REPRINTED FROM

Very Large Data Bases

FIFTH INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES

RIO DE JANEIRO, BRAZIL
OCTOBER 3-5, 1979

Sponsors and Supporters

The following societies and institutions have approved sponsorship or support of the Fifth International Conference on Very Large Data Bases at the time of the publication of the advanced program.

Association for Computing Machinery (ACM)-SIGMOD, SIGBDP, SIGIR
International Federation for Information Processing
IEEE Computer Society
SUCESU-Share Users Association, Brazil
IBM, Brazil

79CH 1406-8C

A Description Language and Pilot-System Executive for Information-Transport Systems

Erik Sandewall
Informatics Laboratory
Linköping University
Linköping, Sweden

Abstract: The paper describes a problem oriented representation for information-flow systems ("information systems"), and an (existing) implementation which interprets the problem descriptions, and which thereby permits demonstration, testing, and modification of new application systems in a prototype stage. The work described here is part of a larger project, whose other parts provide tools for transforming the prototype into a production system.

This research has been supported by the Swedish Board of Technical Development (Dnr 77-4380b).

1. Outline

This paper describes one part of a system for computer-aided development of data processing systems for information transport, i.e. systems which can be characterized as consisting of one or several streams of tasks, where in each stream tasks are initiated in one or a few 'stations' and then forwarded through successive stations which perform other operations. The tasks may represent for example patients that require hospital care, purchase orders, or grant applications. The operations may involve interactions with a terminal operator, accumulation into a file or proceeding along from a file on a certain condition, or printout on an external medium.

For precision, we shall use the term 'information-transport systems' for such applications, although the term 'information system' also often suggests this kind of system. Information-transport systems are different from (the conventional view of) data base systems since they emphasize that information packages are *moving* and *active:* in an organization, they carry a stimulus or an order to a person to perform a certain function, while data base systems are large collections of knowledge which the user can ask questions from, or add knowledge to.

Furthermore, an information-transport system should usually be viewed as a set of cooperating programs, which can be run at different times and/or by different users, and which perform different functions for the common purpose.

The term information transport in this paper *only refers to the logical description* of the application. It may or may not be implemented as transfer of records between several hardware or software systems, i.e. transaction processing. It is perfectly consistent with the approach to keep all records in one data base, with tag fields for indicating the logical location of a record, so that the logical transfer of a record is implemented as an update of the tag field.

In many cases, people in lower echelons in an enterprise (have to) use an information system as an information-transport system, while people on an executive level are more interested in the data base perspective. Thus a good information-transport system should contain files which for its own purposes are resting-places in the data streams, but which also are the sources for queries made from higher levels, and summaries provided for higher levels. But it is important to have tools which can also provide good facilities and flexible systems for the lower echelons.

The *aim of our research program* is to develop a tool for design of information-transport systems (as well as tools for certain other types of data processing systems), with the following characteristic properties in the tool:

- to reduce the amount of manpower required to develop a data processing system for information transport, by providing more sophisticated computer services to the programmer or system designer.
- to reduce the (calendar) time that is required to develop a system
- to increase the flexibility of systems and cut down the time that is required to change the system in response to user's needs

In particular, the tool uses one common system specification as basis for generating several programs which together constitute the information-transport system.

In the approach taken in our research program, each system is to be developed in three steps as follows:

First stage: develop prototype. The prototype version of the intended data processing system is developed in a highly interactive programming system, and then gradually transformed into a production system. The prototype is a model of the information system, which contributes to documentation and to the flexibility of the system, but it is also a running system which can be used for demonstrations and trial use by the final users of the intended system. This makes it possible for users to understand the system and propose changes to it while it is still in a formative stage.

The prototype system is to a large extent expressed in a declarative language, with small segments of application-dependent code (procedures) attached to it. The system for interpreting the prototype description is written in Interlisp (ref. 1). This programming system can best be explained as a data base system for a data base of moderate size, which is able to store programs in its data

base, and which contains an interpreter and a compiler for such programs. It is a workspace-oriented system, like APL, which means that the whole system, including system software, programs, and the data base are stored in one uniform address space (virtual memory).

The tools for developing the prototype are the topic of the present paper. The second and the third stage (described in other papers) are:

Second stage: larger data base. In the second development step, the data for the prototype system are moved out of the Interlisp workspace and to a separate data base system (see figure 1). This allows the system to handle larger volumes of data, and is necessary for field tests of the prototype at least in most cases. The data base system MIMER, developed at Uppsala University Computing Center (ref. 2) is used for the data base part. It is in principle an inverted-file type system, similar to e.g. the 1022 system on DEC-10 computers.

Third stage: generate production system. In the third development stage, the new application system leaves the "dry dock" of the Interlisp environment, and a production system is generated. The major change in this step is to increase performance by eliminating two kinds of overhead in the prototype version, namely:
- the overhead in the Interlisp system, which has been designed as a program development tool and not as a production tool. Its large reportoire of services for the programmer are not needed in the production system
- the overhead involved in interpreting the predominantly declarative information in the prototype description. The third step thus includes a compilation of this declarative information into a program (although still in a machine-independent language) where the same information is implicit.

Thus the development system contains programs (i.e. sets of procedures) for the following services on system prototypes:
- entry of the prototype descriptions
- administration of the meta-database that contains descriptions of prototypes
- interpretation of a prototype, as used in stages one and two
- compilation of a prototype, i.e. performing the operations of the third stage.

The compilation program takes the description of the prototype as input, specializes the program (in the sense described in section 5 below), and generates the corresponding, customized program or (more often) set of programs in a lower-level programming language, namely BCPL (ref. 3).

The essential advantage of this strategy is that it allows the use of a truly problem-oriented representation of the application in the environment of the prototype system. Also, since the Interlisp system is strongly oriented towards manipulation of structured data, including programs and other formal languages, one may fairly easily build a reportoire of tools for analyzing models and for generating pieces of programs from models of the application or the intended DP system.

Since the prototype system contains a description of the desired target system (programs and data base), we refer to it as a *meta data base.* It contains the "data dictionary" of other systems, but the term meta data base emphasizes that the programmer can easily extend it with more information, and he can make use of the meta information in any way he wishes.

In this approach, object data and meta data are handled by two different systems with different characteristics, in our case MIMER and INTERLISP. The first system is oriented towards large volumes of uniform, bulk data, the latter system towards smaller data bases with very rich structure.

This paper describes the prototype system used for stage one, since most of the software for the other stages has been described before (ref. 2, 3, 10, 11). The software that is described here for the first stage has been developed and is in experimental use.

2. Related work

This approach represents a continuation and synthesis of a number of different ideas and systems, in particular:

The use of a meta-database that contains a description of an object data base is in a certain sense characteristic of all data base systems. However, in many systems the meta-database is only intended to be used by the system itself. Works by Roussopoulos and Mylopoulos (ref 4) describes the use of an externally available meta-database, which in their case is a semantic network. The query-language compiler of Risch (ref. 5) and the SRI natural language query facility (Hendrix et al., ref. 6) are other examples.

The use of pilot systems as a way for the final user to influence system design has been described by Berild and Nachmens (ref. 7). The CS4 system which they propose as a tool for prototype systems has some facilities which are similar to those of Interlisp, but we have also made heavy use of Lisp's facilities for analysis and generation of programs.

Strong tools for the programmer in a system for interactive program development, has been pioneered by Teitelman (ref. 8).

The approach in the REMORA project (ref. 12) is similar to ours but does not seem to be in a concrete implementation phase, and does not give any specific advise about what we call phase two and three.

3. A concrete example

When our tools are used, the first stage in the system development process is to build the prototype and to use it for test runs. We shall describe how the prototype is developed, and to make the discussion concrete we will use a specific case from the Linköping University Hospital, namely the information flow between the patient wards and the chemical laboratory. This section describes the example.

This application has the following characteristics in the current, non-computerized organization. Requests for laboratory analyses are prescribed by a doctor, and notes

are taken by the head nurse. She completes forms for the same requests, and forwards them to the laboratory. Sometimes the nurse also takes the test samples; in other cases the laboratory receives the requests, prepares lists of which test sample(s) to take from which patient, and laboratory personnel collect the samples.

In the laboratory, test samples are processed according to the instructions on the request forms. Usually samples requiring the same analyses are grouped into batches that are processed together. A result list for each batch is produced, and the results are transferred to the analysis request forms, which are then returned to the ward.

The head nurse sorts the returned lab results by patients. Once a day (usually), a doctor inspects the results and prescribes further action. The nurse then transfers the results to a table containing the analysis results for the patient over a period of time, with one column per day, one line per analysis.

This is the main structure of the application. It contains a number of fine points, and the reader can probably guess many of them: the ward nurse keeps records of analysis requests which have been sent but not yet been returned, and takes action if they take too long time. This time is widely different for different types of analyses. The laboratory keeps back-up copies of all results, and also accumulates charge information. There is often a transformation from the raw data produced by the analysis itself, to the data desired by the doctor, for example the transformation from concentration and volume to amount for a certain substance.

In summary, this application can be characterized as an information transportation system, where packets of information are generated at one time, obtain more information attached to them as they circulate through the system, and also sometimes stay in information buffers for a duration of time before they proceed. The present manual system requires these data to be re-arranged several times (for example from being sorted by patient, to being sorted by analysis, and later back again). In the present organization, all information is carried on paper, and the reorganization is done manually. It is an obvious candidate for computer support, and we believe it is typical of many DP applications, in medicine and elsewhere.

Let us now describe how this application is approached using our current system.

4. Building a prototype for the application

4.1 The specification of the intended system. The intended information-transport system is initially described by the user-programmer in four ways. (In Cobol terminology, the description consists of four 'divisions'):

- specification of the *stations* and types of stations, that are used in the application. Technically, a station is an intended program that serves one user or small group of users. In the present application, there would be one station type for 'wards' (with one instance for each ward), one station type for analysis works in the lab, one station for archive, and so forth.

  The specification of stations only introduces their names and a few simple facts about them. Additional information about each station is provided implicitly by the other three 'divisions':

- specification of the data types that are to be used in the application. This is a fairly conventional description of records consisting of terms, nested sub-sequences of sub-records, etc. In addition, the data type description also specifies the HISTORY of each term; this will be explained below.

- specification of the *layout(s)* that is(are) to be used for displaying each data type on screen and/or paper. This specification is entered using an interactive tool which enables the user to build up and edit the desired layout on the CRT screen.

- specification of the *information transport lines* that are used in the application. This concept is crucial for our system, and will be abbreviated *IT line*. An IT line is a standard path (through stations) along which information packets are sent for processing. In the sample application, there is one major IT line from wards to laboratory and back to the originating ward. Other lines are for complaints for lost analyses, and for checkin/checkout of patients (a new information packet is set up when the patient checks in, and is transferred to archive when he or she checks out).

  The specification of an IT line indicates which record structure(s) are used along the line, and which operations are performed as packets move along the line.

  To insure that the model is viable, it is important to have a sufficiently powerful set of such operations. Besides operations for data entry, output, waiting in queues, and accumulation and data extraction in files, we also need operations for transformations on record structures and operations where several lines are intermeshed, e.g. 'wait for mate' operations.

4.2 The transformation to the executable model. The four parts of the system specification have to be reorganized somewhat in order to provide a model which can be executed as a prototype. When it is executed (= interpreted), the model must specify for each station which operations can be performed in the station. For example, in a ward there may be operations for "send laboratory request", "receive results from laboratory", "check in patient", "check out patient", and others. Each operation in a ward may be realized e.g. as a separate program, or as a set of parameters for an existing, more general program.

Each operation in a station is usually an aggregate of several consecutive operations along an IT line. For example, if an IT line specifies that a lab request is to be entered interactively, accumulated to a local log file, and then sent to the lab, these three IT line operations together constitute one operation in the station.

For each operation in a station, it must be specified:
- where input information packets for the operation are to be fetched from, and where output information packets are to be sent

- which record type is used
- which IT line operations are to be performed
- which layout (or other interaction format) is to be used in the operations. (Usually each station operation consists of exactly one interactive operation, and optionally one or more non-interactive operations)

Thus the transformation from the user's system description to an executable model cuts up the IT line descriptions into segments, each consisting of one or a few IT line operations, and adds each segment to the appropriate station or station type.

In principle, each record type is associated with a form description, which may be used for all interactions along a line, i.e. in several operations in different stations. However, the layout (= form) must be used somewhat differently in the different steps, since typically a few fields in the record are completed when the record is initiated, and other fields are assigned contents in subsequent steps.

Each layout therefore exists in several variants, which differ with respect to which fields are to be protected, which fields are to be entered from the user, which fields are to be assigned values by reference to existing files in the data base, etc. The most convenient way to enter these variants seems to be the following: the record declaration specifies for each term where along the IT line that term is to be assigned a value, and in what way the value is obtained. This is what we called term history information above. The record declarations, the IT line descriptions, and the basic layout description (consisting of only the X-Y coordinates for each field on the screen as well as auxiliary text there) may then be used to generate the variants of the form that are to be used at the various interaction points.

In summary, the executable model is hierarchical rather than parallell: it consists of stations, each of which contains a number of operations, each of which consists of a number of IT line operations, some of which are associated with layouts, which consist of term descriptions, which contain variant information derived from the record declaration. The transformation from user-defined model to executable model is a transformation from a parallell to a hierarchical structure, and is outlined by figure 2.

4.3 The chemistry-lab example worked out. Let us work out the chemistry-lab example of section 3 in more detail, to illustrate the previous section. We cover only the IT lines for checkin/checkout and lab requests (not complaints), and also make some other simplifications to save space here.

We need two major record structures: PATIENT and LABREQUEST. The first record type contains fields such as:
- patient's name (PATNAME)
- patient's identification (such as social security number) (PATNR)
- date of checkin
- date of checkout
- bed number
- previous lab results (which for the present purpose may be viewed simply as a sequence of records of type LABREQUEST) (LABRES)

Mnemonics for term names are given in those cases when they will be used in the subsequent example. Other information such as "ailment" are of course essential for the application as such but not for the present example.

The second record type, LABREQUEST, contains fields such as:
- patients name (PATNAME)
- patient's identification (PATNR)
- ward that the patient is in (WARD)
- desired analysis (ANALYSIS)
- identification number of the sample (SAMPID)
- date of sample
- result of analysis

Since several analyses are often ordered at the same time for the same patient, it is convenient to have one additional record type LABORDER which is like LABREQUEST except that instead of the term ANALYSIS there is a term ANLIST whose value is a list (sub-sequence) of desired analyses for this patient.

The layouts for these record types offer no surprises. (For the real application, many more fields are needed, and the screen becomes rather jammed).

The two IT lines are described graphically in figure 3 and in equivalent formal notation in figure 4. Appendix 1 *(not included in proceedings; aviavailable from author)* contains the reportoire of operations that we are presently using; the set is of course continously revised. Figure 5 describes the structure of stations, with one station for each ward, one for each analysis workplace in the lab, one each for receiving and delivery in the lab, and one for archive. The arrows in figure 5 specify the normal paths of information flow.

Figures 3-4 should be read as follows. The IT line for patient checkin/checkout starts at CI, where new information packets (records) are entered each time a patient checks in. These records are accumulated to the register of current patients (in the ward), CPAT. At CO, records in the file CPAT are released when patients check out, and are then accumulated to the archive file ARCH.

The other IT line, for ordering lab analyses, starts at OLA where the patient number and list of desired analyses are entered, as a record of type LABORDER. The patient's name (and other information about the patient) is fetched from the current patient register CPAT dynamically during data entry. The record is expanded into several records, one for each analysis, which implies a transformation to records of type LABREQUEST. Each such record is accumulated to the register of pending requests, PREQ. It is also dispatched to the appropriate analysis station in the lab, on the basis of the desired analysis (ANALYSIS field) and a procedure which knows which lab station performs which analysis.

In the appropriate station within the lab (in position RA), the results of the analysis are entered. It is then forwarded to the exit station of the laboratory, where results are accumulated to the backup file BU, and also dispatched to the originating ward on the basis of the contents of the term WARD. Back there, each record goes two ways: to an

interaction operation DI where the doctor is enabled to inspect recent results, and to being accumulated into the patient register CPAT as a sub-record of the main record for the patient.

The reader will have noticed that the formal description in figure 4 is more specific than the graphic description in figure 3, particularly with respect to which data fields are affected in the various steps. Of course, one may informally add text in the figure, or in an appendix to the figure, if one does not wish to proceed directly to the formal representation.

The present, implemented system makes it possible to input the application description in the user-oriented notation, and to generate and execute the prototype model fully automatically. The convenience of this system makes it practical to tailor the information transport system very closely to user's needs, for example to design different stations differently to conform to specific local needs.

4.4 Application dependent meta-data and procedures. In many applications, it is useful to have application dependent meta-data. This observation was previously made by Hägglund and Holmgren in another project in our group. Examples of such meta-data in the example of this paper are:

- for each analysis, one may specify the range of normal results. It is sometimes desired to provide this information on the result form, or to provide the result only if it is outside the range.
- for each analysis, there is also an expectancy for how long it should take at the lab. This information is used by the head nurse at the ward to determine when to complain about a missing result.
- there are groups of analyses which may be obtained using the same sample. This information is used when more than one of the analyses is prescribed by the doctor.

Our meta database system makes it easy to enter and administrate such information, and to use it for table-driving the application dependent procedures. This facilitates modification of the pilot system to conform to changing specifications.

Notice that the application dependent meta-data are strongly intermeshed with the general-purpose meta-data. For example, each analysis usually appears as one term in some record for analysis results, and the different analysis stations in the laboratory are also stations from the point of view of the information system.

Similarly, it is sometimes necessary to write specific pieces of program for an application, and embed them in the model so that they are executed as steps along an IT line. Also, one often wishes to associate pieces of program with fields in a form, for example for automatically computing the contents of the field, for fetching it from another file, for checking correct input data, or for performing a side-effect when a certain term value is entered in a record. This can easily be done in our system, and is available as a consequence of LISP's almost unique facility for *data-driven programs* (see ref. 9). But since the general framework is provided by the generel-purpose system, such application-specific programming may be restricted to those parts that are really specific to the application. Often it comes out as only a few lines of code.

4.5 The executive for the information-flow model. In order to test and demonstrate the prototype, one uses an executive which may be viewed as an interactive simulation system for the information-flow model. The prototype may be run either *resident* or *distributed*. In resident mode, one single Interlisp workspace is used for all work stations, and the system devotes attention to one of them at a time. It may run with one terminal for all work stations (this is the usual mode in early checkout by the programmer), or with one terminal for each of several work-stations. In distributed mode, several copies of the workspace are set up, and are specialized to service one station each, with one terminal for each station. The distributed mode of use is preferred for demonstrations to final users.

For early stages of checkout, the executive is directed by user commands such as "execute operation O" (for example the operation where the nurse creates lab request records and sends them to the lab) or "what are the contents of channel C?" If the required operation is to be performed manually, the interaction is performed over the same terminal; if it is an automatic operation it just runs, and prints out a message when it has finished.

In later stages of checkout while still in resident mode, a higherlevel executive may be used which essentially calls different operations according to the obvious strategy of first clearing channels by doing "receive"-type operations, and then performing operations which create new transactions.

For distributed execution of the prototype, the prototype system is able to create a number of copies of itself, where each copy is customized at least to the extent that the copy knows which operations it is to perform, and which terminal it is to run on. It is also possible, but usually not useful, to trim each copy so that it only contains the meta-database information that it really needs.

The presently implemented system only allows us to execute the different programs in the generated system on one single computer, but the possibility of distributing 'stations' in the sense of our system over several computers is an obvious possibility.

4.6 Status of the prototype system. In summary, the prototype system enables the programmer or system designer to enter his system description in a notation which is predominantly declarative, and which is stored in the metadatabase. This description is interpreted by the standard programs in our system, such as the data editor and the station/operation executive. In addition, most applications require a number of small application-dependent procedures to be written (or retrieved from the library) and attached to entities in this description.

This prototype system is entirely embedded in Interlisp, meaning that standard programs have been written in this language, that the system description is stored in the Interlisp data base, and that the attached procedures must also be written in Lisp.

The standard models in the system (i.e. for forms, information flow, etc.) are not final. The characteristics of the programming sytem makes it easy to modify these models as we go along, and we are making use of this possibility. The significant result of this research is by no means the models, but the system which allows us to maintain a spectrum of meta-information, ranging from the standard patterns to the very application specific.

All parts of the software for prototype support as described here have been implemented and are running on our computer, a system DEC-20. The implementation language is Interlisp, with certain low-level routines in assembler (Macro-10 and LAP). The present status can be characterized as experimental use; the system is not yet stable enough for export.

5. Downloading the prototype into a production system

In the second and third step of the system development, the performance of the prototype is improved by transferring it out of the Interlisp environment, by specializing programs, and by knitting parts of programs closer together by eliminating data-driven procedure calls. The details of how to do this are being reported elsewhere, and we will here only describe this process in short, for completeness, and report on the status of that work.

The second stage is to enable the prototype to communicate with larger volumes of data, administrated by the MIMER system for bulk data. This requires two things:

the Interlisp system must be able to co-exist with and communicate with the MIMER system. This has been done at Uppsala Computing Center for the IBM 370-based systems, but we are running on a DEC-20 system for which MIMER has not yet been implemented. That work is in progress and is well understood.

the form-oriented data editor must be able to access data from MIMER. This has been prepared for in the following way: the data editor assumes that each file is associated with an *access method* and each access method is characterized by a number of access procedures. All data access in the data editor goes through the access method's procedures for the current sequence. Thus in order to communicate with MIMER, a new set of access procedures has to be written. However, the set of procedures that the access method manager requires, is similar in structure to the procedures that are implemented in MIMER, so again we do not foresee any substantial problems.

The third step is to transfer the whole program out of the Interlisp environment. The strategy for this is to write a translator from Interlisp (with certain restrictions on the language) to the low-level, semi-machine-independent language BCPL, as well as the necessary run-time system in BCPL for operating on LISP's characteristic data structures. This software is being completed; it has so far been used successfully on a subset of the data editor.

The prototype system contains general-purpose programs such as the data editor and the executive, which operate on parametric data. One may use the translator in a straight-forward fashion to translate the general programs to BCPL, but in many cases one would prefer to generate a specialized program which has been customized to the chosen parameters. This process is called *partial evaluation*, and has been the subject of a number of studies (see e.g. ref. 10 and the survey of previous work therein).

In simple cases, one can identify certain procedure calls in the general program where some of the procedure's arguments are parameters, and thus can be considered as fixed within each application. One may then use conventional macro techniques to expand the expression to one which is specialized for the constant parameters. We are doing this already when our system is compiled in the Lisp compiler, and the required macro facility is included in the Lisp-to-BCPL translator.

A more systematic specialization requires a full scan of the general program in order to perform constant propagation and various algebraic operations on the program. A system called REDFUN for performing this task has been developed within our group by Anders Haraldsson (ref. 11), and using this system is part of the future plans for the project. We first want to see how far we can get using the macro techniques.

In certain cases, inner loops of parts of the system may be distributed out to local processing capacity in (or close to) the terminal, for example for the core of the data editor. In such cases partial evaluation may not be of interest at all, since the program in the terminal may run better interpretively.

6. Conclusions

The approach and the system described here offer a powerful method of supporting the development of data processing systems, and to meet the objectives stated at the beginning, i.e. to cut manpower costs for programming, cut system development time, and increase the flexibility of the developed system. In looking for possible disadvantages of the approach, two aspects are of particular interest:

- Run time performance. The prototype is developed as a declarative structure, which means that the executive ("interpreter") for the prototype is a very strongly parametrized program. The classical disadvantage of such programs is low performance. We offer a set of techniques for dealing with that, namely:
  - = cross compilation
  - = macro techniques
  - = partial evaluation

  all of which rely on Lisp's almost unique facilities for program manipulation. Although these techniques have been successful in other contexts, they have not yet been tried for any problem where the programs are as big as here.

- Complexity. The programmer or system developer who specifies each new application by building the right

declarative structure, will need a fairly detailed knowledge of the development system. Training time may be a problem. Also, the development system must be very well engineered before it is put in regular use, otherwise the system developer will end up fighting the development system instead of (as now) fighting his own programs.

The obvious approach for dealing with these problems is to make the development system interactive and self-instructing, and to develop it gradually while doing a number of applications with it. This work is in progress.

In summary, the software for stage one is in experimental use; the software for the second and third stage exists and has been reported before; and integration of the full system has not yet been done. It is therefore too early to say whether the possible problems with the approach have been solved, namely performance and complexity, but we have reasonable expactations to be able to handle them both.

7. Future plans

The immediate plans of the project are to use the system for a number of applications, to gain additional experience with the selected approach and the existing development tools.

Medium-range plans emphasize:

- the generation of software for distributed systems from the prototype
- more systematic methods for partial evaluation, in particular using the REDFUN system
- use of the prototype to document and control the production use of the generated system, i.e. as a 'job control system' that supersedes current 'job control languages'.

Longer-range plans include interfacing the system to a model of the application environment as such (including other aspects than the informations processing aspects).

Acknowledgements

This work has been made possible by cooperation with Doctor Werner Schneider, Professor Bo Sörbo, Professor Ove Wigertz, and Doctor Lennart Tegler. Within our own research group, Peter Fritzson, Erland Jungert, Gunilla Lönnemark, Dan Strömberg, and Katarina Sunnerud have worked on related projects without which the present work would not have been possible. Jan Komorowski, and Henrik Sörensen have recently joined the effort reported here.

References

1. Warren Teitelman: *INTERLISP reference manual.* Xerox Palo Alto Research Center; Palo Alto, Calif., 1974

2. A. Berghem; Anders Haglund, Sven G. Johansson, Åke Persson: *A partially inverted database system with a relational approach, MIMER (earlier RAPID).* Uppsala University Data Center (UDAC), July 1977

3. Martin Richards: *BCPL reference manual.* Technical memorandum 69/1, University of Cambridge, Computing Laboratory, 1969

4. Nicholas Roussopoulos and John Mylopoulos: *Using Semantic Networks for Data Base Management.* Proc. 1st International Conf on Very Large Data Bases; 1975, pp. 144-172

5. Tore Risch: *Compilation of multiple file queries in a meta-database system.* Dissertation, Linköping University, Sweden, 1978

6. Gary G. Hendrix et al.: *Developing a Natural Language Interface to Complex Data.* ACM Transactions on Data Base Systems, Vol. 3, Nr. 2 (June, 1978)

7. Stig Berild and Sam Nachmens: *CS4 - A Tool for Database Design by Infological Simulation.* Presented at 3rd International Conference on Very Large Data Bases, 1977. (To appear in ACM Transactions on Database Systems)

8. Warren Teitelman: *Toward a programming laboratory.* Proc First International Joint Conf Artificial Intelligence, 1969

9. Erik Sandewall: *Programming in an interactive environment: the "LISP" experience.* Computing Surveys, Vol. 10, No. 1 (March 1978)

10. Lennart Beckman, Anders Haraldson, Östen Oskarsson, Erik Sandewall: *A partial evaluator, and its use as a programming tool.* Artifical Intelligence Journal 7 (1976), pp. 319-357

11. Anders Haraldson: *A partial evaluator, and its use for compiling iterative statements in LISP.* Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, 1978, pp. 195-202

12. O. Foucault and C. Rolland: *Concepts for Design of an Information System Conceptual Schema and its Utilization in the Remora project.* Proceedings of the 4th International Conference on Very Large Data Bases, 1978.

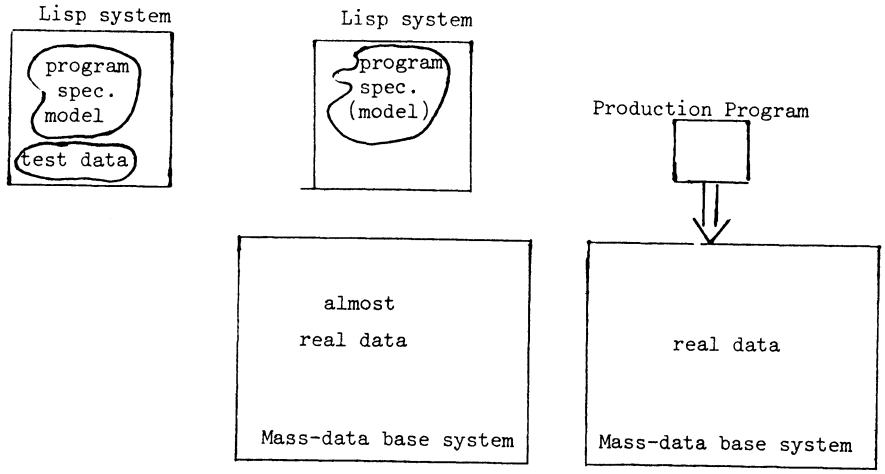

Figure 1: The three stages of system development

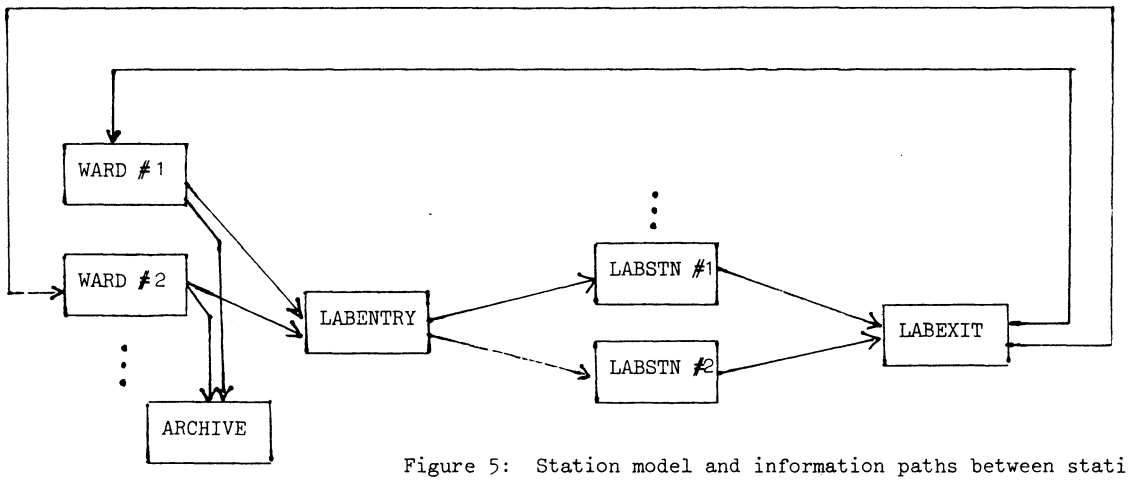

Figure 5: Station model and information paths between stations

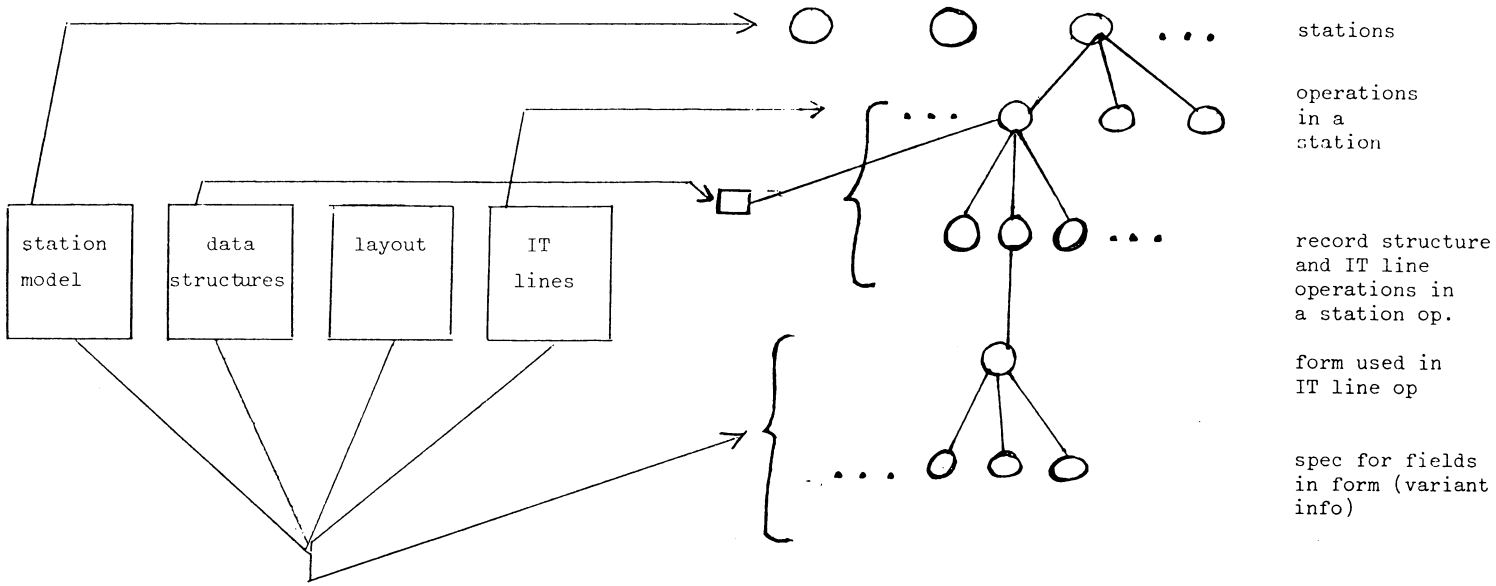

Figure 2: Transformations from user-oriented description to executable model of IT system

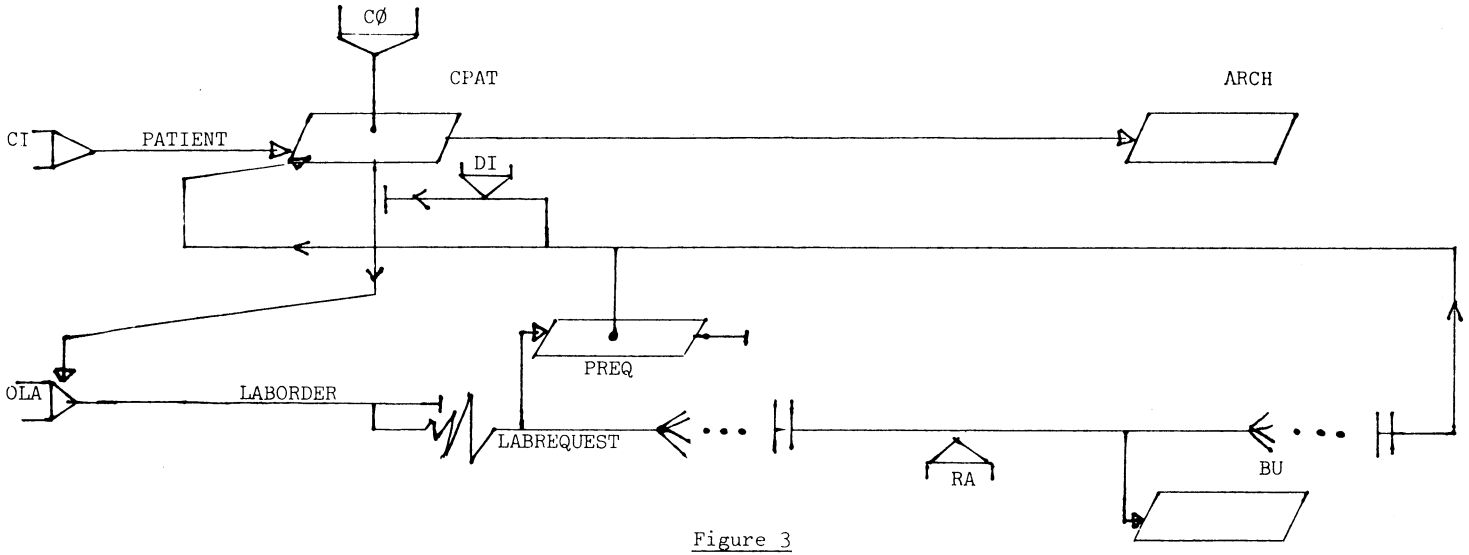

Figure 3

Description of chemistry-lab application's information flow
in IFL notation

IT line for patient checkin/checkout:

record structure: PATIENT

```
operations:     (STATIONTYPE WARDS)
                (ENTRY CI)
                (ACCUM CPAT)
                (RELEASE CPAT CO)
                (STATION ARCHIVE)
                (ACCUM ARCH)
```

IT line for orders of lab analyses:

initial record structure: LABORDER

```
operations:     (STATIONTYPE WARDS)
                (ENTRY OLA)
                (EXPAND
                   (LABREQUEST ANLIST (...) (...))
                   (BRANCH NIL NIL
                      (ACCUM PREQ)
                      (RELEASEBY (READY PREQ (SAMPlD SAMPID))) )
                   (STATION LABENTRY)
                   (DISPATCH (STATIONFOR (GV* ANALYSIS)) RQSTPORT)) )
                (STOP)
                (STATIONTYPE LABSTN)
                (CHANNEL RQSTPORT)
                (MODIF RA)
                (STATION LABEXIT)
                (BRANCH NIL NIL (ACCUM BU))
                (DISPATCH (GV* WARD) FROMLAB)
                (STATIONTYPE WARDS)
                (CHANNEL FROMLAB)
                (AUTORELEASE READY)
                (BRANCH NIL NIL (MODIF DI))
                (ACCUMSUB (CPAT (PATNR PATNR) LABRES))
```

Figure 4.