templates are associated with concepts in the knowledge base. Whenever a table of individuals belonging to a concept, e.g. CUSTOMER, is displayed, the template for CUSTOMER is used to construct the view. And from a display of a table of CUSTOMERs, an analyst can edit the template for CUSTOMER. Finally, templates are inherited down the concept hierarchy and are composed to determine the complete view for a particular table: if we ask to see a table of the instances of CUSTOMER, and CUSTOMER is a specialization of PERSON, the templates for both PERSON and CUSTOMER would be used to construct the table. The template-based scheme described here generalizes that of [15].

The table of CUSTOMERs in figure 3 shows the total amount spent, number of purchases, and percent sale purchases for each CUSTOMER. It also shows the template editor that was used to create this template. The first two pieces of information are stored directly on roles of each CUSTOMER, while the third ("% sale purchases") requires the following QL expression (<x> is a variable that ranges over each CUSTOMER in the table):

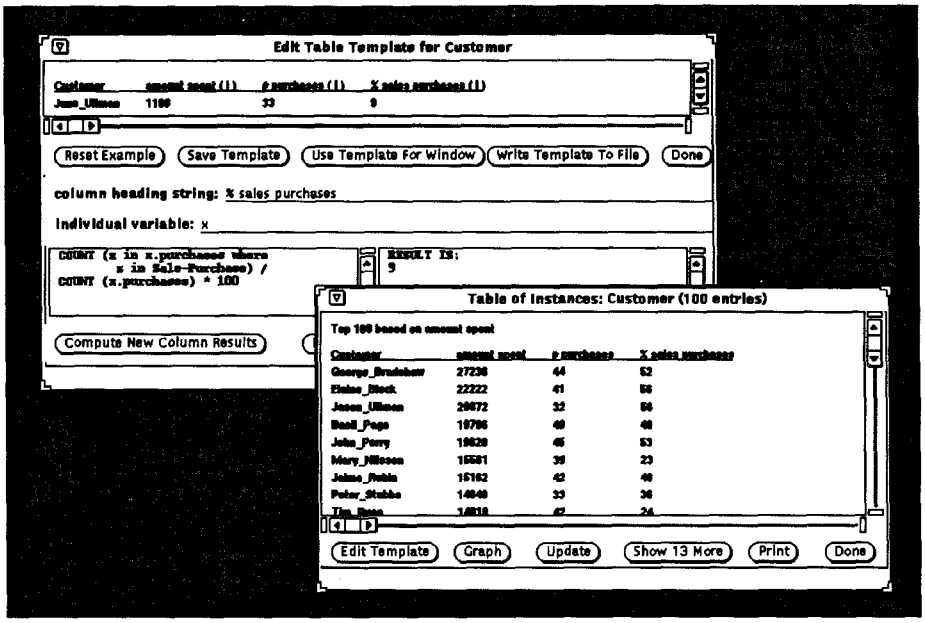


Figure 3: Table of CUSTOMERs and template editor

100 * (COUNT (z in < $x$ >.purchases
where z in SALE-PURCHASE) /
COUNT (<x>.purchases))

Note that the template-based scheme does not require extra work of an analyst: unless the database is extremely small and simple, the analyst always must select certain characteristics of the data to view. And the work of creating a template benefits both its creator and other analysts in the future. As mentioned, one of the shortcomings of current tools for data analysis is that they do not support management of work over time. In other words, the work of viewing and segmenting data I do as part of one analysis does me little good when I do another analysis. While it is clear that the templates I create will be useful for me in the future, the template-based view scheme also affords important opportunities for division of labor and cooperation with other analysts. First, while at least one analyst working in a particular domain must be familiar with the template editing tool and the QL to create appropriate templates, other analysts can use these templates once they are constructed. Second, when other analysts need to view data somewhat different than existing templates provide, their task is to edit an existing template, rather than create one from scratch. Presumably they understands the effects of the QL expressions in this template (if not their detailed syntax), since they have been viewing data formatted by the template. The effect is similar to that pointed out in [10] in a discussion of spreadsheets as a medium for cooperation: templates serve as a point of cognitive contact among users that affords a natural division of labor and task-centered, as-needed learning.

An analyst can request various types of graphs and plots, for example, a plot of the individuals in a table based on the values in a particular column of the table. Figure 4 shows a plot of customers based on percent sale purchases. Graphs afford natural opportunities for segmenting data as breaks in a graph suggest segment boundaries. The analyst can indicate segmentation points in a graph with a mouse click; vertical lines (as shown in figure 4) show the boundaries, and the horizontal dotted lines show the boundary elements from the data vector. Thus, figure 4 indicates a three-way segmentation of CUSTOMERs. Selecting the "Segment Based on Intervals" button brings up a menu that presents English paraphrases of the queries that will be generated to segment the data and lets the analyst name the segments.

### 4.2 Task 2: Segmenting Data

The purpose of segmenting data is to create subsets of analytic interest, e.g., customers who buy mostly during sales, or high spending customers, or customers with high credit limits. The presumption is that useful generalizations can be made about such subsets, e.g., that they may respond well to certain sales or are more likely to get behind in their payments. Viewing and segmenting are interwoven tasks: viewing data initially suggests hypotheses and questions, segmenting the data puts these hypotheses into a testable form (by forming categories over which the hypotheses may or may not hold), then further viewing of the segments tests the hypotheses. Figure 5 illustrates these two tasks and their interrelationships.

IMACS provides 3 ways to segment data: with queries, with forms (abstracted from queries), and from graphs (as mentioned above). Each method has its advantages. The power of a general-purpose query language is necessary since it is impossible to anticipate every way that analysts will want to segment data. On the other hand, it is possible to recognize routine segmentation methods in a domain, and this is where forms come in.

Forms capture common, reusable analysis "cliches" in a domain, e.g., segmenting the instances of a concept by the amount of change in a vector attribute (like purchase history) of each instance. The most important aspect of

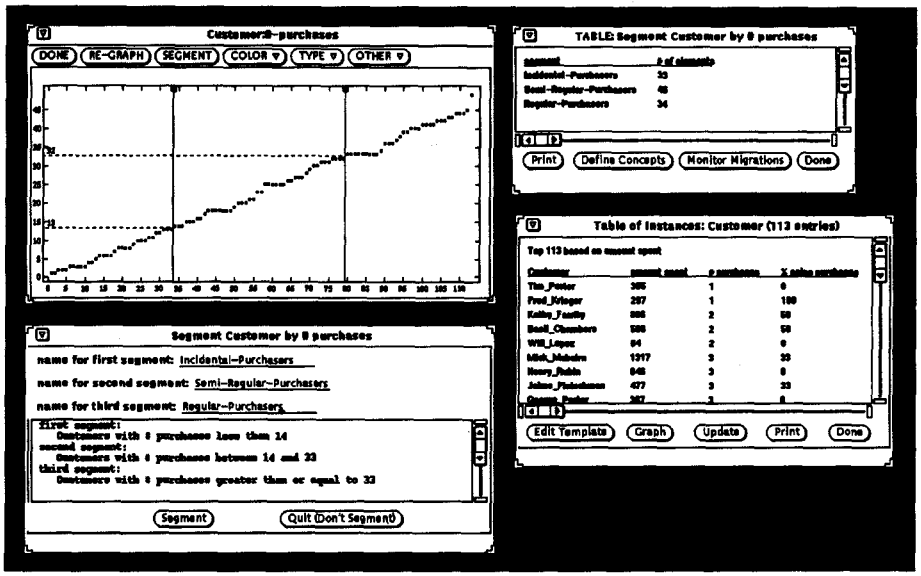


Figure 4: Plot of "% sales purchases" of CUSTOMERs, segmentation form, and resulting segments

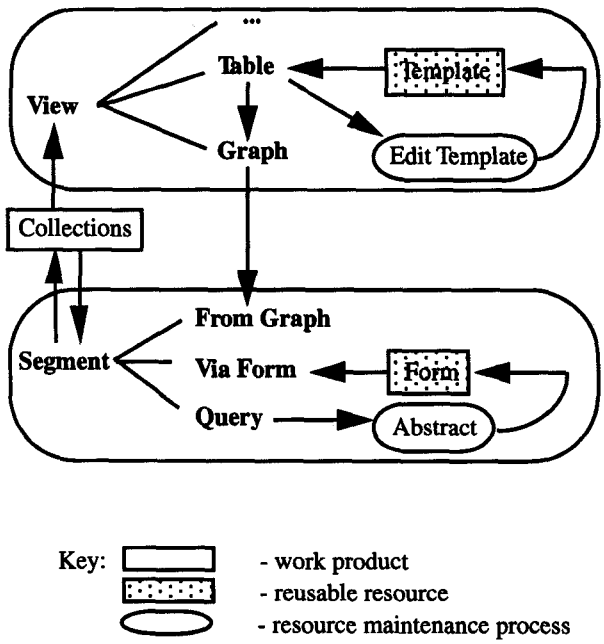


Figure 5: Relationship between Viewing and Segmentation

these forms is that they are all derived from queries in the query language by replacing parts of the queries by variables. Working with the users of a particular IMACS application, we encapsulate the most common queries into forms and save these forms in a library that is loaded at system start-up time; however, if analysts need to construct an ad-hoc query in the query language that they then realize is generally useful, a simple "abstraction" window guides them through the process of creating a form from the query. The observations I made about view templates as reusable resources and media for cooperation apply to forms as well.

Figure 6 shows a form being filled out that will segment customers' purchases by the department of the item purchased; the resulting table might lead the analyst to look for correlations among departments in which customers make their purchases. The analyst specifies iteration over all DEPARTMENTs and CUSTOMERs, thus generating one query for each possible pairing of DEPARTMENT and CUSTOMER individuals. A typical query would be:

```
x in Joe-Smith.purchases.item where
        x.department = Appliances
```

Segmenting initiated from a graph is useful since the graph makes natural boundaries in the data apparent. It is possible to segment from a graph of a column from a table of individuals because the column was defined by a QL expression. In the example we have been considering, the column "% sale purchases" was defined by the expression:

```
100 * (COUNT (z in < x >.purchases
        where z in SALE-PURCHASE) /
        COUNT (<x>.purchases))
```

Suppose an analyst indicates two boundary lines, say at 15% and 40%, in the graph. From these boundaries and from the QL expression that defined the column, queries to segment CUSTOMERs into those with percent of sale purchases greater than 40, between 15 and 40, and less than 15 are generated automatically. For example, the query that defines the second segment is:

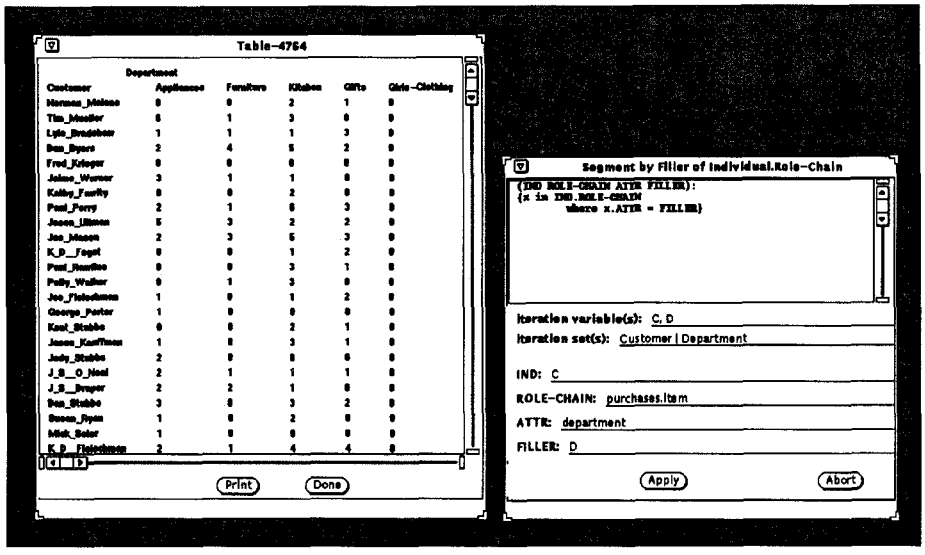


Figure 6: Filling out a form, and the results of applying that form

x in CUSTOMER where
(100 * (COUNT (z in $< x >$.purchases
where z in SALE-PURCHASE) /
COUNT ($< x >$.purchases)) $\geq$ 15) AND
(100 * (COUNT (z in $< x >$.purchases
where z in SALE-PURCHASE) /
COUNT ($< x >$.purchases)) $<$ 40)

The collections generated by queries are first class interface objects that can be analyzed with all available viewing and segmentation operations. This is necessary in order to allow analysts to explore their hypotheses – once the set of CUSTOMERs who make most of their purchases during sales has been defined, the analyst must view this collection to see whether interesting generalizations apply to it, e.g., whether these "Sale Customers" tend to spend more money than other CUSTOMERs. The analyst also might want to further segment "Sale Customers", e.g., into those who tend to buy big-ticket items and those who buy less expensive items. Such information would be useful in deciding what items to put on sale and to feature in customer mailings.

### 4.3 Task 3: Defining Concepts

When an analyst decides that a collection is of permanent interest, it can be turned into a CLASSIC concept. A CLASSIC concept definition is generated automatically, added to the knowledge base, and populated with all individuals that satisfy the definition. The major reason for forming a concept from a collection is to keep track of changes to the concept over time. This brings us to the next task.

### 4.4 Task 4: Monitoring Changes

Real world databases rarely are static. Typically, updates arrive periodically, say, once a month. It is desirable to track the size and makeup of various classes of objects as updates come in. For example, a department store may want to target its sale mailings to customers most likely to respond to them, so, as customers become "sale customers", the store will want to add them to its mailing list. Or, if other customers decrease their purchases during sales, the store may want to send them a special mailing to try to get them to increase their purchases again.

MONITOR: Migration among
LOW-SPENDERS, MEDIUM-SPENDERS, HIGH-SPENDERS

| | Migration To: | | |
|---|---|---|---|
| Migration From: | LOW-SPENDERS | MEDIUM-SPENDERS | HIGH-SPENDERS |
| LOW-SPENDERS | 22 | 5 | 0 |
| MEDIUM-SPENDERS | 2 | 41 | 12 |
| HIGH-SPENDERS | 3 | 2 | 26 |

Figure 7: Monitoring Migration among a Set of Concepts

The interface allows users to specify four types of changes to be monitored: migration (1) into a particular concept, (2) out of a particular concept, (3) between two specified concepts, or (4) among a specified set of concepts. We will consider an example of the latter type to illustrate the monitoring functionality in a little more detail. Suppose an analyst defines three sub-concepts of CUSTOMER, based on the amount of money spent, LOW-SPENDERS, MEDIUM-SPENDERS, and HIGH-SPENDERS. If the analyst specifies that all migrations among the three concepts should be tracked, the system will monitor all the CUSTOMER individuals during an incremental update to the knowledge base. For each CUSTOMER individual, it will track what concept it was a member of before and after the update and will accumulate a record of all the migrations among the three sub-concepts.

After the incremental update is complete, the analyst is alerted if any specified changes occurred. For every set of concepts that is monitored, a matrix of all the migrations is presented (see figure 7). The analyst then can analyze any set of migrating individuals using all the normal interface functionality.

To conclude this section, I summarize the most significant technical contributions of the interface:

- template-driven, user-editable views and abstraction of queries into forms – analysts can build reusable resources as a natural part of the analysis process;
- segmenting initiated from graphs – a powerful method of integrating viewing and segmentation;
- monitoring for changes to concepts – analysts are notified of conditions that require their attention.

## 5 Discussion

IMACS has been developed in cooperation with a data analysis group from AT&T. We have developed a domain model for their application and populated the domain model with data from several databases. The analysts have used IMACS to do some realistic analysis and believe that it has the potential to revolutionize their job. Other groups within AT&T are also interested in using IMACS.

There are several areas for future work. First, based on user feedback, we are developing new interface abstractions to support directly certain common types of analysis. Second, we are considering extending the range of conditions that IMACS can monitor. Third, we are exploring ways to allow analysts to abstract larger scale regularities in their work, involving interrelated, possibly conditional sequences of segmenting and viewing. It is unclear whether such patterns could be performed automatically; rather, we suppose that they will be done interactively, with the system guiding the analyst.

In summary, data archaeology is a task of great practical significance to many organizations that offers interesting challenges to researchers. IMACS is a comprehensive support system that uses knowledge representation and reasoning as a core technology. The interface embodies interesting task-independent design principles and provides powerful support for the data archaeology task.


### Acknowledgements

I would thank all the members of the IMACS project for their contributions to this work, particularly Peter Selfridge, Ron Brachman, Tom Kirk, and Fern Halper.


## 6 References


1. Abarbanel, R.M. and Williams, M. D. A Relational Representation for Knowledge Bases. In *Expert Database Systems.* L. Kerschberg, Ed. Benjamin-Cummings. Redwood City, CA. 1987.

2. Biggerstaff, T.J. and Perlis, A.J. *Software Reusability, Volume 1: Concepts and Models.* Addison-Wesley. Reading, MA. 1989.

3. Biggerstaff, T.J. and Perlis, A.J. *Software Reusability, Volume 2: Applications and Experience.* Addison-Wesley. Reading, MA. 1989.

4. Borgida, A., Brachman, R.J., McGuinness, D.L, and Resnick, L. A. CLASSIC: A Structural Data Model for Objects. In *SIGMOD International Conference on Management of Data.* 1989.

5. Borgida, A., Mylopolous, J., Schmidt, J., and Wetzel, I. Support for Data-Intensive Applications: Conceptual Design and Software Development. In *Database Programming Languages: Proceedings of 2nd International Workshop.* Hull, R., Morrison, R., and D. Stemple, Eds. Morgan Kaufmann. Los Altos, CA. 1990.

6. Brachman, R.J., McGuinness, D.L., Patel-Schneider, P.F., Resnick, L.A., and Borgida, A. Living with CLASSIC: When and How to Use a KL-ONE-Like Language. In *Formal Aspects of Semantic Networks,* J. Sowa, Ed. Morgan Kaufmann. Los Altos, CA. 1990.

7. Brachman, R.J., Selfridge, P.G., Terveen, L.G., Altman, B., Borgida, A., Halper, H., Kirk, T., Lazar, A., McGuinness, D.L., and Resnick, L.A. Knowledge Representation Support for Data Archaeology. In *1st Conference on Information and Knowledge Management.* pp. 457-464. 1992.

8. Mays, E. et al. A Persistent Store for Large Knowledge Bases. In *Conference on AI Applications.* pp. 169-175. 1990.

9. Metaxas, D. and Sellis, T. A. Database Implementation for Large Frame-Based Systems. In *International Conference on Data and Knowledge Engineering for Manufacturing and Engineering.* pp.19-25. 1989.

10. Nardi, B.A., and Miller, J.R. Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development. *International Journal of Man-Machine Studies.* 34 (1991), 161-184.

11. Piatetsky-Shapiro, G. and W. J. Frawley. *Knowledge Discovery in Databases.* AAAI Press. Menlo Park, CA. 1991.

12. Selfridge, P.G. and Brachman, R.J. Supporting a Knowledge Based Software Information System with a Large Code Database. In *KBMS Workshop, National Conference on Artificial Intelligence.* 1990.

13. Schön, D. *The Reflective Practitioner.* Basic Books, New York. 1983.

14. Suchman, L.A. Office Procedures as Practical Action: Models of Work and System Design. *Transactions on Office Information Systems.* 1,4 (1983), 320-328.

15. Terveen, L.G. and Wroblewski, D.A. A Collaborative Interface for Editing Large Knowledge Bases. In *National Conference on Artificial Intelligence.* pp. 491-496. 1990.