SISU rapport
nr 5

HYBRIS

- A first step towards efficient information resource management

Jesper Lundh & Peter Rosengren

SISU
Swedish Institute for Systems Development
Box 1250, S-164 28 Kista, Sweden

Table of contents:

SISU Rapport nr 5:
HYBRIS - A first step ...
Jesper Lundh & Peter Rosengren
1989

ABSTRACT

HYBRIS is a hypertext-based tool that allows inexperienced computer users to navigate in and retrieve information from large corporate databases at a conceptual level. The information in the databases is represented in a conceptual model which is called an *information map*. It shows important business-oriented concepts and how they are interrelated. By pointing and clicking directly in the information map, users can retrieve information from the databases. Constraints that restrict the information search can be formulated. The graphical query is then translated to SQL (Structured Query Language) and sent to the database. The result of the SQL-query is brought back to HYBRIS where the user have different alternatives for manipulating the result.

SISU Rapport nr 5:
HYBRIS - A first step ...
Jesper Lundh & Peter Rosengren
1989

1. Introduction

HYBRIS has been developed by the *Swedish Institute for Systems Development, SISU*, in a joint project together with *Swedish Telecom*. The HYBRIS development team consists of three members - *Jesper Lundh*, *Stefan Paulsson* and *Peter Rosengren*. The project leader is *Björn Nilsson* who also provided the original idea for HYBRIS.

SISU Rapport nr 5:
HYBRIS - A first step ...
Jesper Lundh & Peter Rosengren
1989

1.1. Information Resource Management

Swedish Telecom is a very large and decentralized organization. During the last decades they have built numerous database systems and their investments in data are enormous. The problem is that data is stored in different types of databases - relational, network, hierarchical or simply flat files - and that the database systems run on different kinds of computers - Vax/VMS, IBM mainframes, Unix-machines etc. Since it is a decentralized organization, the computer systems are located at different sites all over Sweden. New systems are built throughout the organization without the use of common and standardized concepts, making the situation even worse.

Today nobody has an overview of all the information available in the organization. Executives and other decision makers have to make important decisions based on incomplete information even though the necessary information is stored somewhere in the corporate databases. New databases are built with large costs for collecting data which might already have been collected at some other site. This is a problem shared by most large organizations.

To summarize, there are three major sub-problems: You have to know that data *exists*, *where* it is and *how* to get it.

If the *information resource* could be managed efficiently and if users and systems developers could easily access all available information, the benefits would be tremendous. Decision makers could use this information resource as a foundation when making important strategic decisions and new systems could be built by reusing existing information.

To achieve efficient information resource management, one has to consider three main activities:

- The development of a conceptual model that describes the business.
- The design of a graphical tool which makes it easy for inexperienced computer users to navigate in the conceptual model at a user-oriented level. The tool should contain a uniform graphical query language that allows users to retrieve all available information, no matter where and how it is stored.
- The third activity is to further develop this tool to make it an instrument for systems developers when building new information systems.

1.2. HYBRIS today

HYBRIS is a first step towards such a tool. Today HYBRIS functions as a tool for retrieval of information from relational SQL-databases. A prototype for splitting SQL-queries in a distributed environment has been implemented [Franzén88], but major work remains to be done in this area. HYBRIS has an intergrated support tool for systems developers that facilitates the construction of information maps given a database schema. But we are far from an instrument that supports the construction of new information systems.

Currently, we concentrate on testing and evaluating the capabilities of HYBRIS as a graphical query interface for relational databases in real world applications. It is important to stress that HYBRIS is not tailored for *Swedish Telecom*. The tool may be used by any large company that has made a conceptual model of their business or portions of their business. As a matter of fact, HYBRIS has already been customized for a sales support system at *Pharmacia Biotechnology Inc.*, a multinational medical producer.

HYBRIS currently runs on a Macintosh with the host database on a Vax/VMS-machine. The graphical interface of HYBRIS has been developed with HyperCard. The SQL-generator was written in C using the Unix-tools Lex and Yacc. The code was then ported to Lightspeed C on the Macintosh.

In the next chapter, we describe the architecture of HYBRIS and give a thorough description of how HYBRIS is used. In Chapter 3, we discuss the graphical query language that is used for information retrieval and problems that occur when designing a graphical database query language. In Chapter 4, the problems of security and performance are discussed. In the last chapter we outline some directions for our future work on HYBRIS.

2. HYBRIS - the Application

The main purpose of HYBRIS today is to allow the same expressiveness as SQL, but to avoid the need for prior knowledge about the underlying database, its structure and its contents. It is also a main purpose to free the user from the lexical and syntactic considerations that makes SQL, let be powerful, a non-trivial query-language even to the experienced user.

2.1. Overview

HYBRIS uses an *information map* to display the structure of the information. The information map is in fact a binary Entity-Relationship data model, but it needs not to correspond one-to-one to the underlying database. On the contrary, we strongly argue that the data model should represent the business that the database supports and not the database structure itself. The binary data model in HYBRIS will hereafter be referred to as the Information Map (IM). HYBRIS runs on a Macintosh to access data on a host computer, such as a VAX. The user defines a query using the graphical interface in HYBRIS. The query is then translated to the database query language SQL. When the user wants data from the database, he simply sends the SQL-query to the DBMS on the host computer where it will be executed. Once the execution of the query has been completed, the user may transfer the result to his personal workstation for manipulation. This is of course optional since the result might be too large to fit into the free space on the personal workstations secondary storage.

The query process is totally transparent to the user. He simply defines a query and sends it to the database. The translation to SQL, the database access, and the communication link is hidden from the user to allow a coherent interface. Thus, the communication link, the operating system and the relational DBMS may vary from one environment to another.

2.2. Navigation

In HYBRIS the user navigates in the IM in order to find out *what* he wants to query about. Several tools have been created to facilitate database queries, but the largest problem - the understanding and definition of information needs - remains unsolved. *You cannot retrieve information unless you are aware of its existence.*

During navigation the user investigates entities, their attributes and relations between entities. He has immediate access to the Meta Database which contains textual descriptions of entities, attributes, data types, value domains and relations. The navigation is not limited to the IM. Once inside the Meta Database, the navigation may continue along the hypertext links that are defined. As an example, if the user is inspecting the description of the entity Contract, he can read that *"a Contract concerns a Service"*. By simply clicking on the word *Service*, the user can read the description of the entity Service.

The ultimate goal of the navigation is to give the user a "feeling" for the information, and to give a good understanding of the structure of the information he will eventually query. This increases the possibility for subsequent queries to be correctly defined.

The IM may be arbitrarily large, or at least so large it will not fit on the screen. Therefore it is necessary to be able to view the IM at different levels of detail. In HYBRIS, two levels of detail are defined:

- the detail level
- the overview level

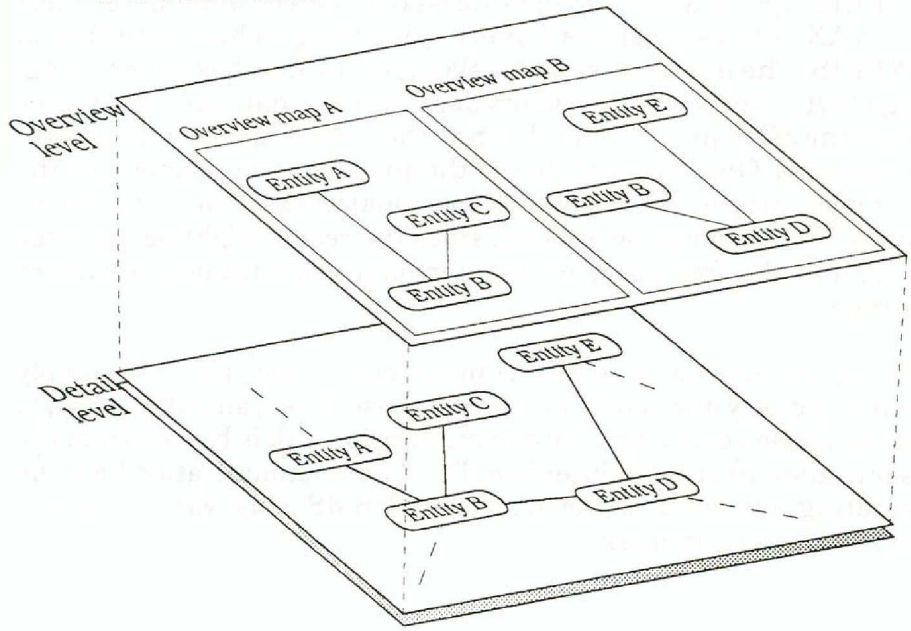

Fig 1. The mapping between the detail and the overview level. The different overview maps consist of - possibly overlapping - subgraphs of the graph representing the entire IM. Note that Entity B can be found on both overview maps.

On the *detail level*, the user focuses on one entity at a time with its outgoing relations and the associated adjacent entities. By clicking on an adjacent entity, the focus will change to that entity. In this way, the user can navigate through the IM just by clicking. An example of the detailed level can be found in Figure 7.

On the *overview level*, a number of *overview maps* are defined. They are groups of entities that are conceptually clustered. An overview map is in fact a sub-graph of the whole IM. Overview maps may overlap and it is desirable that each entity is to be found on at least one overview map, see Figure 1. An example of the overview level can be found in Figure 2.

2.3. Query

Once the user has found out his information needs, he is ready to define a query that will hopefully fulfil them. The query is defined in the same IM where the navigation was performed. Although it seems natural to first navigate and then query, this is just a recommendation. During query definition, the user may want to check some definitions before continuing the query. In HYBRIS, it is possible to do these things in arbitrary order.

A query consists of selecting a number of entities, linking them together and for each entity selecting output attributes and formulate the desired constraints. The process of defining a query in HYBRIS can be viewed as a process of defining an output set. By pointing and clicking and defining constraints the user defines exactly what properties the elements in the output set should have. This has a strong resemblance to *Query By Example* [Zloof75].

To formulate the query: *"Give me the names and addresses of all the customers that are using telex"*, the user has to perform the following steps:

- He has to navigate in the IM in order to understand that *telex* is the name of a Service, and that customers are related to services in two different ways; via the entity Usage and via the entity Contract, see Figure 2. He now has to decide wether he is interested in customers who have *actually used* the telex service or customers who have a *contract* concerning telex. In this case we assume the latter.

- He has to select the entities Customer, Contract and Service, and the connecting relations.

- He has to select the output attributes, i.e *First_name*, *Surname* and *Address* of Customer, see Figure 3.

- He has to formulate a constraint for Service, i.e *Name = 'Telex'*, see Figure 4.

Note that it is possible for the system designer to define *value*

domains for data types. If the attribute *Name* of the entity Service was defined as having the data type *Service_name*, a list of the different service names would be available to the user. This reduces the possibility of faulty searches due to misspelled constants in expressions. The user may either choose from a menu or type by hand as usual. In the latter case, the input will be matched with the value domain to assure correctness.

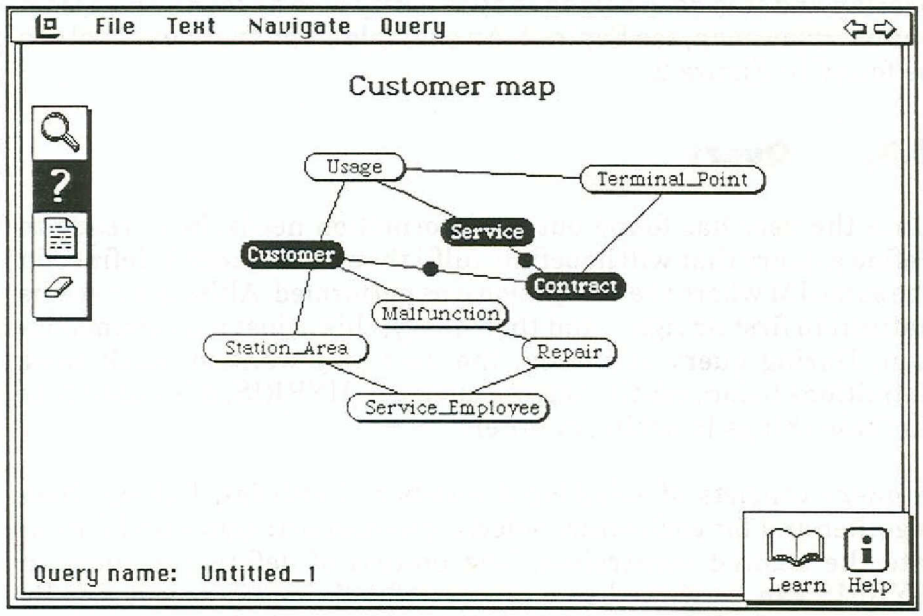

Fig 2. A query definition in the IM. The selected entities are highlighted and the selected relations are marked with a black dot.

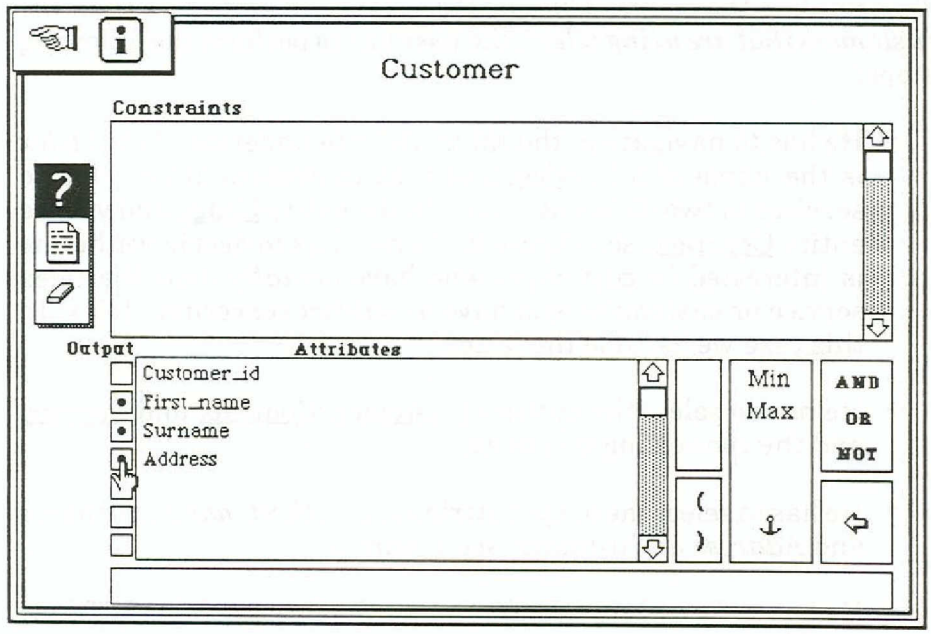

Fig 3. The output attributes for the entity Customer. The selected attributes are marked with a black dot in their respective output box.

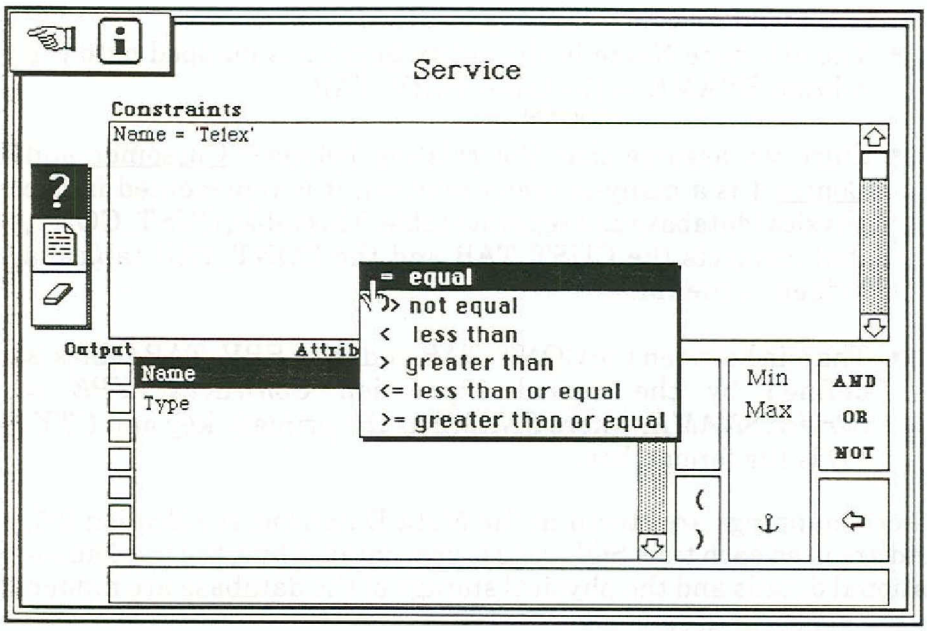

Fig 4. The constraints for the entity Service. It is easy to define more complex constraints for an entity by combining expressions with the boolean operators.

The definition of the query is now complete. Whenever the user thinks it is appropriate, the query can be sent to the database for execution. All queries may be stored for later use, and it is also possible to load queries into the IM for graphical editing.

The SQL-code that is generated from the graphical query is available for inspection by the user, but we chose to make this optional since the SQL-code is rather cryptic to most people. Although, a look at the SQL-code that is generated from the example query might shed some light on the advantages with the HYBRIS approach as well as some of the functionality of the SQL-generator.

```
SELECT Customer.First_name, Customer.Surname,
Customer.Address
FROM CUST_TAB Customer, CONT_TAB Contract, SERV_TAB
Service
WHERE Service.SNAME = 'Telex'
AND EXISTS
   (SELECT *
   FROM CUST_CONT
   WHERE Customer.CUST_ID = CUST_CONT.CUST_ID
   AND CUST_CONT.CONT_ID = Contract.CONT_ID)
AND Contract.CTYPE = Service.SNAME;
```

Fig 5. The SQL-code generated by the SQL-generator for the example query.

There are a number of things worth mentioning about the SQL-code:

- The Customer, the Contract and the Service entities are mapped onto tables in the database with the names CUST_TAB, CONT_TAB and SERV_TAB respectively.

- The attribute Name in the entity Service is mapped onto the column SNAME in the table SERV_TAB.

- Since we assume that the relation between Cutsomer and Contract is a many-to-many relation, it is represented in the physical database as a separate table. This table, CUST_CONT, that connects the CUST_TAB and the CONT_TAB tables is hidden in the IM.

- The join between the CONT_TAB and the SERV_TAB tables is defined by the logical expression *Contract.CTYPE = Service.SNAME*, where *SNAME* is the primary key and *CTYPE* is the foreign key.

These mappings are stored in the Meta Database, see Section 2.5, and are used each time SQL-code is generated. Thus, the implementational details and the physical storage of the database are hidden to the HYBRIS user.

2.4. Result manipulation

When the query has been executed on the host computer, the resulting file can be transferred back to the Macintosh. This file is a flat tabular text file and it is probably desirable to manipulate this file in order to create reports or graphics to be presented.

Since HYBRIS is implemented in HyperCard, any Macintosh application can be started from within HYBRIS. In the current version of HYBRIS there are links to allow the user to open result files with a spreadsheet and a word processor. There is no limit to the number of different applications that can be integrated with HYBRIS.

2.5. The Meta Database

One user is called the HYBRIS SA (Systems Administrator). The HYBRIS SA is responsible for the distribution of new versions of the HYBRIS system to all the personal workstations that run it. New versions of the system is created whenever the Meta Database is updated.

The Meta Database (MDB) is the most fundamental part of HYBRIS. It is supported by the HYBRISSA and a local copy is stored on each machine that runs HYBRIS. The MDB contains information about the following:

- The structure of the IM
  (i.e. entities, attributes, data types, value domains, relations and their respective textual descriptions)

- The structure of the underlying database
  (i.e tables, columns, data types and their physical location)

- The mapping between the IM and the database (i.e. how entities, attributes and relations in the IM correspond to tables and columns in the database)

The MDB is defined in terms of a relational database in order to make it easy to update. The HYBRIS SA is the only user that may update the MDB. However, the local HYBRIS systems do not contain links to the original MDB, just *copies* of it. It is worth mentioning that a local MDB copy is not represented as a relational database, but as a totally integrated part of that specific HYBRIS system (i.e as a HyperCard stack).

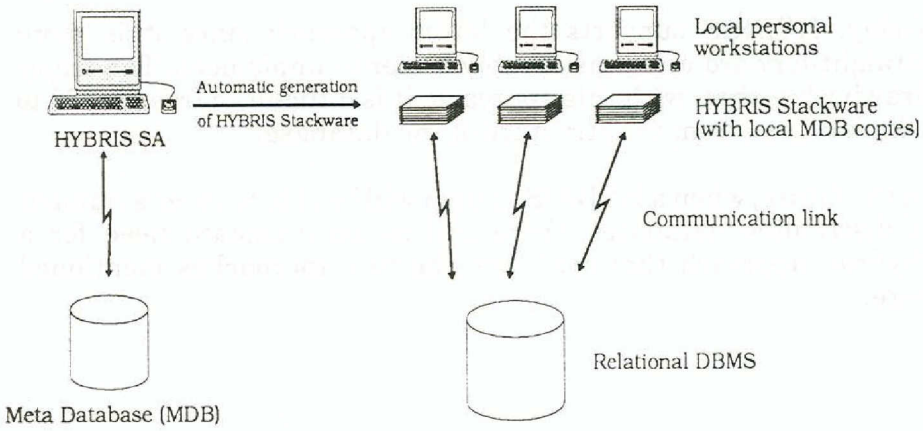

Fig 6. The HYBRIS SA generates new versions of the HYBRIS Stackware that can be distributed to all the machines that run HYBRIS.

There are three major reasons for having a local MDB copy in each HYBRIS system:

- It eliminates the need for a relational DBMS on each personal workstation.
- It supports hypertext navigation in the MDB from within HYBRIS.
- It drastically increases the performance of the MDB access.

The HYBRIS system is automatically generated from the original MDB. This process is invoked by the HYBRIS SA. The generating process converts the contents of the original MDB's relational DB to a HyperCard stack with the appropriate hypertext links.

2.6. How to create an Information Map

The process of defining an Information Map is in fact the process of defining the contents of the MDB. There are two major approaches to perform this process:

- Conceptual-to-physical, i.e begin with a conceptual model over the information and then create the mapping to the physical database. The conceptual model will then be the IM.

- Physical-to-conceptual, i.e begin with the physical database structure and refine it step by step to create the final IM.

Today, HYBRIS supports the latter approach since it is more straight-forward and considerably easier to implement. The major drawback is that, with this approach, it is difficult to free the IM to a high degree from the structure of the database.

In the future, when a HYBRIS system will be able to cover a number of distributed databases, there will be an increased need for a "hybrid" approach that combines the two approaches mentioned here.

3. The graphical query language

The design of a graphical query language is a compromise between "easy-to-learn" and expressiveness. One of our design goals was to avoid implementing merely a graphical dialect of SQL, but still to keep as much expressiveness in the language constructs as possible. Another design goal was to have the query interface totally integrated with the Information Map. An alternative solution would have been to allow the user to express his query interactively by choosing entities from a dictionary and put them on a "workbench". We did not choose the latter solution because we think it is necessary that the user always have a view over the information structure. This will provide him a lot of support when formulating his query.

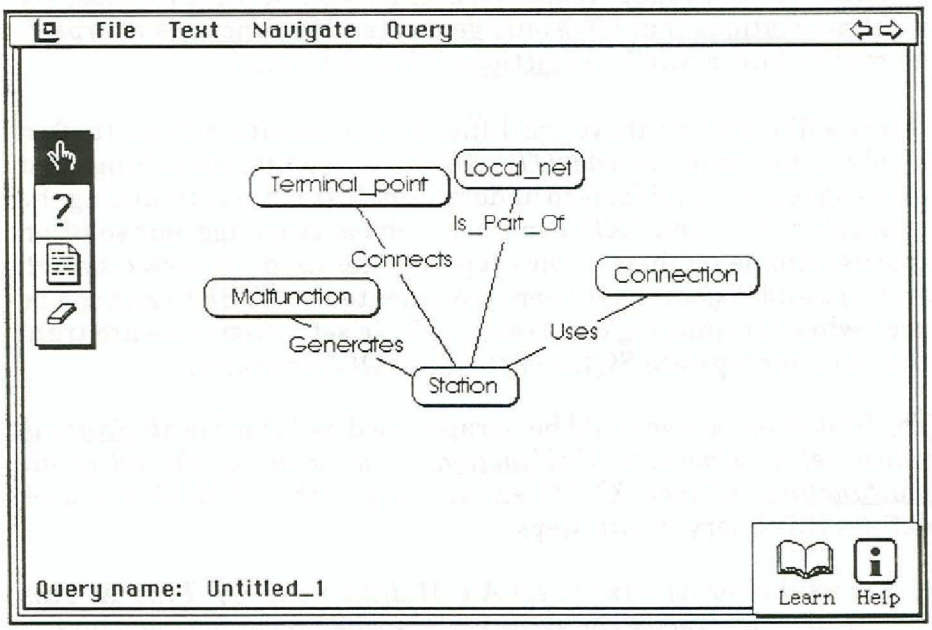

Fig 7 The IM at the detail level. Note that the relations are readable at this level.

3.1. Simple queries

It is rather straightforward to handle flat queries where the user defines simple restrictions on entity sets and possibly links between such sets. Two examples of queries that may be defined in Figure 7 are:

- *"Give me all Stations that are part of Local_net L".*
- *"Give me all Malfunctions that have been generated by Station X".*

The procedure for defining this kind of queries has been described in Section 2.3.

3.2. Complex queries

In real world applications users often have much more complex information needs than can be expressed in a flat query. Two typical examples are:

- *"Give me all Stations that have only generated Malfunctions of type X".*
- *"Give me all Stations that use the same Connections as station A".*

Consider the first example: at first this seems to be a simple query. Why not select Station and Malfunction and set the constraint *type = X* in Malfunction? However, this will give us all Stations that have generated *some* Malfunction of type X. This is not what we seek. We want all Stations that have *only* generated Malfunctions of type X, no matter how many Malfunctions it has generated.

Queries like the two above are difficult to formulate in SQL. It often involves the use of nested *NOT EXIST* statements, something most users have great problems to understand. Instead of introducing the quantifiers *ALL* and *EXIST* in the graphical language, our solution is to use simple set theory. The set predicates *equal*, *superset*, *subset*, *overlaps* and *disjoint* have been implemented in HYBRIS and can be used when formulating constraints. These set constraints are translated to appropriate SQL-code by the SQL-generator.

The first query above could be paraphrased as *"Give me all Stations whose set of generated Malfunctions is a subset of the set of all Malfunctions of type X".* When working with HYBRIS the user defines this query in two steps:

- First he constructs the set *All_Malfunctions_Of_Type_X*. This is done in exactly the same way as when formulating a query. First he selects Malfunctions and adds the constraint *type = X*. Then he chooses **Define set** instead of **Define query** from the menu. This set is now defined and can be used anywhere in HYBRIS.

- Then he selects Station. When he opens Station he will find an attribute called *Malfunctions{}* which represents the set of Malfunctions that are generated by a given Station. The curly brackets are used to indicate that this is a set of entities. Now he chooses **Subset** (actually, a graphical symbol that represents subset) from the pop-up menu. Doing so will show a list of all available sets. Only those sets that are compatible with this attribute (i.e. sets of Malfunctions) are shown in the list. After making his choice from the list he is ready to send the query.

The SQL-query generated by the SQL-generator is shown below:

```
SELECT Station.ID
FROM STAT_TAB Station
WHERE NOT EXISTS
   (SELECT *
   FROM MALFUNC_TAB Malfunction
   WHERE NOT Malfunction.TYPE = 'X'
   AND EXISTS
      (SELECT *
      FROM ERR_REP_TAB
      WHERE Station.ID = ERR_REP_TAB.SNR
      AND ERR_REP_TAB.NAME = Malfunction.NAME));
```

Fig 8 Complex SQL-code generated by the SQL-generator.

Even though the concept of user-defined sets adds a lot of expressiveness to the graphical language, our experience is that most inexperienced computer users have problems understanding the idea of using sets. This is because they are not trained in thinking in set-oriented terms.* Therefore we have introduced two levels in HYBRIS - one for normal users and one for advanced users. At the normal level the user can only formulate flat queries. He cannot define his own sets and he does not see any set-attributes. In the advanced mode sets can be defined and used when formulating queries. Sets can also be defined in terms of other sets making it possible to construct arbitrarily complex sets.

It is our intention that the users should start using HYBRIS at a normal level without having any knowledge of the set concept. Eventually they will find out that they have information needs that cannot be satisfied at this level. That will hopefully give them an understanding of sets as something that can help them formulate their information needs instead of something abstract and mathematical.

* However, with some training, set theory is shown to be an abstraction suitable for non-mathematicians. [Katzeff88] presents an empirical study that compares different user models for efficient use of database query languages - no model, a table model and a set model. The results showed that users that worked with the set model made fewer mistakes formulating queries than the users that worked with the other models.

3.3. Self-references

One very important conceptual modelling construct is the hierarchy. Examples of that are *Part that consists of other Parts* or *Employee that has a manager who is another Employee*. Hierarchies are expressed in a conceptual model by using self-referencing arrows, see figure below.

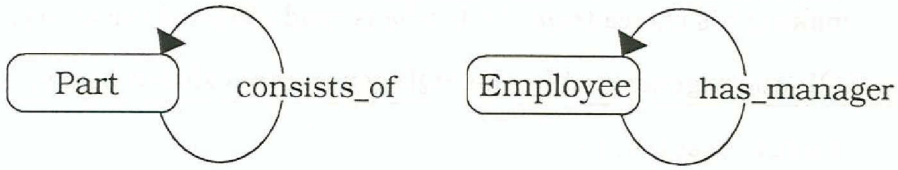

Fig 9 Self-reference is used to represent hierarchies.

Information structures like this arises in all real world applications. Unfortunately it is very difficult to deal with them in a graphical query language if you still want to keep the language simple with few language constructs.

There are two problems:

- How to express queries concerning self-referencing relationships graphically
- Standard SQL does not support recursion

How do the user formulate a query like *"Give me all Employees in the network Department that earn more than their managers"*? Here the concept Employee is used in two different roles: First we are talking about Employee as someone working in the network Department. Then we are talking about Employee as someone being a manager to the first one. A possible solution is that when the user clicks on the relation *has_manager* he will get a duplicate of the entity Employee, as is shown in Figure 10. Then he can formulate constraints separately for the two different roles of Employee.

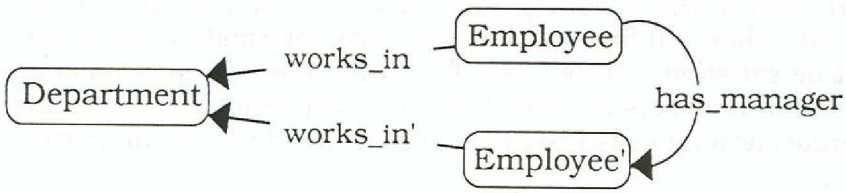

Fig 10 Duplication may be used to represent different roles for an entity. In this case the entity Employee' represents the manager. Note that it is not obvious that the relation ***works_in*** *should be duplicated. It depends upon the reality which the model attempts to describe.*

We feel that this solution is somewhat unsatisfactory because it might give the user the feeling that the information structure has changed in some way. There might also be a problem understanding the different roles of the entity Employee. Our future work will

include the investigation of ways to express recursive relationships in a more natural way.

The fact that standard SQL does not support recursion makes it impossible to formulate queries like *"How many Parts does Part X consist of?"*. In most applications this is solved by using a programming language that has an interface to the database management system. Of course it would be possible to let HYBRIS generate this code, but this would introduce a lot of other problems since different SQL-databases supports different programming languages.

3.4. Subsets and subclasses

One unsolved problem is how to handle subsets of entity sets. At certain times we want to reason about a subset of entities that are related to a certain entity. We might want to know the Stations that have generated more than 50 telex Malfunctions. Today the user only have access to a certain station´s malfunctions and can only formulate constraints concerning the cardinality of a station´s malfunctions, not subsets of that entity set.

An idea which is evaluated at the moment is to incorporate subsets in the Information Map once they have been defined. They can then be treated as any other entity set in the Information Map. This would also make it possible for a user to tailor his own environment.

A powerful modelling technique often used in conceptual modelling is to use abstraction hierarchies such as isa-relations which can help enhance the clarity of the model. This is not supported in the current version of HYBRIS.

4. Security

HYBRIS solves many problems but introduces some new. In all real world database environments security and integrity is of uttermost importance. Since HYBRIS is only used for information retrieval, it does not create any new problems concerning data integrity.

However, it adds some aspects to data security. In a large organization it would be unwise if everybody had access to all available information. There are two ways you can deal with this: at the *conceptual level* or at the *data level*. To handle security at the conceptual level means that users see only the part of the Information Map they are authorized to query about. In this way, users will *not have knowledge* about all existing information.

On the other hand, if security is handled at the data level, the users would see the whole Information Map. If they try to retrieve information they are not authorized to, the system will notify them about this and fail to return a result. Using this approach, a user may understand that there is data - not available to him - that could support his daily work. This might result in a change of access privileges.

The approach depends on the organization and its security policy. Both of them can be implemented in HYBRIS. The data level approach would of course be easier since most organizations already have security systems installed in their DBMS.

5. Performance

Another problem that arises in large databases is response time. With HYBRIS we have made it easy for naive users to send complex queries to database management systems. Queries that may involve very complex search criteria which might download the database manager for hours in the worst case. This has not been a big problem in the past since naive users have been unable to spontaneously formulate complex queries in SQL. To solve this problem it will be necessary to give the user a search cost estimation of a given query based on the query itself and the actual size of the database tables.

6. Future work and research directions

As indicated earlier, our future work will be to further develop HYBRIS towards a tool that fully supports information resource management in large organizations. This will include full support for systems developers to build information maps from a database schema Also, it will be a future goal to make HYBRIS work in a distributed heterogeneous database environment.

Furthermore, we will try to make our query language more "graphical" than today, especially when it comes to formulating constraints. Sometimes the user is not interested in entities with a specific property but wants to ask something more fuzzy like *"Employees that are well paid"* or *"Networks with many Stations"*. We will try to use graphical symbols to express fuzzy quantities. Of course, the system designer has to define what is actually meant by *"many Stations"* or *"well paid Employees"*. However, this information can easily be stored in the MDB.

An obvious extension of the user interface is to connect HYBRIS to a videodisc. Then images, video sequences and sounds could be used to illustrate important concepts in a better way than today. Then HYBRIS could also be used as a tool for teaching new employees how the company is organized and the underlying business ideas.

As was discussed in Chapter 4, several problems arises when users want to express more complex queries in the Information Map. This is an instance of a more general problem, namely how to express predicate logical statements in a graphical model. An important research issue would be to develop a general theory for expressing predicate logical statements graphically.

HYBRIS can be viewed as a hypertext interface that has been put on an existing relational database. Another interesting research topic would be to investigate if it is possible to work in the opposite direction. That is to first define a hypertext system at a conceptual level and then automatically create a relational database schema that supports the system.

References

[Arnborg80] Stefan Arnborg: *"A simple Query Language based on Set Algebra"*, Department Of Numerical Analysis and Computing Science, Royal Institute of Technology, S-10044 Stockholm, Sweden, 1980.

[Chen76] P. P. Chen: *"The Entity-Relationship Approach - Toward a unified View of Data"*, ACM TODS, vol 1, no 1, Mar 1976.

[Czejdo85] Bogdan Czejdo, David W. Embley: *"An algebra for an Entity-Relationship Model and its application to graphical query processing"*, Proc. of the International Conference on Foundations of Data Organization, Kyoto, May 1985.

[Date86] C. J. Date: *"An introduction to Database Systems"*, Fourth edition, Addison Wesley, 1986.

[Estier88] Thibault Estier, Gilles Falquet: *"QFE: A Query Interface using a Hypertext Approach based on Semantic Contexts"*, Centre Universitaire d'Informatique University of Geneva 12, rue du Lac CH-1207 Genève, Switzerland.

[Fogg84] Dennis Fogg: *"Lessons from 'Living in a database' Graphical Query Interface"*, Proc. ACM SIGMOD 84 International Conference on Management of data, Boston 1984.

[Franzén88] Peter Franzén: *"Frågehantering i ett distribuerat databassystem med centraliserad kontroll"*, SISU Internal Paper, only available in swedish.

[Gars88] Pankaj K. Gars: *"Abstractions mechanisms in hypertext"*, Communications of the ACM, vol 31, no 7, July 88.

[Halasz88] Frank G. Halasz: *"Reflections on NoteCards: Seven Issues for the next generation of hypermedia systems"*, Communications of the ACM, vol 31, no 7, July 88.

[Kanga89] Hannu Kangassalo: *"COMIC: A System for conceptual modelling and information construction"*, CASE89 Conference in Kista, Sweden, May 1989

[Katzeff88] Cecilia Katzeff: *"The effect of different conceptual models upon reasoning in a database query writing task"*, Department of Psychology, University of Stockholm, S-106 91, Sweden, 1988.

[Zloof75] M. M. Zloof: *"Query by Example"*, 1975 National Computer Conference, May 1975