Integrated Support for Data Archaeology[1]

Ronald J. Brachman
Peter G. Selfridge
Loren G. Terveen
Boris Altman
Alex Borgida (Rutgers University)

Fern Halper
Thomas Kirk
Alan Lazar
Deborah L. McGuinness
Lori Alperin Resnick

AT&T Bell Laboratories
Murray Hill, NJ 07974

Abstract

Corporate databases are potentially rich sources of new and valuable knowledge. Various approaches to "discovering" or "mining" such knowledge have been proposed. We identify an important and previously ignored discovery task, data archaeology. *Data archaeology is a skilled human task, in which the knowledge sought depends on the goals of the analyst, cannot be specified in advance, and emerges only through an iterative process of data segmentation and analysis. We describe a system that supports the data archaeologist with a natural, object-oriented representation of an application domain; a powerful query language and database translation routines; and an easy-to-use and flexible user interface that supports interactive exploration. A formal knowledge representation system provides the core technology that facilitates database integration, querying, and the reuse of queries and query results.*

1. Introduction

Databases are ubiquitous in virtually all aspects of society, from home use on personal computers to massive corporate and government distributed information systems. Data usually is collected to support mundane concerns like billing and inventory control or for passive record-keeping. Recently, however, as evidenced by workshops such as this one, such data is being seen as a source of new and potentially useful strategic information about customers, products, trends, and organizations. A corporate manufacturing database may contain information that could be used to streamline production; a retail database of customer purchases may contain valuable indicators of future behavior; and a scientific database may contain evidence of underlying causality in an experiment. The key issue is what techniques can be used to extract this hidden information? This question has been the focus of the KDD workshops.

One general approach to discovering knowledge implicit in databases is data *mining* or *dredging* [11, 14]. This tends to emphasize the use of unguided, automatic statistical or machine-learning mechanisms to search for implicit patterns, and has been found valuable in numerous applications. But not all data analysis tasks can use the coarse-grained, relatively inflexible sieve of a dredger. In these other tasks, answers are not buried as full-blown nuggets waiting to be discovered, but rather emerge through an iterative, dialectic process that requires constant human participation. This process more resembles the highly skilled work of an archaeologist than that of a miner or dredger.

What we then might call *data archaeology* is fundamentally a human task. Consequently, the issue of appropriate support tools becomes crucial. Currently, data archaeologists rely on

[1]This paper borrows substantially from one of the same name, to appear in *The International Journal of Intelligent and Cooperative Information Systems.*

conventional database management technology (relational databases and SQL), which fails to provide adequate support for several reasons: it suffers from passive and inflexible *representation* of data; tedious, complicated, and error-prone *access* methods; poor support for tentative, iterative *exploration* of data; and no provision for *managing work over time*, for example, reusing general queries and monitoring analyzed categories over time[2].

In this paper, we sketch an approach to the support of data archaeology that addresses these problems. Our work builds on knowledge representation and advanced interface technologies. It features natural, user-centered domain representations, flexible access mechanisms that combine the power of general-purpose query languages with the ease of use of form-based methods, full support for iterative exploration, and means to reuse work and manage analyses over time. We have implemented a generic system called IMACS (Interactive Market Analysis and Classification System) that can be applied to different application domains [7]. We elaborate on the task of data archaeology, showing how current technology does not support it well; describe our approach and system, arguing briefly how IMACS provides more adequate support; and provide a simple example of an analysis task performed using IMACS.

2. Data Archaeology

The task of an archaeologist is to derive knowledge from a study of artifacts and material remains from the past. Archaeologists extract buried objects with surgical skill, then interpret the objects in the relevant historical and cultural context. This is a dialectic process, in which prior knowledge is used to form plausible hypotheses about the nature and use of objects, but evidence derived from consideration of the objects may lead to revision of previously held beliefs. It may also lead to new goals and suggest promising places for the next round of digging.

Now, suppose you are a *data* archaeologist, seeking to derive new knowledge from a database of customer information maintained by a large department store. Your general goal is to understand and predict customer behavior, and understand how this behavior might change in response to actions by the store, such as mailings and sales. You might begin by focusing in on one attribute of customers, say their credit limit, and using it to segment customers into sub-groups. Your next step could be to compare the sub-groups to attempt to discover useful generalizations about them, for example, that customers with high credit limits tend to spend more than other customers, or are less likely to make purchases during sales, or are more likely to default on their debts. Patterns that emerge from looking at the segments might lead you to abandon or modify some of these hypotheses or to re-segment customers (e.g., on the basis of how long they have been credit customers) to see whether the hypotheses hold over the new segments. What you find may in turn lead you to form new hypotheses, or segment customers on a different attribute. Key aspects of this task include that your goals change in response to what you discover, and it is impossible to predict in advance what you will learn.

As a human task, data archaeology requires appropriate support tools. To derive requirements for such tools, we need first consider the characteristics of the task in more detail. We use the department store example to illustrate these characteristics. Suppose a department store maintains the following data:

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

[2]Conventional DB techniques do not provide support for automated discovery either; however we focus here on the iterative, skilled, human portion of the knowledge discovery task.

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:

- 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 overall 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". Key characteristics of this process include:

(1) *Useful patterns may be hidden within a large existing class, like the class of all customers.* For example, it may contain a hidden 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 parameters (such as purchase date) and parameter values (such as "between December 26th and 31st"). The description of such segments can be quite complex. There is no general way to do this sort of segmentation a priori, as might be required for automatic data mining.

(2) *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", the analyst 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 the category itself, e.g., for certain purposes it might be useful to exclude Christmas sales from the definition of a "sale" since so many people make purchases at this time.

(3) *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 monitor 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.

To summarize, the data archaeology task is an interactive human task that cannot be performed by automatic methods alone. Analysis consists of segmenting classes with potentially complex descriptions, viewing and comparing data, and saving and reusing work (both the results of analyses and the techniques used to generate the results). Useful knowledge is discovered through a dynamic process of interacting with the data as hypotheses are formed, investigated, and revised.

While current data management technology, typically relational databases and SQL, has raised the possibility of being able to do such analysis, it does not provide adequate support for data archaeology. This is primarily because this technology has evolved for static, batch-oriented processing over very large amounts of data (for which it is quite suitable). For interactive tasks, it falls short in a number of ways:

(1) problems in the *representation* of data – while the relational model is good for certain types of data storage and retrieval, it is inflexible for exploratory analysis. In addition, real databases often are incorrect unless integrity controls are specially built in.

(2) 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 (both whether you asked for the right thing and whether you received the result you expected).

(3) little or no support for iterative *investigation, exploration,* and *visualization* of data – it is difficult to analyze or visualize the results of a query to determine how useful they are. Graphical views of data, if available, are not interactive – one cannot get more information directly from the graph. It is hard to vary a query slightly to compare its results to the results returned by the original query.

(4) no support for *managing work over time* – there is no way for analysts to recognize and reuse common patterns in their queries. Further, limitations of SQL often lead analysts to write a query simply to extract all potentially relevant data into a very large flat file and then use a general purpose 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.

(5) no support for tracking category *changes* – because the results of a query are not easily captured, it is very difficult to monitor how the members of a category, like "Sale Customers", change over time.

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

- The system should represent the underlying domain in a natural and appropriate fashion; objects from the domain should be easily incorporated into queries.
- It must be possible to extend the domain representation with new categories formed from queries; these categories must be usable in subsequent queries.
- It should be easy to form tentative segmentations of data, to investigate these segments, and to re-segment quickly and easily; there should be a powerful repertoire of viewing and analysis methods, which are applicable to segments.
- Analysts need support in recognizing and abstracting common analysis (segmenting and viewing) patterns; it must be easy to apply and modify the patterns.
- There should be facilities for monitoring changes in categories over time.

3. Our Approach: Knowledge Representation as a Semantic Core

We have developed a new approach to supporting data archaeology, based on the use of a knowledge representation (KR) system to represent an application domain. The KR system provides a flexible language for describing the domain, and various inference mechanisms serve to manage the domain model as a hierarchy of class descriptions, maintain constraints between descriptions, and provide the foundation for a comprehensive query language.

We use the KR system CLASSIC [4, 6]. CLASSIC is a frame-based knowledge representation language of the KL-ONE family [8]. CLASSIC has three kinds of formal objects :

- *concepts*[3] – structured descriptions of sets of objects formed by composing a limited set of operators (e.g., `Pre-Holiday-Sale` might represent a Sale whose date role is specified to be the week previous to a `Holiday` date); concepts correspond to one-place predicates;

[3] CLASSIC objects are written in `typewriter` font, with concepts (`SALE`) in uppercase, individuals (`Joe-Smith`) capitalized, and roles (`credit-limit`) in lowercase.

- *roles* – two-place predicates that relate individuals (e.g., credit-limit might represent the credit limit of a customer);
- *individuals* – objects in the domain of interest; individuals are given properties by asserting that they satisfy concepts (e.g., Joe-Smith is a GOOD-CUSTOMER) and/or that their roles are filled with other individuals (e.g., the preferred-season of Lawn-Mower is Spring).

Concepts and individuals are organized in a taxonomy or hierarchy. More general concepts subsume more specific concepts in the taxonomy. For example, the concept for a "clearance sale couch" is more specific than both "furniture item" and "sale item" because a couch is a kind of furniture item, and a clearance sale item is a kind of sale item (because a clearance sale is a kind of sale). CLASSIC maintains this taxonomy automatically. Among the deductive inferences that CLASSIC computes are various kinds of completion, including inheritance, combination, and propagation; contradiction detection; classification and subsumption, including classification of concepts and individuals and subsumption of general descriptions; and rule application, using simple forward-chaining rules. CLASSIC also provides a "trap door" for specifying class membership tests in the host language (e.g., Lisp or C); these are called test functions. Test functions are associated with a concept to define properties that cannot be expressed in the concept definition language. They are constructed so that with careful use, the semantics of the declarative part of the language will not be violated.

Several properties of CLASSIC are particularly important in making it the basis for an adequate data archaeology support system. First, it allows a much more natural and expressive representation of the domain of interest than do relational representations. Second, the "schema" (concept hierarchy) can be extended dynamically; that is, new concept definitions can be added, and CLASSIC's inference mechanisms will continue to guarantee consistency among concepts and individuals. Third, it serves as the foundation for a powerful and flexible query language that enables the exploratory sort of analysis that data archaeology requires.

4. The IMACS Architecture

The implemented system based on our approach is called IMACS (Interactive Market Analysis and Classification System). The architecture of IMACS is shown in Figure 1. Note that the interface, query processor, and the CLASSIC KR system form a domain-independent, generic system, while the domain model, translation routines, and databases vary from one application to another.

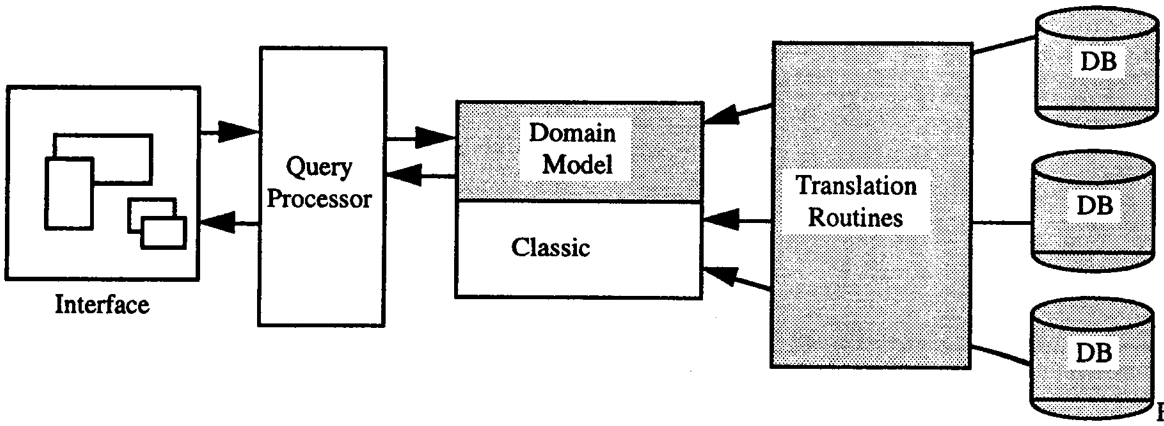


Figure 1: The IMACS architecture (shaded components are application-specific)

4.1 Domain Model

To create an IMACS application, the users must agree on a "domain model" that represents the objects and relationships of the domain in as complete and as natural a fashion as possible. Once conceptual analysis of the domain of interest is completed, the model is implemented as a CLASSIC knowledge base of concepts, roles, and individuals. The domain model forms the core of the system and influences the naturalness of users' interactions with the system. Figure 2 shows a simple domain model for the department store example.

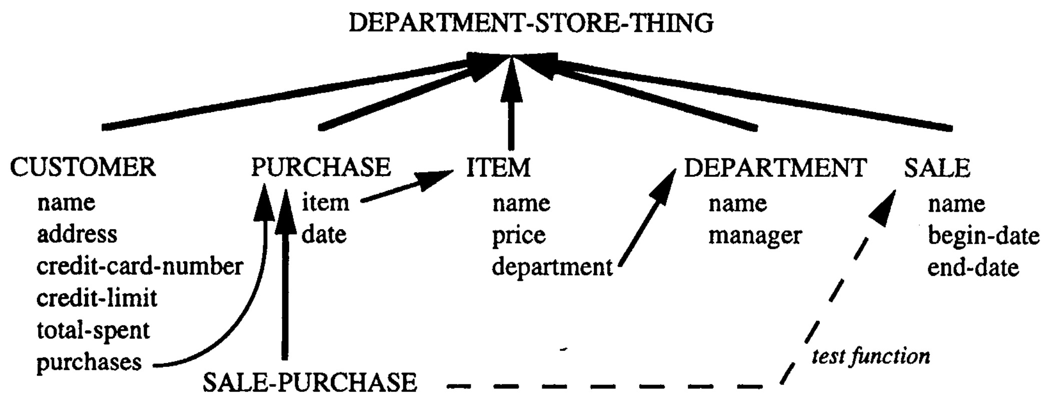


Figure 2: Department store taxonomy

The nodes in the taxonomy are implemented as CLASSIC concepts. For example, consider the concept of an ITEM, a generic description of the properties of all items. An ITEM may record a name, price, department, and other stock information. CLASSIC is used to represent the constraints on the fillers of the roles, and CLASSIC rules and test functions are used to maintain consistency. Concepts may be sub-divided: for example, sub-concepts of ITEM may be created by restricting the department or current-price roles to define concepts like FURNITURE-ITEM or EXPENSIVE-ITEM.

There is a fundamental difference between this kind of domain model and schemas of standard relational databases. Relational schemas are tailored to make particular types of queries run efficiently over large amounts of data. Indeed, the relational model was designed for very large databases and complex, but relatively static, batch-oriented queries. The IMACS domain model is used as a conceptualization tool independent of efficiency considerations. The data model provided by CLASSIC also is much richer than that of the relational model. This allows a more realistic picture of the analysts' domain, which better supports "archaeological" exploration. In addition, the CLASSIC data representation avoids problems like referential integrity and enforces several other kinds of consistency constraints not typically available with relational technology.

4.2 Database Translation

The problem of integrating knowledge bases and databases has received a fair amount of attention [1, 2, 10, 12, 13]. This integration typically is attempted either to augment the number of individuals handled by a KR system or to use the greater flexibility and expressiveness of KR technology to build an interface layer on top of relational data.

There are several ways in which the CLASSIC domain model could interact with the data stored in several DBs. Our current implementation translates the data from our databases into

CLASSIC, and then decouples the domain model from the databases. In other words, we build a version of the data in CLASSIC and work with that. This is adequate for our current application, in which data is updated on a monthly basis. This allows us to translate the data "off line", performing incremental updates once a month. This scheme will not handle huge amounts of data (however, we have completed a prototype persistent version of CLASSIC that will significantly extend the capacity of the system), but we have demonstrated adequate performance on a KB containing tens of thousands of individuals.

To facilitate translation, we have developed a table-driven mechanism that allows users to fill in a form specifying the mapping from relations to CLASSIC; the mechanism then does all the tedious work of translating fields and values [3]. This requires a reasonable amount of upfront effort: for every primitive CLASSIC concept – those that do not have a necessary and sufficient definition – a user must specify SQL code that will fetch the appropriate data for conversion into CLASSIC's format. However, any concepts that analysts define during the process of data analysis are fully compositional. We thus have been able to develop algorithms that can construct SQL from these compositional concepts and the SQL for their primitive subparts; thus, users need specify SQL mappings only for the primitive basis of the concept hierarchy.

4.3 Query Language and Query Processor

Our query language is a set-oriented language designed specifically for CLASSIC, although it borrows from state-of-the-art systems like TDL [5]. It includes constructs for defining sets using logical operators, role-following syntax and conventions, and aggregation mechanisms.

A set formed by a query is called a *collection*. The QL can operate on collections – where a concept like CUSTOMER could appear, so could a collection of CUSTOMERs resulting from a previous query. This is necessary for the iterative, exploratory analysis that characterizes 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.

It is important to note that our query language is substantially more expressive than the CLASSIC concept description language. This lets users describe collections that cannot be described by CLASSIC alone, such as "big ticket" purchasers: "x in Customer.where AVG (x.purchases.item.price) > 250". This gives the analyst an appropriate amount of power for querying while the data is adequately stored in a weaker form.

4.4 The IMACS Interface

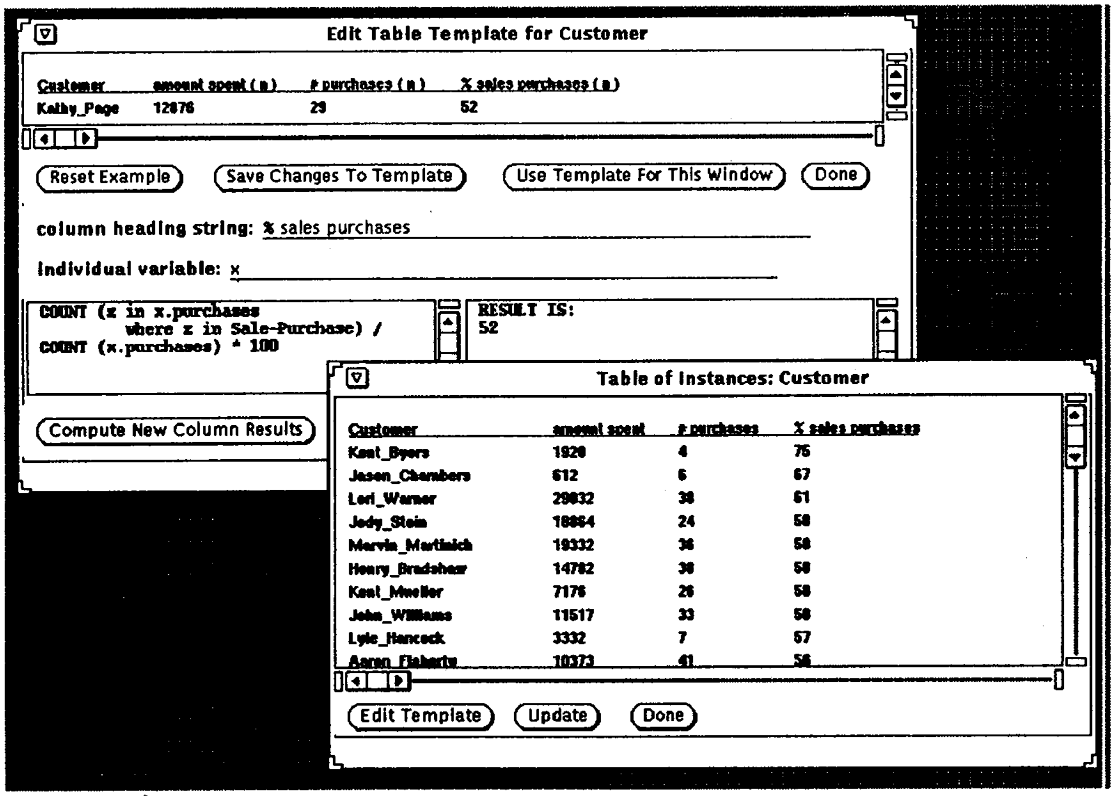
The IMACS interface supports four sub-tasks of the data archaeology process:
- *viewing* data in different ways, including concept definitions, aggregate properties of concepts, tables of individuals, and graphs;
- *segmenting* data into subsets of interest;
- *defining* new CLASSIC concepts from a segmentation;
- *monitoring* changes in the size and makeup of concepts that result from incremental updates to the databases.

4.4.1 Task 1: Viewing Data

An analyst typically views data to first "get a feel for the data", e.g, to discover the attributes that characterize a customer, the average amount customers spend, etc., and second, to formulate questions to investigate, 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 see. 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 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. Each template consists of a set of column headings and expressions in the query language; the QL expression describes how to compute the column entry. 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.


Edit Table Template for Customer
Reset Example
Save Changes To Template
Use Template For This Window
Done
Compute New Column Results
Table of Instances: Customer
Edit Template
Update
Done

Customer	amount spent (n)	# purchases (n)	% sales purchases (n)
Kathy_Page	12876	29	52

column heading string: % sales purchases

individual variable: x

```
COUNT (z in x.purchases
       where z in Sale-Purchase) /
COUNT (x.purchases) * 100
```

```
RESULT IS:
52
```

Customer	amount spent	# purchases	% sales purchases
Kent_Byers	1920	4	75
Jason_Chambers	612	6	67
Lori_Warner	29832	38	61
Jody_Stein	18864	24	58
Marvin_Martinich	19332	36	58
Henry_Bradshaw	14782	38	58
Kent_Mueller	7176	26	58
John_Williams	11517	33	58
Lyle_Hancock	3332	7	57
Aaron_Flaherty	10373	41	56

Figure 3: Table of CUSTOMERs and template editor

Figure 3 shows a table of CUSTOMERs with columns for the total amount spent, number of purchases, and percent sale purchases; it also shows the template editor that was used to create this table. Defining the "% sale purchases" column requires the following QL expression:

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

IMACS supports a variety of interactive graphical visualizations of data. 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 next section gives details on how this is done.

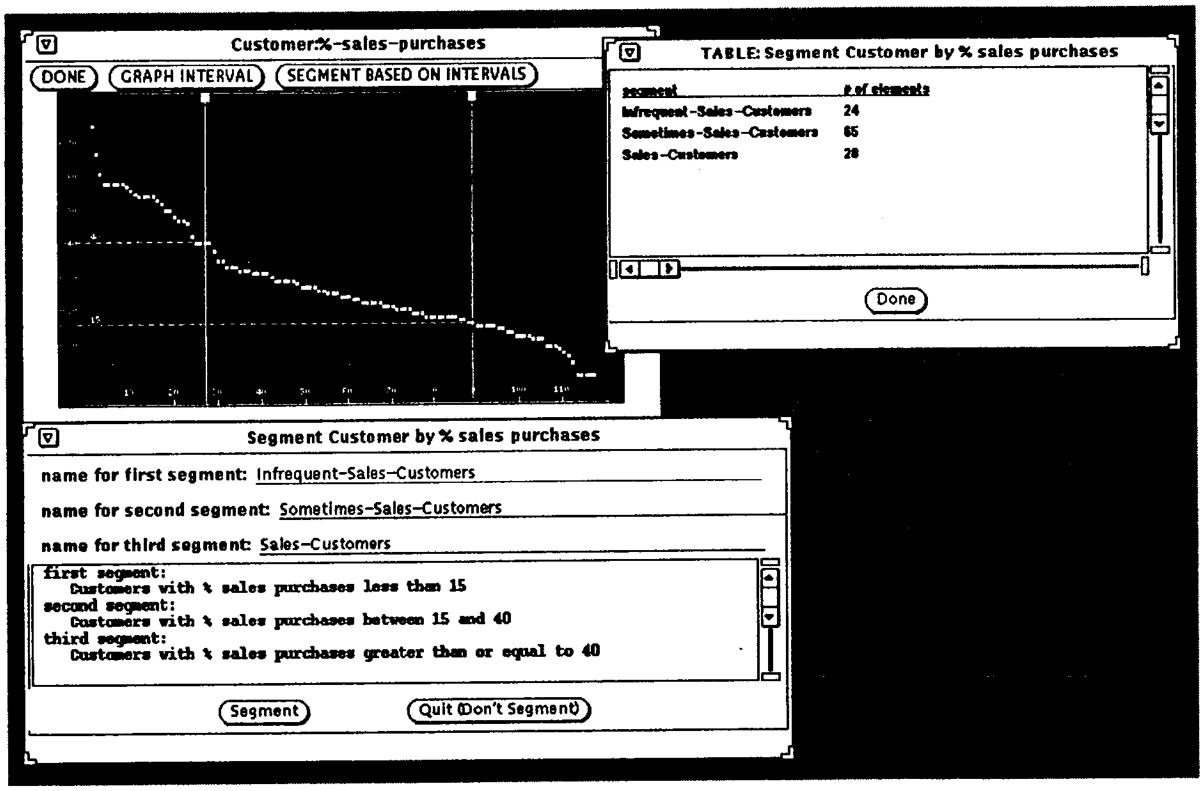


TABLE: Segment Customer by % sales purchases

segment	# of elements
Infrequent-Sales-Customers	24
Sometimes-Sales-Customers	65
Sales-Customers	28

Segment Customer by % sales purchases

name for first segment:	Infrequent-Sales-Customers
name for second segment:	Sometimes-Sales-Customers
name for third segment:	Sales-Customers

```
first segment:
   Customers with % sales purchases less than 15
second segment:
   Customers with % sales purchases between 15 and 40
third segment:
   Customers with % sales purchases greater than or equal to 40
```

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

4.4.2 Task 2: Segmenting Data

The purpose of data segmentation 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.

IMACS provides three ways to segment data: with queries, with forms (abstracted from queries), and from graphs (see Figure 4). Each method has its advantages. The power of a general purpose query language is necessary since it is impossible to predict 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 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 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 common queries into forms and save these forms in a library that is loaded at system start-up time; however, if analysts construct ad hoc queries that they then realize are generally useful, a simple "abstraction" window guides them through the process of creating a form from a query.

Figure 5 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.

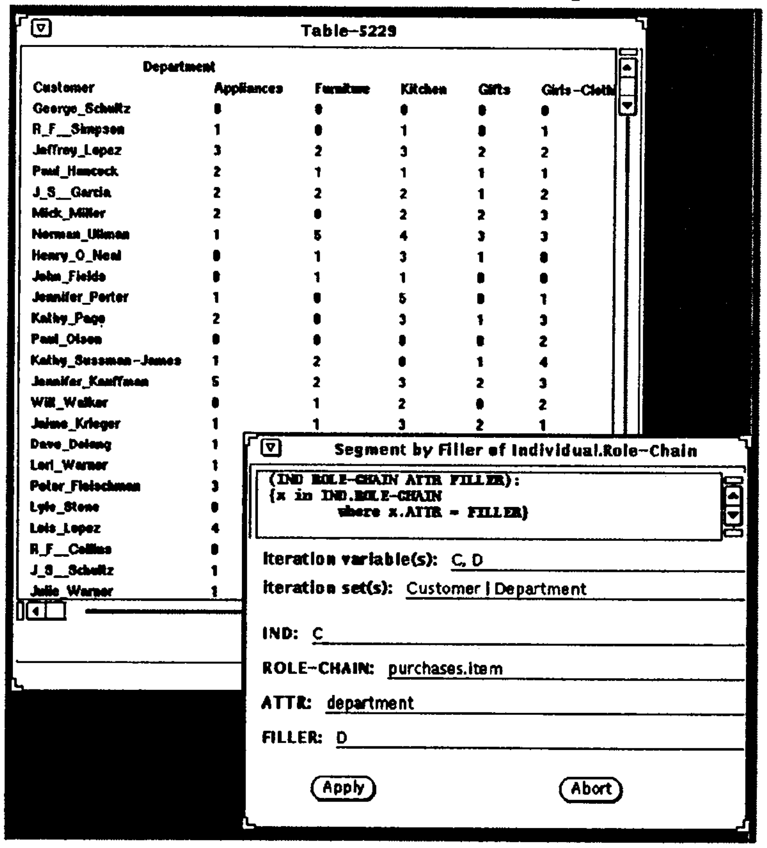


Table-5229

	Department				
Customer	Appliances	Furniture	Kitchen	Gifts	Girls-Cloth
George_Schultz	0	0	0	0	0
R_F__Simpson	1	0	1	0	1
Jeffrey_Lopez	3	2	3	2	2
Paul_Hancock	2	1	1	1	1
J_S__Garcia	2	2	2	1	2
Mick_Miller	2	0	2	2	3
Norman_Ullman	1	5	4	3	3
Henry_O_Neal	0	1	3	1	0
John_Fields	0	1	1	0	0
Jennifer_Porter	1	0	5	0	1
Kathy_Page	2	0	3	1	3
Paul_Olsen	0	0	0	0	2
Kathy_Sussman-James	1	2	0	1	4
Jennifer_Kauffman	5	2	3	2	3
Will_Walker	0	1	2	0	2
Jaime_Krieger	1	1	3	2	1
Dave_Delong	1				
Lori_Warner	1				
Peter_Fleischman	3				
Lyle_Stone	0				
Lois_Lopez	4				
R_F__Collins	0				
J_S__Schultz	1				
Julie_Warner	1				

Segment by Filler of Individual.Role-Chain

```
(IND ROLE-CHAIN ATTR FILLER):
{x in IND.ROLE-CHAIN
      where x.ATTR = FILLER}
```

Iteration variable(s): C, D

Iteration set(s): Customer | Department

IND: C

ROLE-CHAIN: purchases.item

ATTR: department

FILLER: D

Filling out a form that will generate a set of queries. 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

Figure 5: Filling out a form, and result of applying the form

Segmentation 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, and the interface manager maintains records of data that is presented to the user. 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)
```

From this QL expression and from the boundaries indicated by the analyst, IMACS can generate queries to segment CUSTOMERs into those with percent of sale purchases greater than 40, between 15 and 40, and less than 15 automatically.

The collections produced 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.

4.4.3 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 to form a concept is to monitor changes to the concept over time.

4.4.4 Task 4: Monitoring Changes

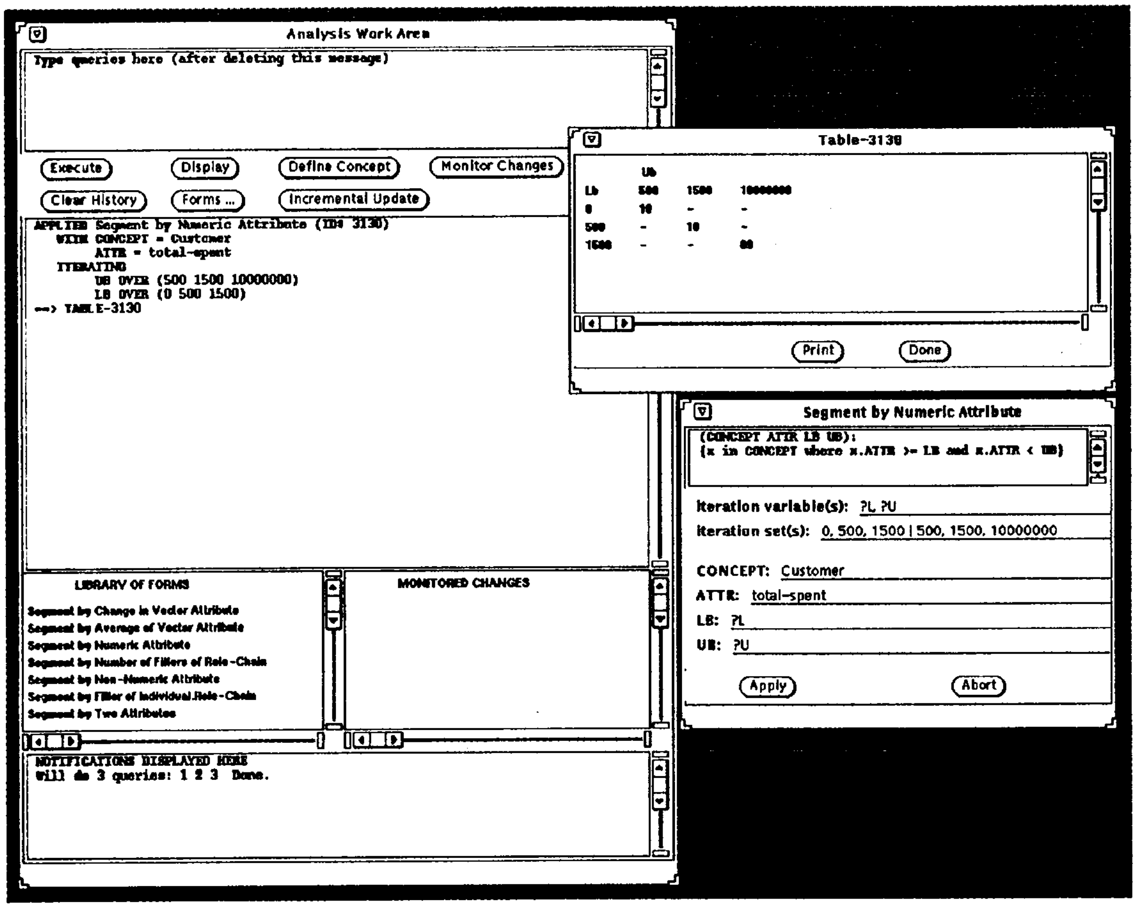
Real world databases rarely are static. In many situations, updates arrive periodically, say, once a month. (We are not currently concerned with situations where data arrives continuously.) It is desirable to monitor the size and makeup of various classes of objects as updates come in. For example, a 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 sale mailing list.

The interface allows users to specify three sorts of changes to be monitored: migration (1) into or (2) out of a particular concept or (3) between two specified concepts. After an incremental update to the knowledge base, the analyst is alerted if any specified changes occurred. The set of migrating individuals is a first-class object in IMACS, and thus the analyst then can analyze this set using all the normal interface functionality.

5. IMACS in Action

Consider again the department store example. Assume that the analyst wants to group customers into categories such as "regular", "semi-regular", and "infrequent", which are useful for predicting customer activity and targeting marketing campaigns.

The analyst begins by browsing the domain model, displayed as a directed graph, locating the CUSTOMER concept, and displaying it in a *concept-at-a-glance* window. This window displays aggregate information about the set of all customers, here the minimum, maximum, and average of the numeric role `total-spent-1991`. She then begins to segment the set of customers by using the form *Segment by Numeric Attribute*. To fill out the form, the analyst specifies the concept to be segmented (CUSTOMER), the role on which to key the segmentation (`total-spent-1991`), and the attribute values that determine the segments. We assume that the analyst wants to divide CUSTOMERs into three segments, those who spent less than $500 ("low spenders"), those who spent between $500 and $1500 ("medium spenders"), and those who spent more than $1500 ("high spenders"). To do so, she specifies appropriate boundary values. Note that specific bounds are only best guesses: it is only through further analysis that the utility of any segmentation can be determined. The results of the segmentation are displayed in an *analysis table* window. This table shows the three segments and the number of customers that fell into each segment. The state of the analysis is shown in Figure 6.



```
Type queries here (after deleting this message)
```

```
APPLIED Segment by Numeric Attribute (ID# 3130)
   WITH CONCEPT = Customer
        ATTR = total-spent
   ITERATING
        UB OVER (500 1500 10000000)
        LB OVER (0 500 1500)
--> TABLE-3130
```

	Ub		
Lb	500	1500	10000000
0	10	-	-
500	-	10	-
1500	-	-	80

```
(CONCEPT ATTR LB UB):
{x in CONCEPT where x.ATTR >= LB and x.ATTR < UB}
```

Iteration variable(s): ?L, ?U
Iteration set(s): 0, 500, 1500 | 500, 1500, 10000000

CONCEPT: Customer
ATTR: total-spent
LB: ?L
UB: ?U

Segment by Change in Vector Attribute
Segment by Average of Vector Attribute
Segment by Numeric Attribute
Segment by Number of Fillers of Role-Chain
Segment by Non-Numeric Attribute
Segment by Filler of Individual Role-Chain
Segment by Two Attributes

```
NOTIFICATIONS DISPLAYED HERE
Will do 3 queries: 1 2 3  Done.
```

Figure 6: Segmenting CUSTOMER by `total-spent-1991`

Let us assume that the analyst is interested in the customers who spent only a small amount at the store in 1991; perhaps because they are not regular customers. To explore the relationship between amount of money spent and regularity of purchasing, the analyst again segments CUSTOMERs using the *Segment by Numeric Attribute* form, this time based on the role `number-of-purchases-in-1991`, to create segments for incidental, semi-regular, and regular purchasers. Suppose the analyst next displays a table of the incidental purchasers and discovers that some spent quite a lot while others spent very little. She now may form the hypothesis that the high spenders are more likely to make purchases during sales.

To investigate this hypothesis, the analyst edits the table view template for incidental purchasers to show not only the amount they spent, but also the percent of purchases they made during sales. She then can specify that she wants to see a scatter plot of the amount spent vs. the percent sale purchases for each incidental purchaser. If the scatter plot indicates a positive correlation between the percent sale purchases and the amount spent, the analyst may recommend that the store increase the number or length of sales it holds or that it advertise sales more extensively. Figure 7 illustrates the current state of the analysis.

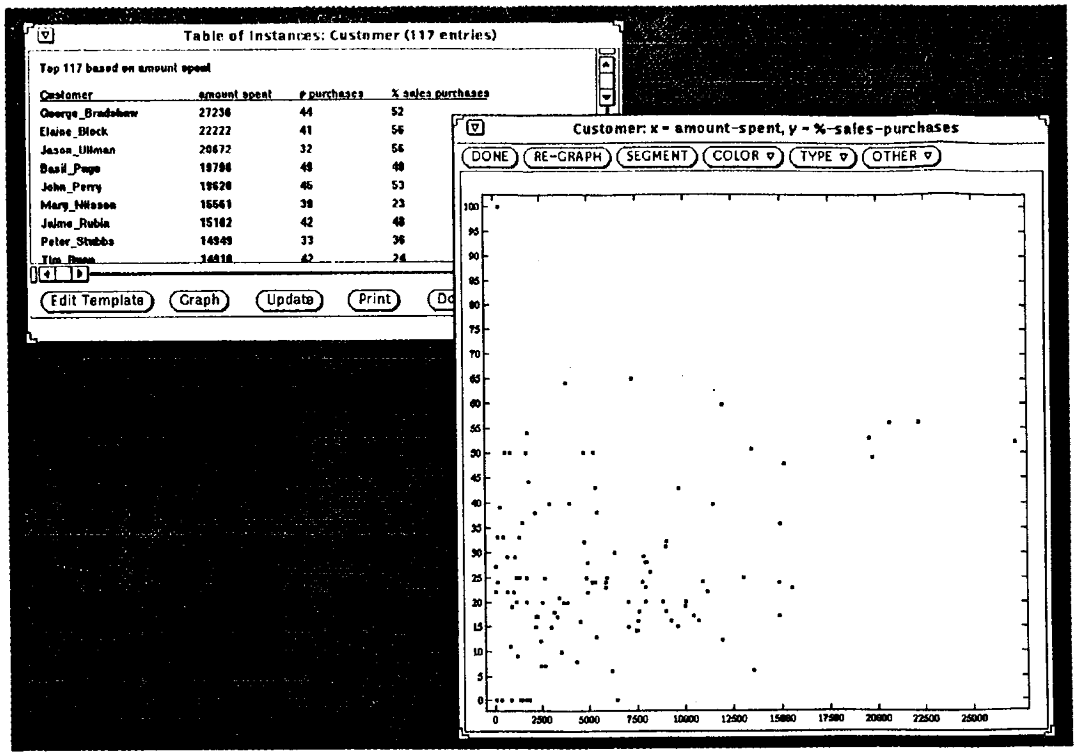


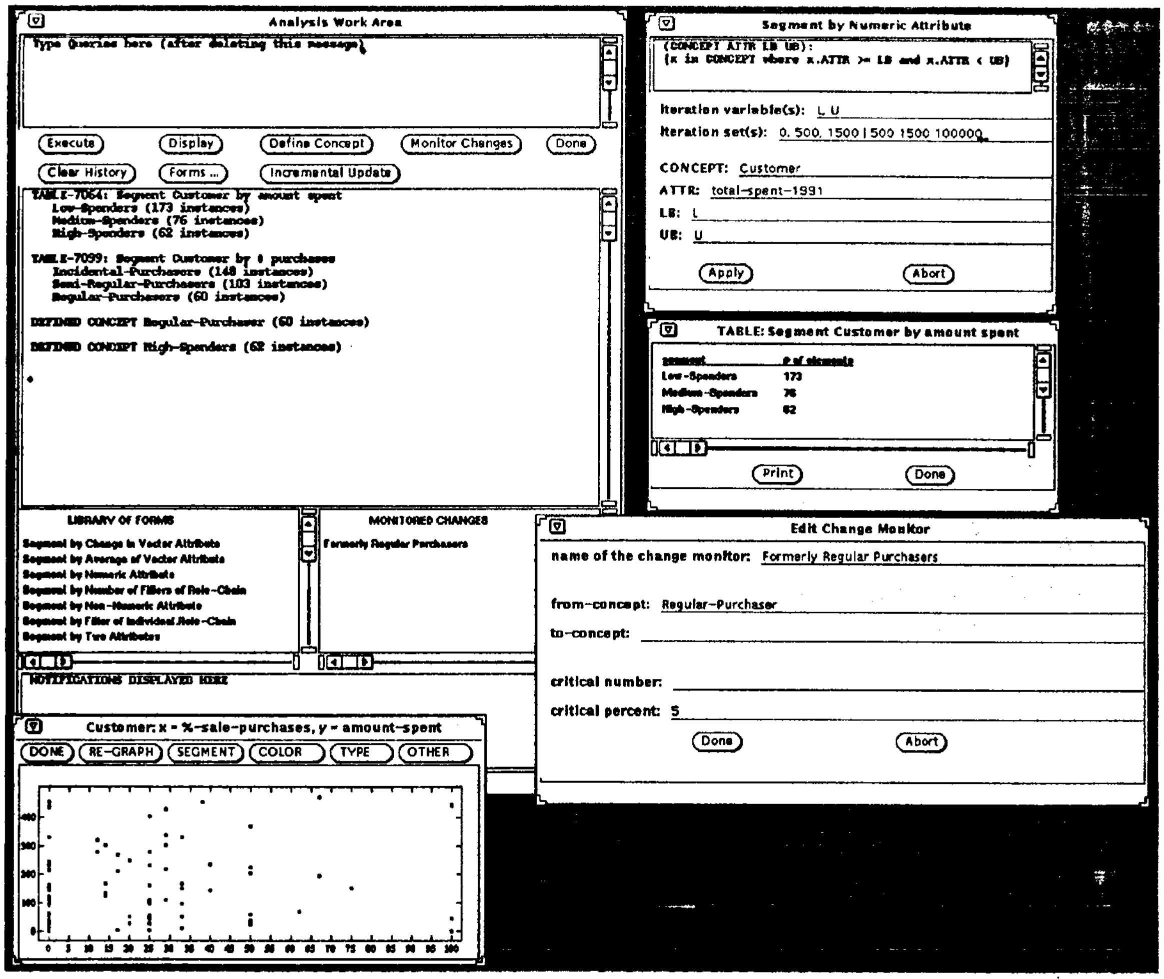
Table of Instances: Customer (117 entries)

Top 117 based on amount spent

Customer	amount spent	# purchases	% sales purchases
George_Bradshaw	27236	44	52
Elaine_Block	22222	41	56
Jason_Ullman	20672	32	56
Basil_Page	19796	48	40
John_Perry	19620	46	53
Mary_Nilsson	15561	38	23
Jaime_Rubin	15162	42	48
Peter_Stubbs	14949	33	36
Tim_[illegible]	14910	42	24

Figure 7: A scatter plot of amount-spent vs. % sale purchases

Finally, assume that the analyst decides that it is appropriate to permanently monitor the size and makeup of some of these segments. She can create CLASSIC concepts for the regular purchaser and high spender segments. By filling out a *Monitor Change* form (see Figure 8), she can specify that she wants to be informed whenever 5% of the customers in the (newly created) REGULAR-PURCHASER concept migrate out of the concept. When incremental updates to the knowledge base are processed, all changes to the classification of individuals in the knowledge base are recorded, and if any of the conditions specified by the analyst are met, the analyst will be notified. The store then can take proper action.



```
Type Queries here (after deleting this message)

TABLE-7064: Segment Customer by amount spent
   Low-Spenders (173 instances)
   Medium-Spenders (76 instances)
   High-Spenders (62 instances)

TABLE-7099: Segment Customer by # purchases
   Incidental-Purchasers (148 instances)
   Semi-Regular-Purchasers (103 instances)
   Regular-Purchasers (60 instances)

DEFINED CONCEPT Regular-Purchaser (60 instances)

DEFINED CONCEPT High-Spenders (62 instances)
```

Segment by Change in Vector Attribute
Segment by Average of Vector Attribute
Segment by Numeric Attribute
Segment by Number of Fillers of Role-Chain
Segment by Non-Numeric Attribute
Segment by Filler of Individual Role-Chain
Segment by Two Attributes

Formerly Regular Purchasers

```
NOTIFICATIONS DISPLAYED HERE
```

```
(CONCEPT ATTR LB UB):
(x in CONCEPT where x.ATTR >= LB and x.ATTR < UB)
```

Iteration variable(s): L U
Iteration set(s): 0, 500, 1500 | 500 1500 100000
CONCEPT: Customer
ATTR: total-spent-1991
LB: L
UB: U

segment	# of elements
Low-Spenders	173
Medium-Spenders	76
High-Spenders	62

name of the change monitor: Formerly Regular Purchasers
from-concept: Regular-Purchaser
to-concept:
critical number:
critical percent: 5

Figure 8: The IMACS interface at the end of the analysis

6.3 Future Work

First, we are working on extending our architecture to handle much larger volumes of data. We are exploring two approaches: (1) using a persistent version of CLASSIC, and (2) building a "loosely coupled" system, where there is a dynamic link between CLASSIC and the databases, and data remains in the databases until it is needed.

Second, we will integrate automatic data mining techniques (both statistical and machine learning) into IMACS. While we have focused thus far on supporting the human aspects of knowledge discovery, we consider automatic techniques to complement our approach, and we are beginning to explore the appropriate role of such techniques in our system.

Finally, we are evolving the interface to make it even easier to perform common sorts of analyses. In response to feedback from our data analyst partners, we have identified analyses that should be easy to perform but are not, and we are adding abstractions to the interface and query language to facilitate these analyses.

7. Conclusion

The current wisdom is that interactive systems provide "perhaps the best opportunity for discovery in the short term" [9], partly because human guidance is needed to specify what data is interesting. We go further: there is a large class of data analysis tasks that are best seen as *data archaeology*, in which even the analyst can discover interesting knowledge only through an iterative, exploratory process. IMACS uses knowledge representation as the core technology for supporting this task, and adds database translation routines, a query language, and a comprehensive interface to provide a powerful tool for the data archaeologist. Initial experience with this system has been highly favorable, and promises to generate new practical and research problems.

8. References

1. Abarbanel, R.M., and M. D. Williams, A Relational Representation for Knowledge Bases, in: *Expert Database Systems*, L. Kerschberg, ed., Benjamin-Cummings, 1987.
2. Bernstein, P.A., Database System Support for Software Engineering - An Extended Abstract, *Proc. 9th ICSE,* pages 166-178, 1987.
3. Borgida, A. and R.J. Brachman, Coupling Description Reasoners to Databases, submitted for publication
4. Borgida, A., Brachman, R.J., McGuinness, D.L, and L. A. Resnick, CLASSIC: A Structural Data Model for Objects, *Proc. 1989 ACM SIGMOD Int'l. Conf. on Management of Data*, 1989.
5. Borgida, A., Mylopolous, J., Schmidt, J., and I.Wetzel, Support for Data-Intensive Applications: Conceptual Design and Software Development, in *Database Programming Languages: Proc. of 2nd Int'l Workshop*, Hull R., Morrison, R, and D.Stemple, Eds., Morgan Kaufmann, 1990.
6. Brachman, R.J., McGuinness, D.L., Patel-Schneider, P.F., Resnick, L.A., and A. Borgida, Living with CLASSIC: When and How to Use a KL-ONE-Like Language, in: *Formal Aspects of Semantic Networks*, J. Sowa, Ed., Morgan Kaufmann, 1990.
7. Brachman, R.J., Selfridge, P.G., Terveen, L.G., Altman, B., Borgida, A., Halper, F., Kirk, T., Lazar, A., McGuinness, D.L., and L.A. Resnick, Knowledge Representation Support for Data Archaeology, *Proceedings of the ISMM International Conference on Information and Knowledge Management (CIKM-92)*: 457-464, Baltimore, MD, November 1992
8. Brachman, R.J. and J.G. Schmolze, An Overview of the KL-ONE Knowledge Representation System, *Cognitive Science* 2: 171-216, April-June, 1985
9. Frawley, W.J., Piatetsky-Shapiro, G., and C.J. Matheus, Knowledge Discovery in Databases: An Overview, in [14].
10. Kim, W., Ballou, N., Chou, H-T., Garza, J. and D. Woelk, Integrating an Object-Oriented Programming System with a Database System, *Proc. OOPSLA '88*: 142-152, 1988.
11. Krishnamurthy, R., and T. Imielinski, Research Directions in Knowledge Discovery, *SIGMOD Record* 20 (3): 76-78, September, 1991.
12. Mays, E., et al., A Persistent Store for Large Knowledge Bases, *Proc. 6th Conf. on AI Applications*: 169-175, 1990.
13. Metaxas, D., and T. Sellis, A Database Implementation for Large Frame-Based Systems, *Proc. 2d Int'l. Conf. on Data and Knowledge Engineering for Manufacturing and Engineering*: 19-25, 1989.
14. Piatetsky-Shapiro, G. and W. J. Frawley, editors, *Knowledge Discovery in Databases,* AAAI Press, 1991.