A REXX-based Stock Exchange Real-time Client/Server Environment for Research, Educational and Public Relations Purposes: Implementation and Usage Issues

Martin P. Misseyer
Lou W. M. Guse
Armoud W. Morsink
Vrije Universiteit Amsterdam

Pages 292-322

A REXX-based Stock Exchange Real-time Client/Server Environment for Research, Educational and Public Relations Purposes: Implementation and Usage issues

Martin P. Misseyer, Lou W.M. Güse, Arnoud W. Morsink

Vrije Universiteit
Faculty of Economic Sciences, Business Administration and Econometrics
Department of Information Systems

Amsterdam, April 1995

Abstract

For many years now the Faculty of Economic Sciences, Business Administration and Economics of the Vrije Universiteit in Amsterdam propagates to impart students of economics scientific 'real market' skills and experience in, for example, portfolio management. Aside from the risk neither the faculty nor most of the students have sufficient means to practice in portfolio management. In the early 1980s the idea evolved at the faculty to develop and use a portfolio management simulation. The Amsterdam Stock Exchange (ASE) granted the faculty in 1983 a free of charge data link with its administrative clearing information system. The data link provided the faculty with real-time data including stocks traded (time, price and volume), exchange news, stock splits and many more. At the time the technology used was relatively simple: the data link consisted of a 1200 bps modem connection using the X-modem protocol to receive data. The received data were put in flat files and were in turn read by a small, written in C, in-house developed portfolio management simulation system named TRANSPAS.

In 1990s it became inevitable for ASE to acquire a modern trading system as more and more trade leaked to more sophisticated foreign exchanges (London, Paris, and Frankfurt). ASE developed a new trade system, entailing major operational and technical changes. In September 1994 a new, fully automated, trade system became operational for complete administration of all ASE transactions (to be) made. Nowadays at the ASE tens of millions of stocks, bonds, futures change hands every day.
The major changes were twofold. First the trade itself was re-engineered by ASE, however this is not discussed in this paper. Secondly, a tremendous change at the technology level was unavoidable. ASE moved from simple analogue asynschronous communication to X.25-based digital synchronous communication (SDLC), anticipating on the need for both much more capacity as data streams increase (re-engineering!) and for full reliable data-link monitoring. Therefore, the faculty was faced with the fact that TRANSPAS as well as its data link became outdated, and implied that the whole system had to be rebuilt.

In 1993 the first author of this paper headed the project team, a composition of colleagues (researchers, graduates, automation staff) and enthusiastic undergraduates, to rebuild TRANSPAS and its ASE data link. The first step the project team took, before setting up several projects, was re-examining the faculty goal. After thorough research, two goals were aimed at. First, the basis of the integrated environment should be based on state-of-the-art relational database technology (the database project was named after the database to be developed: BeursBase) in which all the raw ASE data received would be stored real-time/on-line for a long period of time, preferably three years or more. Secondly, the project team distinguished three major application areas for the database: research, education and public relations. For each of these areas a specific range of application programs

must be developed. Obviously, one of the core application programs is the portfolio management simulation system TRANSPAS, which was renamed into: VUPOS. It became clear that VUPOS could be used in all three areas defined.

Concurrently with the ASE new trade system, the faculty-built REXX-based Client/Server (C/S) became operational. Though already halfway implementation the faculty was informed by the Computer Services Center about its new strategy: the IBM S/390 host facility, the VM Server in the REXX C/S environment (and the basis for both BeursBase and several applications including VUPOS) would be stopped at the end of 1995. It would be replaced by a AIX cluster of 4 very large 590 RS/6000 mid 1994. When the project team was acquainted with the news immediate action was taken. In contrast with the SQL/DS version installed DB2/6000 (AIX) supports full C/S. For several reasons the project team decided to develop in parallel a second, REXX-based C/S environment using the AIX host as Server. This 'new' C/S environment - referred to as the AIX C/S environment - went in operation last January 1995. As the 'old' VM C/S environment was primarily based on REXX, porting the applications to a 'new' AIX C/S environment was relatively simple. Both VM C/S and AIX C/S environments are now fully operational and perform as was planned for, having many similar as well as distinguishing characteristics. The portfolio management Simulation (VUPOS) for the VM C/S environment is written in CSP, and is already used by hundreds of students. Since the AIX C/S does support full C/S the project team was able to develop VUPOS in VX-REXX. Recently the development of this version of VUPOS has entered its final stage. Other applications in VX-REXX, APL/2, VisualGen and VisualAge under construction, range from an import/export facilities to fundamental and technical analysis, and are primarily developed for the AIX C/S environment. The first quarter of 1995 will be used for large scale tests of the system.

This paper presents the design, development, and implementation of these C/S systems from both developer and user views and from both technical and non-technical points of view.

1 General introduction

1.1 Students, portfolio management and TRANSPAS

For many years now the Faculty of Economic Sciences, Business Administration and Economics (FEWEC) of the Vrije Universiteit in Amsterdam (The Netherlands) propagates to impart students of economics scientific 'real market' skills and experience in, for example, portfolio management. Aside from the risk neither FEWEC nor most of the students have sufficient means to practice in portfolio management. In the early 1980s the idea evolved at FEWEC to develop and use a portfolio management simulation. The Amsterdam Stock Exchange (ASE) granted FEWEC in 1981 a free of charge data link with its administrative clearing information system. The data link provided FEWEC with real-time data including stocks traded (time, price and volume), exchange news, stock splits and many more. At the time the technology used was relatively simple: the data link consisted of a 1200 bps modem connection using the X-modem protocol to receive data. The received data were put in flat files and were in turn read by a small, written in C, in-house developed portfolio management simulation system named TRANSPAS.

1.2 Developments at the Amsterdam Stock Exchange

Early in the 90s it became inevitable for ASE to acquire a modern trading system as more and more trade leaked to more sophisticated foreign exchanges (London, Paris, and Frankfurt). ASE developed a new trade system, entailing major operational and technical changes. In September 1994 a new, fully automated, trade system became operational for complete administration of all ASE transactions (to be) made. Nowadays at the ASE tens of millions of stocks, bonds, futures change hands every day.

The major changes were twofold. First the trade itself was re-engineered by ASE, however this is not discussed in this paper. Secondly, a tremendous change at the technology level was unavoidable. ASE moved from simple analogue asynschronous communication to X.25-based digital synchronous communication (SDLC), anticipating on the need for both much more capacity as data streams increase (re-engineering!) and for full reliable data link monitoring. Therefore, FEWEC was faced with the fact that TRANSPAS as well as its data link became outdated, and implied that the whole system had to be rebuilt.

1.3 Rebuilding TRANSPAS: a project plan

In 1993 the first author of this paper headed the project team, a composition of colleagues (researchers, graduates, automation staff) and enthusiastic undergraduates, to rebuild TRANSPAS and its ASE data link. The first step the project team took, before setting up several projects, was re-examining FEWECs goal. After thorough research, two goals were aimed at. First, the basis of the integrated environment should be based on state-of-the-art relational database technology (the database project was named after the database to be developed: BeursBase) in which all the raw ASE data received would be stored real-time/on-line for a long period of time, preferably three years or more. Secondly, the project team distinguished three major application areas for the database: research, education and public relations. For each of these areas a specific range of application programs must be developed. Obviously, one of the core application programs is the portfolio management simulation system TRANSPAS, which was renamed into Vrije Universiteit POrtfolio Simulation, in short VUPOS. It became clear that VUPOS could be used in all three areas defined.

The VUPOS/BeursBase project defined the following phases:
1° Provide a strategy plan, including a re-examination of FEWECs goal(s);
2° Evaluate possible solutions, alternatives for the new system;
3° Design a new organizational setting;
4° Design several subprojects in which the system should be developed and implemented;
5° Provide a maintenance structure based on the strategy plan, including a costs/benefits analysis.

Phases 1°, 2° and 5° lie beyond the scope of this paper, except for a re-examination of FEWECs goal (phase 1° partially) these phases are not discussed. This paragraph elaborates on the subprojects defined (phase 3°). Phase 4°, the design of several subprojects is discussed in the next paragraph. Before continuing with the subsequent section it has to be said that the focal point of this paper is an exemplification and a discussion of the implementation and usage aspects of the REXX-based Client/Server environment developed at FEWEC.

1.4 The strategy plan: a re-examination of FEWECs goal

In the early 1980s TRANSPAS was set up as a kind of 'test' or 'toy' system primarily by students, two lecturers and a system administrator. At the time focus was purely on practice: how can we implement a portfolio simulation system for usage in education? A re-examination of this simple FEWEC goal is shown in figure 1.

At best stock data provided by Beursdata should be stored in a underlying relational database (BeursBase). Then three user application areas can be distinguished: *research, education* and *public relations*. The research and education user applications areas seem more likely than the public relations application area. True, an university position stems from the quality of research and education. The public relations more or less benefit from these facts. At this moment universities in the Netherlands are faced with a significantly declining number of students, partially because of the aging population and partly because of a declining

willingness to study, and therefore competition is high. This is where the public relations part plays a role: FEWEC needs ways to attract potential students. The public relations user application area is distinguished for this goal. FEWEC members, lecturers, researchers and student can aid in application development for this purpose. Other public relations activities are for example finance , i.e. stock and bond investment competitions, school lectures, educational seminars, and media publicity.

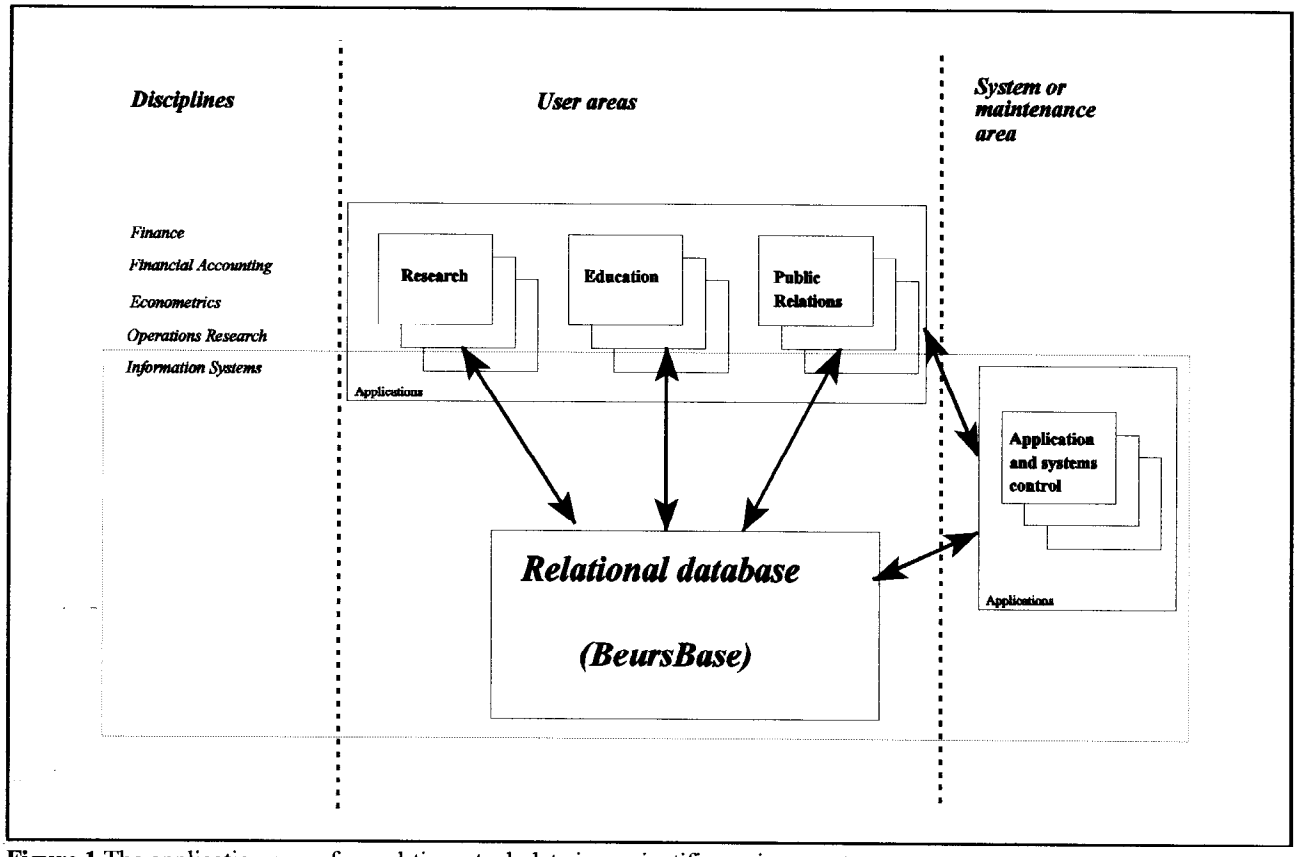

Figure 1 The application areas for real-time stock data in a scientific environment.

The fourth application area is the one for system and maintenance. In this area users, programs and database control rather than stock data are the issue of concern. Finally, each of the user application areas distinguished can be viewed from a variety of economic disciplines. The most relevant disciplines are among Finance, Information Systems and Information Technology, Econometrics, Financial Accounting and Operations Research or Management Science.

The information above can be concentrated into a general purpose framework, a matrix structure in which the user application areas are defined as column and the economic disciplines as rows. For each matrix cell one or more specific users application programs can be developed. Or, if application programs are already available abundantly and are easily adapted in current environment, they should be used instead Furthermore, some application programs if generally developed can be used in more than one area distinguished, and/or in more than one discipline. In that case one can look for or develop application programs for more widespread usage. In table 1, some examples are provided.

The fourth application area forms an exception to the framework. This is primary the area of the Information Systems and Computer Science disciplines and involves more fundamental research on information systems, decision support systems, databases, (tele)communication and networking.

Application area / Discipline	Research	Education	Public Relations
Finance	- Portfolio management and theory; - Theory on market efficiency; - International stock markets; - Development of performance indicators - Analysis of stock market and financial investment data - Application of IS and DSS	- Portfolio management simulation; - Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - Demonstrations; - Financial workshops; - Financial seminars; - Bank and stock market sponsoring; - Media (financial and newspapers);
Information Systems	- Design DSS, ES, KNN systems - Database concepts and theory - System and application development concepts- (Client/Server, Object Orientation) - Application of IS and DSS - Human computer interface	- Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - IS demonstrations; - IS workshops; - IS seminars; - Media (IS and newspapers);
Econometrics	- Statistical (exploratory) data analyses - Longitudinal and time series analysis - Application of IS and DSS - Design of simulation models	- Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - Demonstrations; - Workshops; - Seminars; - Media;
Financial accounting	- Analysis of performance indicators - Analysis of stock and equity issues - Application of IS and DSS	- Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - Demonstrations; - Workshops; - Seminars; - Media;
Monetary economics	- General market theories - Impact stock market on economy - Application of IS and DSS	- Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - Demonstrations - Workshops; - Seminars; - Media.

Table 1 Framework for application program development.

1.5 Criteria for the C/S environment development: in search for processing power

The requirements for a BeursBase and VUPOS platform needed to be specified. Several criteria were defined to determine the amount of processing and database power necessary for BeursBase, VUPOS and other application usage. The project team came up with the following criteria:

- ☐ Number of users in every area distinguished;
- ☐ Number of applications in every area;
- ☐ Type of database processing;
- ☐ In-house experience with systems;
- ☐ Costs/benefits;
- ☐ Designing for flexible systems in terms of portability, efficiency and effectiveness;

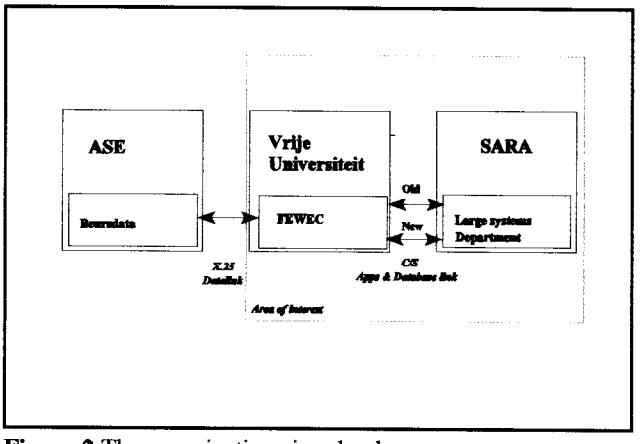

Figure 2 The organizations involved.

Having limited resources available, added to the fact that use of SARAs, the academic computer center of the two universities in Amsterdam, was already paid for up to 1996, the choice was simple. As the department of Information Systems was already using SARAs facilities, the cooperation was intensified. The organizational setting is shown in figure 2. Beursdata, ASEs data vendor provides FEWEC since 1981 with real-time stock exchange data. In 1993 the communication link was upgraded to a X.25 structure. FEWEC, with approximately 3100 students the largest of 15 faculties of the Vrije Universiteit, transforms the real-time ASE data into SQL data-format and puts it into BeursBase (SARA). Ideally, FEWEC members, lectures, researchers, assistants and students should be able to use the data for many different purposes. Data should be available both directly by extracting it (by query) and indirectly by using application programs.

1.6 SARA the computer services center

Since 1987 SARA supports an S/370 facility. During the years it was first expanded from a 3090-150 to a 3090-180 and in 1990 it was replaced by a huge 3090-600VF. The 6 processors were used to run VM, MVS and AIX concurrently. The project team selected the Virtual Machine (VM) operating system to become the BeursBase database server because it came with SQL/Data System installed. The MVS operating system did have DB2 installed, though this facility was not supported for general usage. During the development of the REXX-based Client/Server environment it was found that the SQL/DS installed did not support a server mode. SARA didn't want to invest in a higher SQL/DS level, as it had other plans. But first, the project team decided to develop itself the necessary Client/Server programs, as will be discussed in paragraph 2. Without a 'true' Client/Server environment the project team also decided to develop the first user applications on the VM host (with the IBM development environment Cross Systems Product).

The first signs of strategic movements were already disclosed in 1992 as the 3090-600VF (S/370) was replaced by a 9021-720 (S/390). Developments accelerated in the fall of 1993 when SARA announced it would stop its VM service at the end of 1995. First it reduced the 9021 to a 580 when it stopped the AIX service on this machine. SARA decided that the future role of AIX would become more important, therefore it adopted a new facility, a one of IBMs new developments: a AIX cluster of RS/6000 computers. The cluster installed consists of four 590s and three 980s each equipped with 1 Gb RAM and 6 to 10 Gb disk space. The reason why SARA adopted the new hardware is because it is relatively simple to add processing power to the cluster. One simply adds one or more RS/6000s. Another reason is that an AIX cluster supports 'farming', or distributed processing, which was highly necessary for the high performance computing services offered to the more technical faculties Chemistry and Physics.

Though already in the final stage of development of the VM C/S environment, the project team decided not to wait for more developments to come, but to test the REXX-based C/S environments' flexibility immediately. Because the new AIX facility was provided with IBMs latest version of the DB2 database management system, DB2/6000, now a real C/S environment became an option. The project team decided to develop in parallel to the VM C/S environment an AIX C/S environment in which DB2/6000 would operate as a real database server. It was hoped for that REXXs flexibility would minimize the redevelopment effort.

1.7 Keeping focus

Inorder to avoid problems in discussing the REXX-based C/S environments, the following commentary is necessary. First, ***not one but two*** REXX-based C/S environments were developed. The VM C/S environment is referred to as the 'old' C/S environment, the AIX C/S environment is referred to as the 'new' C/S environment. Secondly, the C/S environments have two levels: a X.25 data link level (ASE-FEWEC) and a database level (FEWEC-SARA). The C/S implementations have a common,

i.e. fixed, X.25 data link (receiving ASE data). With respect to implementation, the area of interest in this paper is primarily the database level (see figure 2). The difference in implementation of the C/S environments stems from the fact that DB2/6000 (the AIX RDBMS) ***is*** and SQL/DS (version 2.2 of the VM RDBMS used) ***is not*** a database server. Thus to establish a VM C/S environment we needed to develop our own C/S environment. From an application and database perspective the VM C/S environment is not a genuine C/S environment because both applications, e.g. VUPOS, and database, i.e. BeursBase, reside at the host. This is in contrast with the AIX C/S environment which fully supports client applications. Thirdly, both systems are fully operational and perform well. Fourthly, with respect to design, development, and implementation issues both VM C/S and AIX C/S implementations are discussed from developer and user views and from both technical and non-technical points of view. Fifth, after VUPOS was implemented further application development for the VM C/S environment was stopped. Thus a discussion about future plans and strategy in this paper, refers to the AIX C/S environment.

2 The design for a real-time Stock Exchange client/server environment

2.1 Moving to a client/server environment: the first level

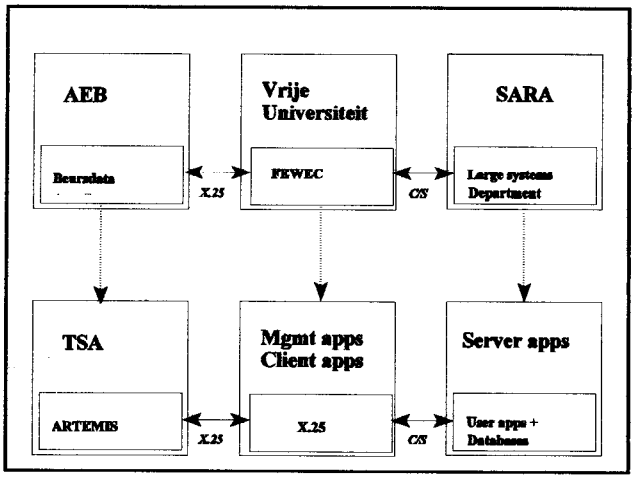

Figure 3 The organizations involved and their systems.

Figure 3 shows the ideal setup of the organizations involved and their systems. The ARTEMIS system of Beursdata is fed on a real-time basis by ASEs trading system TSA. FEWEC has to develop X.25- based communications, a local real-time data link, including a subsystem for temporary data storage. Also FEWEC has to adapt a C/S environment for data and information. Data received should be inserted in a database server (BeursBase); information should be retrieved by client applications from the host database. The C/S environment should be developed using open communication standards.
For manageability reasons the REXX-based Stock Exchange C/S environment project is divided into the following sub-projects, see table 2.

3 Client/Server environments

3.1 The Client/Server concept

Since its introduction, the concept of Client/Server has been discussed among a broad range of disciplines by a large number of people. Scientists, business professionals and many others have been vividly discussing of what one should and should not include in the C/S concepts. As there are so many distinguishable perspectives as opinions, no full-proof C/S definition has been formulated, so far. This paragraph elaborates on our ideas of C/S, hereby avoiding great difficulties and long discussions in placing our C/S environments into general C/S frameworks.

3.2 The five Client/Server levels

C/S can be viewed from both technical and non-technical perspectives. The C/S concept is limited to four aspects, namely, data, database, application programs and users. These four areas include both technical and non-technical perspectives. In general one can distinguish five C/S levels which are exemplified in figure 4. Every higher C/S level inherits lower level functionality.

In the next paragraph first the general levels of C/S are discussed. Then table 2 is further explained by schemes of the architectures of the two C/S environments.

Project	Subproject	Description
A) The X.25 data link	(1) X.25 Communication control	The X.25 data link is logically divided into two parts. The first part receives data packets through a number of logical circuits. These data packets are numbered sequentially per logical circuit. The second part validates received data packets on consistency and sequence number. If one or more data packets are missing or corrupted, this part of the X.25 data link does a retransmission request.
	(2) X.25 data link monitor	In order to have control of the X.25 data link and to be able to automate this part of the C/S environment, it is highly recommended to have a GUI-based monitor. This monitor has to show the status of necessary active communication programs, the logical circuits and information about the amount of data and data packets received, retransmitted and the retransmit status. If one or more hardware, software, X.25 or file errors occur, the GUI-based monitor should be able to restart system components, or the system itself.
(B) The relational database BeursBase	(1) Relational database model	The data received from the Stock Exchange has to be formatted appropriately without losing any information for later data manipulation. In compliance with the ARTEMIS data dictionary a relational data model was developed. This data model is logically divided in two parts. At the one hand all X.25 data received is stored into a submodel. On the other hand application and system programs store their own specific data.
	(2) DBA manipulation, monitoring and authorization programs	Though application programs for database, not data, manipulation, database monitoring and authorization are necessary, the are out of the scope of this paper. At the moment these applications are under development and will be REXX-based.
(C) Client/Server environment	(1) data-SQL converter	Before the data received can be stored into BeursBase, one has to remodel the data to the relational model defined. The function of this program is to create SQL DML statements out of the data received.
	(2) BeursBase link	An application which provides the data and session link layers from the PS/2 to the database (BeursBase) for SQL DML statements execution. In the VM C/S this function is partially implemented by a REXX FTP-based program. In the AIX C/S environment this program is replaced by a C/S software package named Client Application Enabler/2 (CAE/2).
	(3) SQL statements executor	This application program executes the SQL DML statements into BeursBase, in both C/S environments. In the 'old' VM C/S the application runs at the VM host, in the 'new' C/S environment it runs at the pc.
	(4) C/S monitoring program	Like the X.25 Monitor the Client/Server Monitor information about the C/S and BeursBase status is necessary. Like the X.25 environment the C/S environments should run continuously and autonomously. For the 'old' VM C/S environment the monitor is slightly different than for the 'new' AIX C/S environment.

Table 2 An overview of the projects and components of the REXX-based C/S environments.

(A) A Client/Server data link.

If the database as well as user application programs reside on one computer system, one cannot speak of C/S. In our C/S environments, the database residing on this computer system, is fed remotely with data. Therefore we argue that, from a data perspective, this is the most simple form of C/S. In general, if one looks at the user, database or application program perspectives this is considered not to be C/S. The reader should keep in mind that all subsequent C/S levels discussed use this 'C/S level'. As will be shown later, it is this data C/S level which distinguished the two C/S environments developed.

(B) Application program clients - application program/database server level

In the second C/S level simple computers of end-users are the clients populated by several relatively small application programs. These application program make use of a computer system acting as a database server. In addition, the remote host system

is reachable by the end-user too, for larger applications programs, as the local computer system lacks performance and/or the communication line capacity is too small for large scale database I/O. Then, in that case the end-user computer system is used as a dumb terminal as both processing and database I/Os are executed at the remote server.

(C) Application program clients - remote database server level

The next higher level of C/S is when all applications are executed at the clients and the host purely acts as a database server. Thus advanced and complex processing is done at the client level. Three requirements have to be fulfilled. First, faster and more complex client computers. Secondly, a sophisticated communication infrastructure is necessary to cope with large scale database I/O and processing. Thirdly, the application programs developed are by far more complex than in the case of C/S level B.

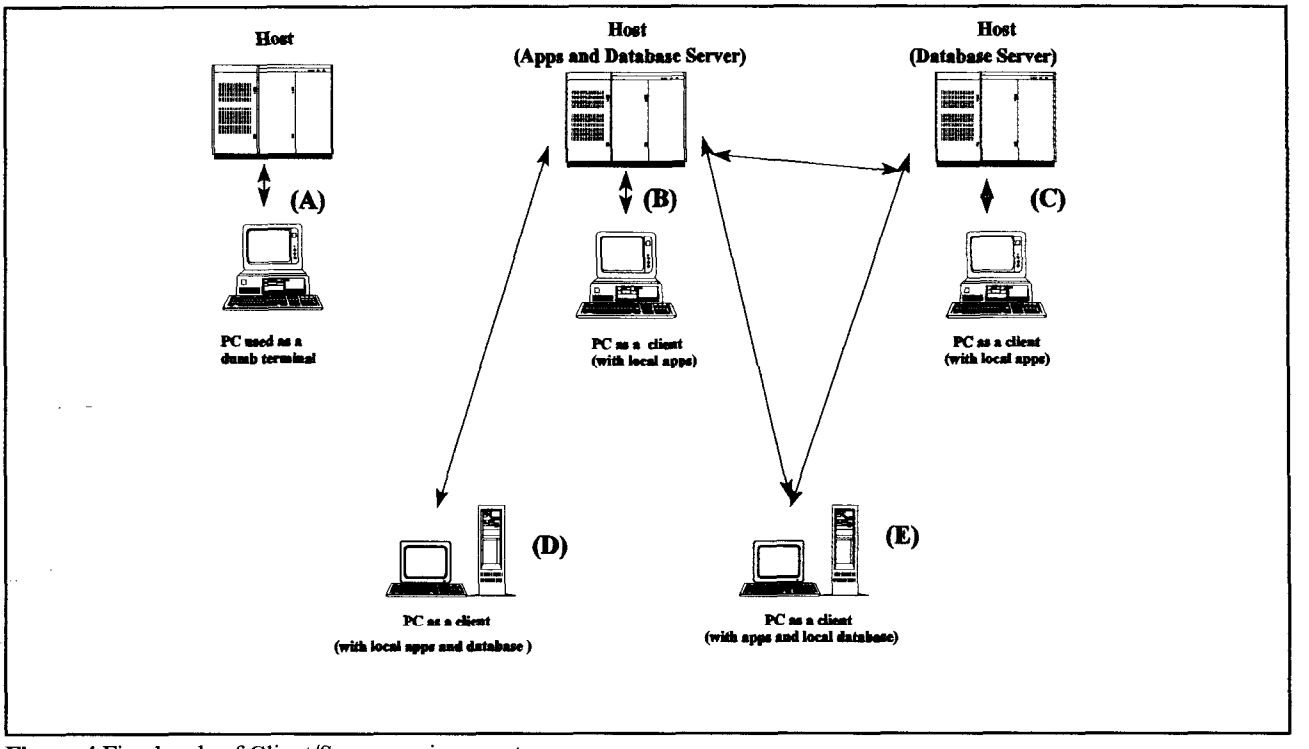

Figure 4 Five levels of Client/Server environments.

(D) The distributed database level

An even more advanced C/S level is established when database processing is distributed and fully integrated in the C/S environment. In this case one has remote as well as local database servers co-operating in a C/S environment. The client computers system can be such a local database server as well. Thus it is not important where the data is stored and where the processing is done. At this level data is stored where it should, for example user specific data is stored as close as possible to the user. In addition, for performance reasons, the distributed database environment decides where to store application specific data. For performance reasons this can be as close as possible to end-users, to some extend introducing data redundancy, or as central as possible at a remote database server. Thus the distributed database environment decides where database processes can be handled at best. At this level client computers can act as application clients as well as local database servers simultaneously. At this C/S

level due to limited client computer system performance (as high performance computing applications programs need) end-users are still able to use application programs directly at some host system, which acts as both database and application server.

(E) The distributed application level

The most sophisticated level of C/S is established when, aside from the distributed database environment, the application programs too are distributed among several clients and servers. Thus the C/S environment decides where to handle data as well as application program processing. For example I/O intensive database operations are executed at a specific database server , CPU intensive calculations are executed on a specific application programs server, small database operations are run on a local server (client) and the remainder of the application programs are executed on another client. In research nor in the business environment has the fifth C/S level been implemented yet.

In summary, the FEWEC-SARA data link in both VM (SQL/DS) and AIX (DB2/6000) C/S environments is purely C/S level A. The AIX C/S environment from an end-user and application programs perspective, is of C/S level B. Before discussing in greater detail the future plans and strategy with respect to the directions the AIX C/S environment will move, the developments of the C/S environments so far, are further discussed.

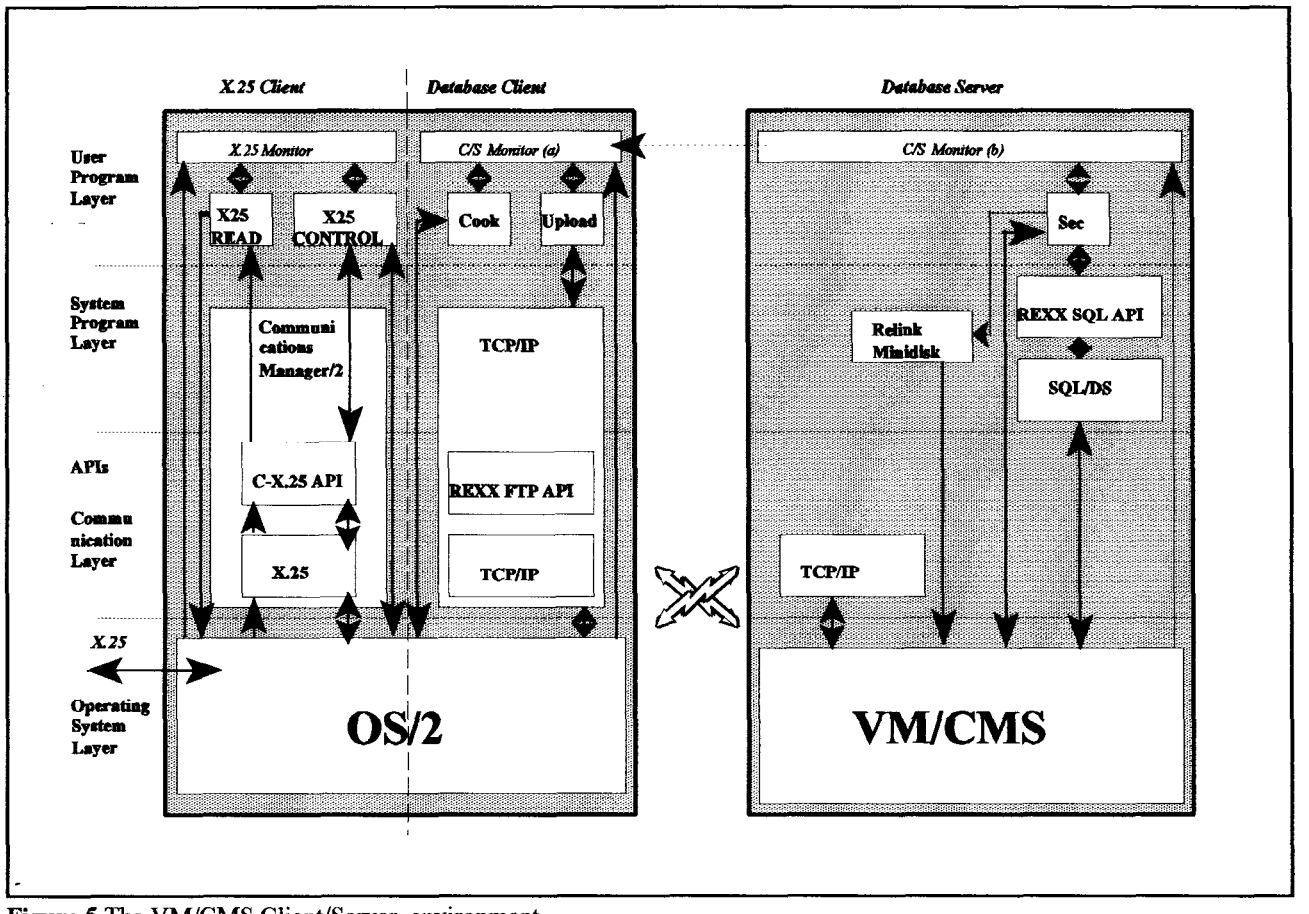

Figure 5 The VM/CMS Client/Server environment.

3.3 The VM-C/S environment

Figure 5 shows the architecture of the VM C/S environment. The data flows distinguished are explained briefly. The X.25 data packets are received by X25READ (arrows up) and written to a file (arrow down to OS/2). X25CONTROL checks this file on validity (packets complete?), consistency (right sequence?) and completeness (all packets received so far?). If not the case retransmission of one or a range of data packets is requested (arrow down) and, when received (arrow up), written to a temporary file (arrow down). The temporary file appended to the first file, is written to a complete data packets file by X25CONTROL (arrows up and down). Reading the complete data packets file (arrow up), Cook decides which lines read have to be converted to BeursBase data model formats (SQL). Lines containing only a synchronizing timestamp, are not used directly as we will see later. Each SQL statement generated is put in a separate file (arrow down), preceded by the parsed aebfcode, timestamp created and some C/S control parameters. Next, Upload sends new files (arrow up) to the host (arrow down) through a FTP connection (TCP/IP). At the host, the SQL statements executor, Sec, sequentially reads received files (arrow up) and first creates a SQL statement based on the parameters (the aebfcode plus timestamp) found at the first line. The result of execution of this SQL DML (table SELECT) statement (our unique fcode) is combined with the SQL statement, containing the actual SQL DML operation (table INSERT, UPDATE or DELETE), found in the file. This SQL statement is subsequently executed into BeursBase (arrow down).

For a continuous autonomous environment, VM file management is quite different compared with other operating systems like UNIX, OS/2 and MS-DOS, a VM file limitation had to be overcome. In contrast with a hierarchical file structure used by most of the operating systems, VM uses a flat file structure. Files are stored with the format <filename, 1 to 8 characters> <file type, 1 to 8 characters> <file mode, 1 character plus 1 digit between 1 and 6> on a so-called virtual minidisk. Minidisks are mapped onto physical storage devices (DASDs). To be used by a program a virtual minidisk has to be linked physically (by a CP LINK command) and has to be logically attached (by a CMS ACCESS command). VM allows multiple links, both in exclusive or shared read and/or write modes. Because links to minidisks are static links, file operations by one program are not 'seen' by another program. Thus, to establish a real-time VM C/S environment, Sec has to refresh its link with the minidisk where Upload writes the SQL DML data files to. This is done every time Sec does not find the next file in sequence (file type = number). To avoid the probability that Sec updates the link to the Upload minidisk continuously, Sec pauses a few seconds when after relinking no new files are found.
Every program mentioned writes a status file to be read by one of the two monitor programs, X.25 monitor and VM C/S monitor. To establish a C/S environment, the VM C/S monitor needs information about Sec. Since it cannot read directly the Sec status file, it has to be frequently downloaded by Upload. In addition , Upload reads the Sec status file frequently too, in order to determine successful execution of SQL statements, and to delete corresponding files by issuing remote (FTP) deletes accordingly.

3.4 The AIX-C/S environment

At a first sight the architecture of the AIX C/S environment shown in figure 6 is almost identical to its VM C/S counterpart. However, what seem 'minor' differences in design, with respect to the VM C/S environment, results in tremendous improvements. First, because a standardized C/S interface (Client Application Enabler/2) is applied, Upload is dropped. In addition, because Sec executes the SQL DML statements from the PS/2 (client), no special file operations (minidisk link refresh) are needed. Thus, this C/S environment gives advantages from both control and integration perspectives.
At the one hand, better control (effectiveness) because all status (file) information is directly available to the C/S monitor program, without necessary tricks. On the other hand, integration of functions is enhanced (efficiency) because programs involved run concurrently on one computer.

In the next paragraph we discuss the C/S architectures in greater detail, especially with respect to REXX and its interfaces used in the C/S environment.

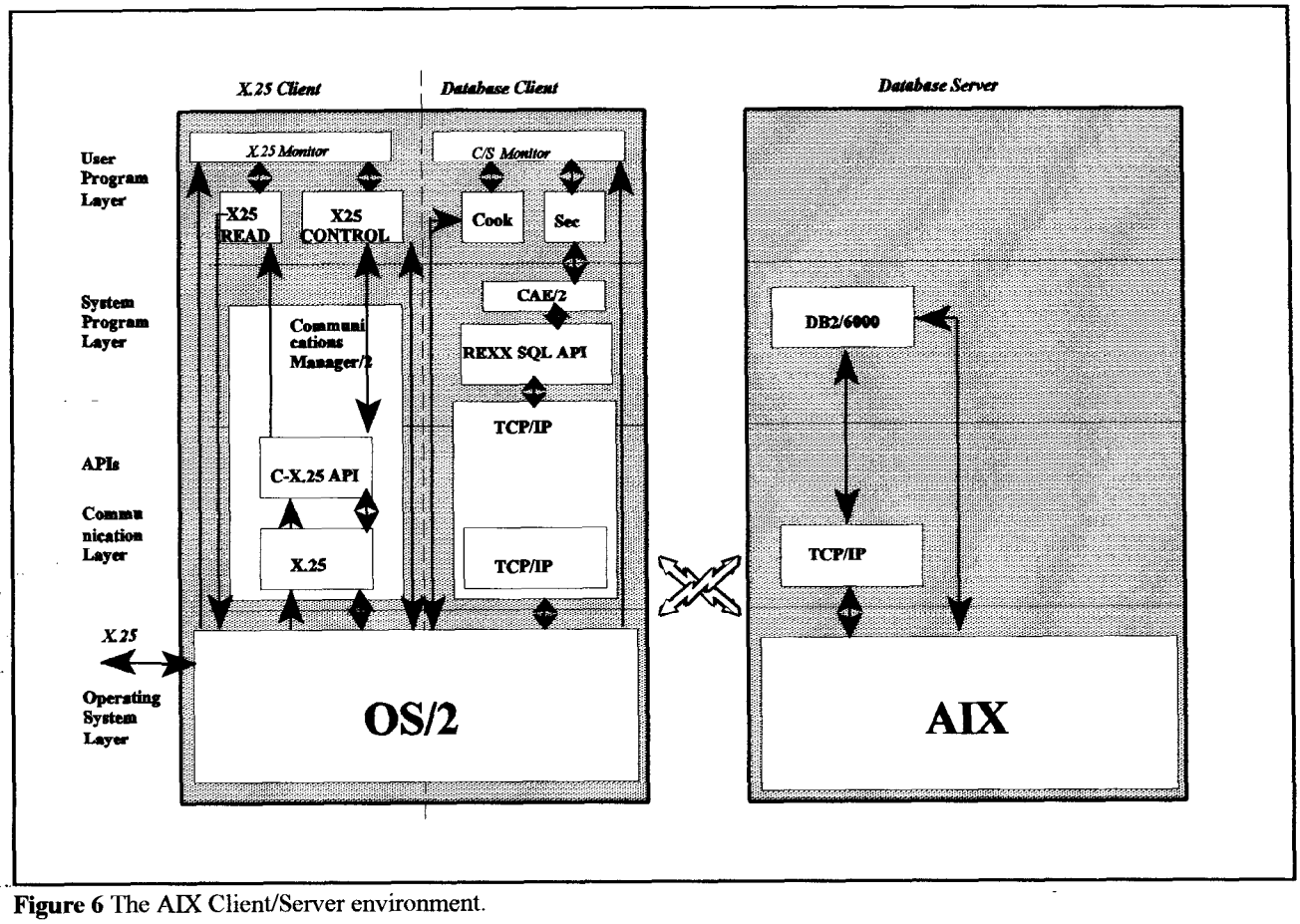

Figure 6 The AIX Client/Server environment.

4 REXX and the interfaces used in the VM and AIX C/S environments

4.1 Introduction

The basis of this paper is to elaborate on the impressive role REXX performed in development and implementation of the VM and AIX C/S environments. In particular, programming techniques, tips and tricks applied form the main subject for this paragraph.

Basically, the answer on the question why REXX forms the core in the development of the VM and AIX C/S environment, is visualized in figure 7. This figure shows that REXX, compared to other programming languages and development environments, is exceptional with regard to supported interfaces. True, REXX is not the exclusive language having so many different interfaces, in this respect for example is C of equal quality, though it's the ease of us which makes REXX unique. In addition, the fact that REXX programs cab be both interpreted and compiled, makes REXX special.
For each interface shown in figure 7 some general and REXX programming concepts applied are discussed.
Unfortunately, to avoid lengthy discussions, only some glimpses are provided.

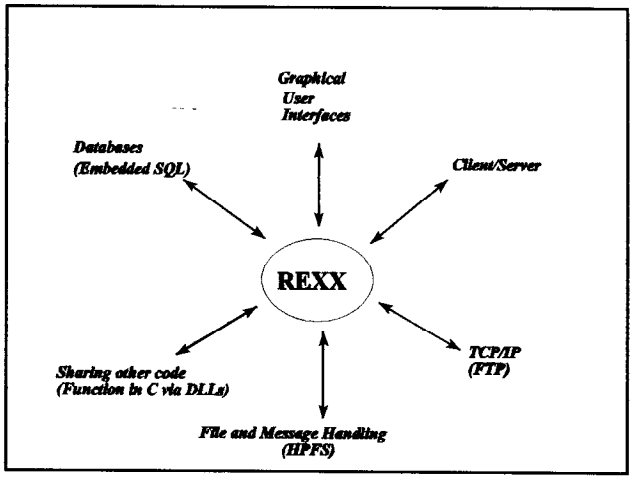

Figure 7 REXX interfaces.

4.2 REXX flexibility in C/S environments: designing for both performance and portability

Our experience is that REXX can be used very effectively in C/S development. One should be aware of the programming power which comes with REXX. Like any language one can benefit tremendously if one is cautious about performance and flexibility. First, develop REXX programs as universal, i.e. system-independent as possible for portability reasons. Secondly, do not make use of operating system specific functions, unless there is no alternative available. Thirdly, code well-documented, though as compact as possible. This is especially true if one does not use compiled code. Fourthly, when necessary and if possible, test programs on different hardware as soon as one can. Don't wait until there's no way back.

4.3 REXX and embedded SQL: the REXX-SQL interface

Aside from some exceptions, most of the administrative information systems in the economics discipline, especially in the business environment, are characterized by only a few fundamental functions. These comprise data storage, data manipulation and information retrieval and presentation. Since its introduction as a general purpose query language, its popularity is growing. Nowadays use of SQL, an acronym for Structured Query Language, is widespread. For years now, SQL has a solid place in the FEWEC IS curriculum. Its adoption in the C/S development was inevitable.

The REXX-SQL programming interface is available for all IBM database management systems (SQL/DS, and several DB2 versions). Despite some minor differences the REXX SQL programming interface is implemented uniformly for all database management systems. The usage of the REXX-SQL programming interface in the C/S environment is explained by the following example. The example shows how SQL database manipulation language (DML) statements are generated by an data-SQL converter, which we called ***Cook***, and how the are interpreted by ***Sec***.

Each line from the checked data packets file is examined by Cook. If Cook finds relevant data, it 'cooks' the corresponding SQL DML statement, based on the specified action on the data line. Lines not of interest should not be converted into SQL DML statements, though there is one exception: Cook does not use lines containing control data consisting of synchronizing timestamp for the X.25 connection. However, lines containing ASE trade volumes data do not have a timestamp. In that specific case Cook uses the timestamp found in previous line processed, which sometimes contain such control data.

```
1 1887ANFA36058NL0000360584     8104811 NLG2      4040            55500
3  864AOFC22311NL0000223113      220000    2
1 1888AOFC37750NL0000377505       19655   15
2  783ANFA00208NL0000002087     8104824 NLG2     10040
1 1889ANFA00921NL0000009215    53104826 NLG2      6000           245136
1 1890ANFA34948NL0000349488    30104826 NLG2      1680           975360
8 1665AQFL34948NL0000349488      104828 NLG2      1680           19800          1690          103800
```

Example 1a Small portion of the checked data file.

The body of Cook is a huge four-level select, which corresponds with four alphanumeric characters found on positions 8 to 11 of the parsed data line. The *parse* command, like *select* another powerful REXX feature, has been applied in Cook many times. This increase flexibility and maintainability significantly, in contrast with direct (static) usage of the BeursBase data definitions. The last line of example 1a shows the code **'ANFA'**, which stands for **'ASE'**, **'Price'**, **'Stock'**, **'New'**, meaning that ***at the Amsterdam Stock Exchange a new price for a stock transaction*** has been established. Based on the ARTEMIS data dictionary Cook builds a SQL DML INSERT statement for the NOTERINGEN table (example 1b).

```
1995-02-20-16.39.13 O 00173 0
INSERT INTO V67CVPOS.NOTERINGEN VALUES (:fcode, '1995-02-20-16.39.13',    5, 'K', '0', 99.9, 'H', ' ', ' ')
```

Example 1b SQL DML: INSERT statement generated by Cook preceded by timestamp, C/S control parameter 1, ASE fcode and C/S control parameter 2.

The SQL DML statement is preceded by the ASE timestamp, a C/S control parameter, the ASE stock code (aebfcode), and a second C/S control parameter. Only the ASE parameters were found in the raw data file line, the other two parameters were added by Cook. In the SQL DML statement the host variable **:fcode** is put in place of the ASE stock code, because the ASE stock code is not unique over time. First Sec parses the SQL DML statement from the file which is preceded by the creation timestamp, a C/S control parameter, the ASE stock code (aebfcode), and a second C/S control parameter found in the raw data file. Then Sec generates a SQL DML statement to retrieve the unique FEWEC stock code using the ASE stock code and the creation timestamp (example 1c).

```
command = "SELECT fcode FROM" InstrumentenTabel,
  "WHERE aebfcode = :oldfcode",
  "AND ts_intro  <= :timestamp",
  "AND ts_extro  IS NULL"
SQLS = 'SQLGETOLD'
CALL PrepareS
```

Example 1c SQL DML: SELECT statement generated by Sec to search for the unique FEWEC fcode.

The characters preceding the SQL DML code are stripped away and the found FEWEC stock code is put in place of the host variable :fcode. Now this SQL DML statement is executed by Sec.
Stocks are characterized by both a moment of introduction and a moment of extroduction. The majority of stocks once introduced exist permanently, however, there is always a possibility that a stock may be extroduced. Stocks can be extroduced for many reasons. In case of a management buy-out, a stock split or a bankruptcy of the firm, trade is ended, and sometimes a new stock will be introduced. In contrast with stocks, bonds are always extroduced. ASEs policy is that after stock extroduction the stock code comes free and is re-usable. If one intends to store all stock prices ever listed, like in BeursBase, one has to introduce

```
command = "SELECT MAX(fcode) FROM" InstrumentenTabel
SQLS = 'SQLGETNEW'
CALL PrepareS
```

Example 1d SQL DML: SQL DML statement to retrieve the maximum FEWEC fcode.

instead a unique stock code. Thus every time ASE introduces a new stock accompanied with their 'unique' ASE stock code (aebfcode), we have to map this stock code to a time-independent one. A simple solution is to use the maximum FEWEC stock code found in the stock table, plus one (figure 1c). For identification purposes and to keep track of all stocks, one needs to store both stock codes as well as both dates of introduction and extroduction. If stocks are still tradable their extroduction remains empty, in SQL we set the data value to NULL.The generated unique fcode is subsequently used in a SQL DML INSERT statement to add the stock introduced to the INSTRUMENTEN table (example 1e). Identical to the SQL DML insert statements update and delete SQL DML statements are generated, except for the INSTRUMENTEN (securities) table from which a delete is not allowed.

Many additional remarks can be made, however, three essential are discussed. First, it is never the case that a stock price precedes stock announcements. Thus the introduction of a new stock (or a renewed stock, for example, due to a stock split) happens always before new stock prices arrive and therefore there exists always a unique FEWEC stock code (the SQL statement generated is an INSERT into the stocks table). Secondly, official stock prices send can be modified or even withdrawn. Thirdly, if the X.25 connection is closed in the evening, a final file is generated. In contrast with the other files uploaded, this file does not contain a SQL DML statement but an 'End-Of Day' message. This way Sec is notified that no more files will be send that day.

```
SQL1 =      "INSERT INTO" InstrumentenTabel "VALUES (",
    ":fcode,",                          /* FCODE              */   FEWEC 'unique' stock code
    ""fondsnaam",",                     /* FNAAM              */   ASE security name
    ""fondstype",",                     /* FTYPE              */   FEWEC security type (bond, stock, warrant, et cetera)
    ""timestamp",",                     /* TS_INTRO           */   ASE introduction date
    "NULL,",                            /* TS_EXTRO           */   ASE extroduction date
    ""isincode",",                      /* ISINCODE           */   Official International Standard Identification Number
    fondscode+0",",                     /* AEBFCODE           */   ASE stock code
    rentoperc",",                       /* RENTE              */   Interest (bonds)
    ""marktcode",",                     /* MARKT_CODE         */   Market code (official market, non-official market, et cetera)
    ""noteringswijze",",                /* NOTERING           */   Current price
    ""valnotering",",                   /* VAL_NOTERING       */   Valuta traded
    ""valafrekening",",                 /* VAL_AFREKENING     *    Valuta paid
    ""swaif",",                         /* FONDS_SW_AIF       */   ASE
    ""omzetcode",",                     /* OMZET_CODE         */   Amount traded code
    ""cdatum1",",                       /* CDATUM1            */   Coupon date 1 (bonds)
    ""cdatum2",",                       /* CDATUM2            */   Coupon date 2 (bonds)
    eenheid",",                         /* EENHEID            */   Number traded
    "0,",                               /* NOMINAAL           */   Nominal value
    ""fondstype2",",                    /* FONDSTYPE          */   ASE security type (bond, stock, warrant, et cetera)
    ""swvoorlopc",",                    /*                    */   Definition 1 in official ASE trade newspaper
    ""swdefopc",",                      /*                    */   Definition 2 in official ASE trade newspaper
    ""symbol",",                        /*                    */   Symbol
    ""kleinstecoupure",",               /*                    */   Smallest number of instrument available
    ""coupondate",",                    /*                    */   Dummy coupon date
    interestfrequentie",",              /*                    */   Interest frequency
    ""intereststart"",                  /*                    */   Interest start
    ")"
```

Example 1e SQL DML: INSERT statement from Cook regenerated by Sec with the unique FEWEC fcode.

4.4 REXX and GUIs

REXX on IBM platforms doesn't come with a sophisticated GUI. One reason one can think of lies in the diversity of systems, in terms of hardware and software to be supported. The GUI is the most hardware and software, i.e. operating system, dependent of all software components. For example, mainframes support primarily text-based character terminals. Personal computers work with the OS/2 GUI known as the Presentation Manager. AIX based RS/6000s use the widely accepted UNIX GUI Xwindows extended with OS/F Motif. Then making it even more complex some GUIs are supported on multiple operating systems for instance Xwindows is supported at the PS/2, the RS/6000, the S/390 and SP families. Finally other hardware manufacturers have adopted REXX onto their systems like SUN (Sparc), Hewlett Packard (HP xxxx) and Commodore (Amiga), for instance. IBMs strategy not to support a platforms wide REXX GUI is the only choice.
Thus for software (especially programming and development tools) manufacturers specializing merely one or two platforms REXX-supported GUIs can be profitable market. For PS/2s there are two REXX-based GUIs available. At the one hand there is VX-REXX marketed by Watcom, and VisPro REXX marketed by VisPro. The first REXX GUI FEWEC acquainted with, VX-REXX, was bought. Not to undervalue VisPro-REXX, FEWECs choice was not a poor one.

4.5 REXX and TCP/IP: REXX FTP API

In the VM C/S environment the REXX-FTP interface plays an vital role. Without this interface the quasi C/S environment could not be established with REXX. The REXX-FTP interface, written by several IBM employees, became available as freeware, add-on product of TCP/IP for OS/2, in 1993. The first author of this paper was first acquainted with this product in June 1994. After solving installation and operation problems, we acknowledgement several persons in the REXX and TCP/IP community, as the REXX-FTP interface now works excellent. As an elaboration on section 3.3, our utilization of the REXX-FTP interface is discussed next.
At the PS/2 Cook reads the file containing the X.25 received data packets. Other lines with data packets with synchronizing timestamps for the X.25 connection, are only processed for these timestamps. Cook converts each relevant line into a SQL DML statement and subsequently writes it to a OS/2 HPFS file with a naming convention of <yyyymmdd.#>, where # stands for a sequence number. When started Upload starts an FTP session with the VM host via the REXX-FTP interface with the ftpuser function. The core of Upload is a loop in which several actions are programmed. Upload pauses until new files have been 'cooked' (Cook), then uploads them individually to the host using the ftpput function. If successful uploaded, Upload deletes the local file. Another action within the Upload loop is the frequently download of the Sec status file using the ftpget function (table 3). The download frequency is set based on a fixed number of uploads. Every day the FTP connection is closed using the ftpclose function and Upload is ended. Early next day Upload is started again by the VM C/S Monitor. Cook (packets processed) pauses when the Upload (packets uploaded) delay exceeds a certain threshold (200 files). This is necessary because the number of files in a local directory is negatively correlated to Uploads speed. Analogous to local file deletion, Upload frequently erases remote files, successfully executed by Sec, using the ftpdelete function. For the VM C/S Monitor, Upload frequently writes its operational status to a local file (table 3).

Program name	Filename used	Description
X	PVC.raw*	all data packets received, but not checked on consistency, validity or sequence;
	PVC.status	a status file containing information about the PVCs. This information is displayed in the X.25 monitor;
*XX**	HERTRANS.tmp	a buffer for the retransmitted data packets;
	CHECKED.raw**	the complete checked set of data packets so far;
	CHECKED.status	a status file including information about the PVCs (number of retransmissions, total number per PVC) and the CHECKED.raw file offset which happens to be the same as its filesize). This information too is displayed in the X.25 monitor.
	yyyymmdd.CHECKED	at 11.00pm the CHECKED.raw file is renamed with the current date at the beginning of the filename.
Cook*	Cook.status	a status file containing information about Cook to be displayed in the C/S monitor;
	yyyymmdd.#***	every SQL DML statement written to a file starting with the date and ended by a sequence number (#).
	yyyymmdd.#S***	stop file
	yyyymmdd.Cook.Log	Dependent on the loglevel chosen, every relevant operation (reading CHECKED.raw, creating a SQL DML, filing the SQL DML) is logged into this file.
	yyyymmdd.Cook.Status	all information needed for the C/S monitor is written to this file.
Upload* (VM C/S!)	Upload.Status	a status file containing information about Upload to be displayed in the C/S monitor
	yyyymmdd.Upload.Log	every operation (FTP connect, FTP put, FTP get, connect status, current activity) is written to this file.
	yyyymmdd.Upload.Status	all information needed for the C/S monitor is written to this file.
Sec*	Sec.Status	a status file containing information about Sec to be displayed in the C/S monitor
	VM: yyyymmdd.Log AIX: yyyymmdd.Sec.Log	every SQL DML statement file read and executed (SQLCA) is written to this file.
	VM: yyyymmdd.Status AIX:yyyymmdd.Sec.Status	all information needed for the C/S monitor is written to this file.

Table 3 Some filenames used by the X.25 data link programs.
*,** and *** mean that the files referred to are used as input for the particular program.

4.6 REXX and X.25

Although it was not used in the development of the C/S environments, the X.25 interface to the OS/2 Communications Manager (CM/2) is a good example of the importance of REXX in communications. The CM./2 X.25 interface is provided for a variety of programming languages such as C, COBOL, FORTRAN, Assembler and REXX. The main reason not to use REXX for X.25 based communications is in order to control the X.25 data link in terms of priority scheduling and multithreading and to secure application performance, one would be better of with C. As will be discussed later in this paper, action is taken to take a more detailed look to port the C-based X.25 programs to REXX.

4.7 REXX and file handling

From maintainability and flexibility perspectives it is wise to develop REXX programs using High Performance File System (HPFS). HPFS, which comes standard with OS/2, should be preferred over the 8.3 character limitation of the DOS File Allocation Table file system (FAT). In our REXX-based programs we benefitted from the HPFS features which allows using long filenames

which can be as 255 characters long. In table 3 the most important filenames used in the X.25 data link are shown. In the development of the C/S environments several advantages of using HPFS over FAT were exploited.
First, if a system crash occurs HPFS file are almost always recovered. Secondly, it is preferable to use semantically sound filenames. With the 8.3 character FAT limitation it is impossible to name files appropriately.

4.8 REXX and C programming

In the development of the C/S environments, like REXX, C plays an important role too. As discussed earlier, C was used to develop the specific X.25 programs. In addition, some general unctions , implemented with C, were developed for C/S simulation and to overcome some REXX limitations. One of these general C-functions, xread, was to overcome the REXX file-sharing limitation. REXX programs have to find external functions in a so-called Dynamic Link Library, or DLL. Such a DLL has to be developed and compiled by C.
This paragraph is concluded with table 4 listing the major hardware and software components used in the development of the C/S environments. In summary, at the one hand REXX was partially used to develop some of the components required for a C/S environment (Cook, Upload, Sec). On the other hand, for those components not developed with REXX (X25READ, X25CONTROL, external functions), REXX was the 'glue' to integrate these components (X.25 and C/S Monitors) with the REXX-based components. This paragraph focussed on more technical aspects of the C/S environments developed, especially the role of REXX within the C/S environment.

Subsystem: / Components:	X.25 data link	New AIX Client/Server environment	Old VM Client/Server environment
Computer system	PS/2	RS/6000 Cluster (4 x 590)	ES/9021-580
Operating System	OS/2	AIX	VM CMS
RDBMS	DB2/2 (small tests)	DB2/6000	SQL/DS
Client/Server application	CAE/2 CAE/DOS (MS-Windows)		
Client/Server programming	REXX, C Set/++, WorkFrame	REXX	REXX
Communication	TCP/IP CM/2 (X.25)	TCP/IP	TCP/IP
API	C-X.25 API (CM/2)	REXX SQL API CAE/2 and CAE/DOS C-REXX API	REXX SQL API (RxSQL) REXX FTP API (TCP/IP) C-REXX API
Application programming:	VX-REXX C/S	Server (AIX): - APL2/6000 - QueryManager/ 6000 Client:(PS/2s): - Watcom VX-REXX C/S - APL2/2 (special) - VisualGen (4 GL) - VisualAge(OO) - DB2/2 QueryManager - Any ODBC Windows pack-age	CSP/370 QMF/370 VX-REXX C/S

Table 4 The software packages used for design of the C/S environments.

Once established a C/S environment, one can develop a broad range of user and system application programs using BeursBase. The next paragraph, using section 4.4 (REXX and GUIs) as a starting point, discusses the value of REXX for application development. We will elaborate on important aspects of information visualization, templates and object-orientation using REXX. The following applications are discussed: system applications (the X.25, VM C/S and AIX C/S monitors) and user area applications like the AIX version of VUPOS.

5 REXX User programs for the C/S environments

5.1 Database connectivity

In the AIX C/S environment, C/S concepts used are much more sophisticated. For instance, applications at the client, preferably GUI-based, are able to show more information about the C/S environment. In this way a user can be more actively involved. Figure 8a shows an object with user and database information. Depending on the authorization level a user is or is not allowed to modify the information displayed. The object in figure 8a is retrieved through object 8b, by a click on the 'Wijz' pushbutton. If the user is not allowed to do so, the pushbutton is set to not-clickable and its color is changed. If the information displayed is correct or modified, the user can click on pushbutton 'Con.' to establish a connection. In the case of a simple end-user an auto-connect is pursued. Status information about the necessary REXX DLLs as well as status information about the database connection.

5.2 Object Orientation and REXX

Figures 9a thru 9c exemplify a way how to define and use objects in VX-REXX. Figure 9a shows the complex object which consists of a listbox and four entry fields. The object used to display tabular or graphical information is showed after activating the 'Genereer grafiek' pushbutton in figure 8b. At first the object is shown as in figure 8a. Then, based on the user selection from the listbox., one or more properties are changed by message passing. If the user selects 'Real-time', no additional information is necessary to show the tabular of graphical information requested, because the system date can be used. In contrast, if 'Meerdere dagen' (= more than one day) is selected, begin and end date are required. Also if 'Meerdere dagdelen' is selected begin and end time is required too. Using a GUI this way keeps the user's eye focussed onto the display and does not provide the user irrelevant information.

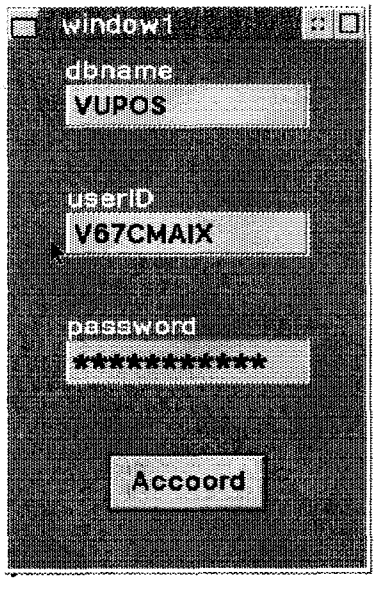

Figure 8a Database connect information.

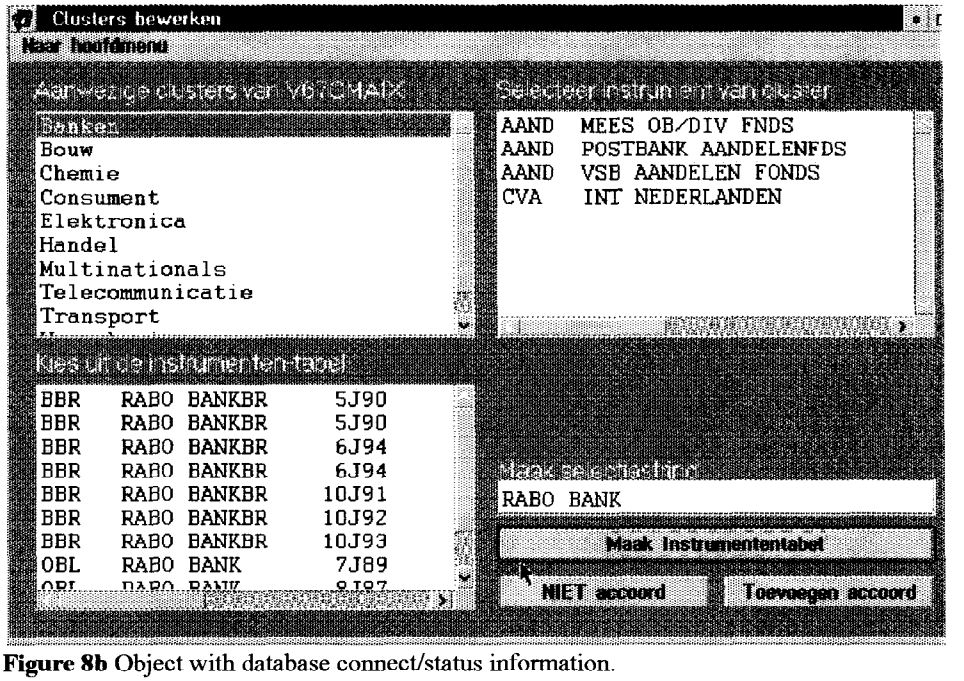

Figure 8b Object with database connect/status information.

Figure 9a Object with hidden entry fields.

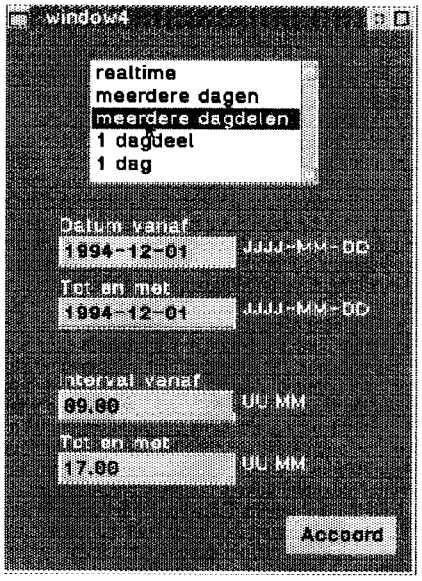

Figure 9b Object with all entry fields shown.

Figure 9c Object with some entry fields shown.

5.3 Developing general parts and templates

A sophisticated way of programming is the use of objects in the form of templates. Object orientation facilitates the use of general parts or templates. An example of a general part is shown in figure 10. This object is especially useful for debugging. Every SQL statement parsed is checked on its SQLCA. If the SQLCA is not equal to 0 (success) or 100 (no more rows certified the conditions specified), further program execution stops and the SQLCA object is displayed.
An earlier example of a general part was already discussed. Figure 8a, the object for displaying database information, is used in every application developed so far.

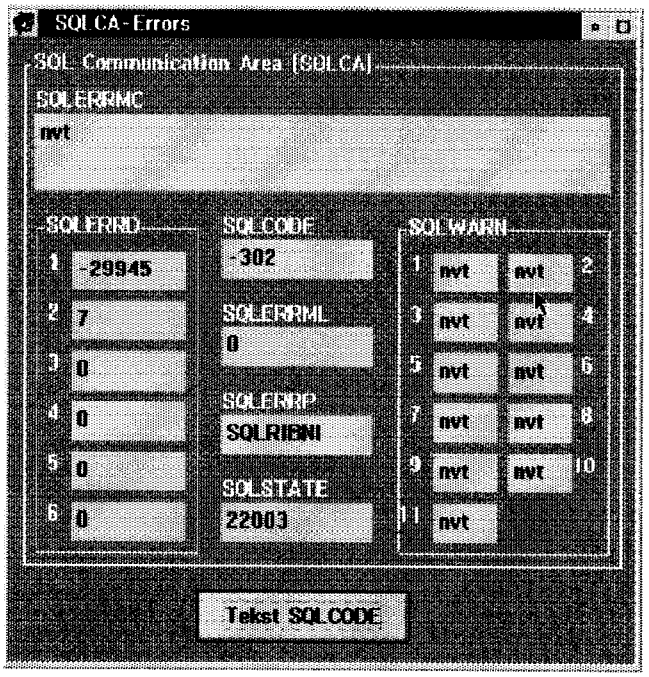

Figure 10 Object for presentation of SQL Communication Area Descriptor.

5.4 GUI design and usage: monitoring

The X.25 monitor
A monitor should be kept simple and should give direct information. The X.25 monitor, developed for control of the X.25 data link displays all the information necessary. Information includes the # of packets received, retransmitted, and the # of retransmissions per logical circuit (PVC group).

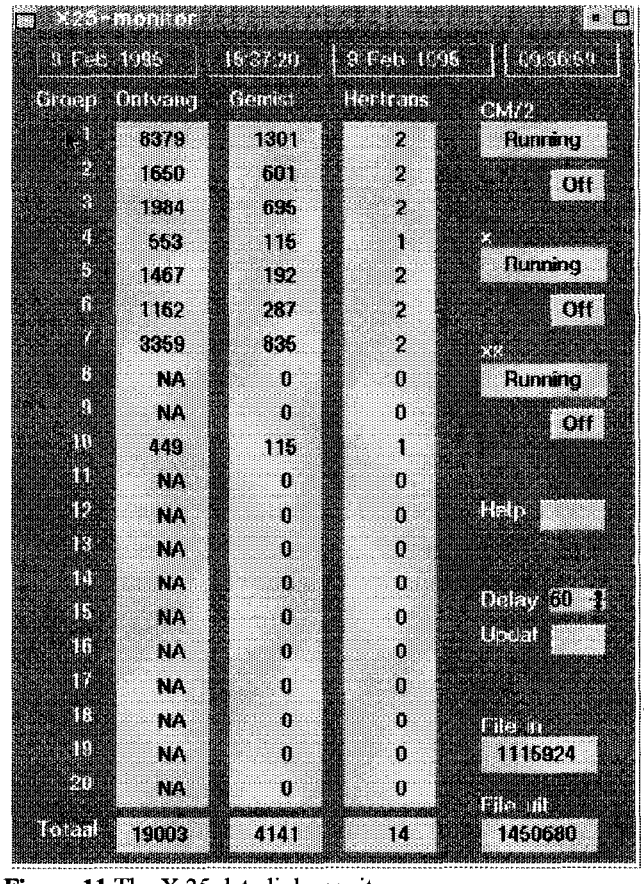

Figure 11 The X.25 data link monitor.

The X.25 and C/S monitors are design to be displayed simultaneously. The information of two monitors, X.25 and AIX C/S are integrated as follows: the difference between the number of received data packets (X.25) and the packets cooked (AIX C/S) is the first packets delay. The packets cooked means the number of SQL DML statements created. The difference between this number and the packets executed (AIX C/S) gives the second delay.

The VM C/S monitor

Notice that the VM C/S monitor is very different from its AIX C/S counterpart. Since the VM C/S environment is user written, information about the underlying program (upload) is displayed. The information of two monitors, X.25 and VM C/S are integrated as follows: the difference between the number of received data packets (X.25) and the packets cooked (VM C/S) is the first packets delay. The packets cooked means the amount of SQL DML statements (files) created. The difference between this number and the packets uploaded (VM C/S) results in the second delay.

In addition vital program information is provided. Finally the sizes of the input (unchecked) and output (checked) files, where the raw data (packets) are stored, is displayed.

Several features are added to the monitor. First, to be certain the screen update process does not consume too much time, a timer event updates the screen periodically. The time period can be set between 10 and 60 seconds in steps of 10 seconds. If an immediate screen update is required, one simply clicks the update pushbutton.

The AIX C/S monitor

Like the X.25 monitor the AIX C/S monitor provides direct information about the underlying C/S processes. As the AIX C/S environment supports full C/S, all the processing and programs can be run at the client. Information about the various stages of processing and programs is displayed. In contrast with the user applications the database information is integrated in the monitor window. The reason is to have all the information displayed in just one screen.

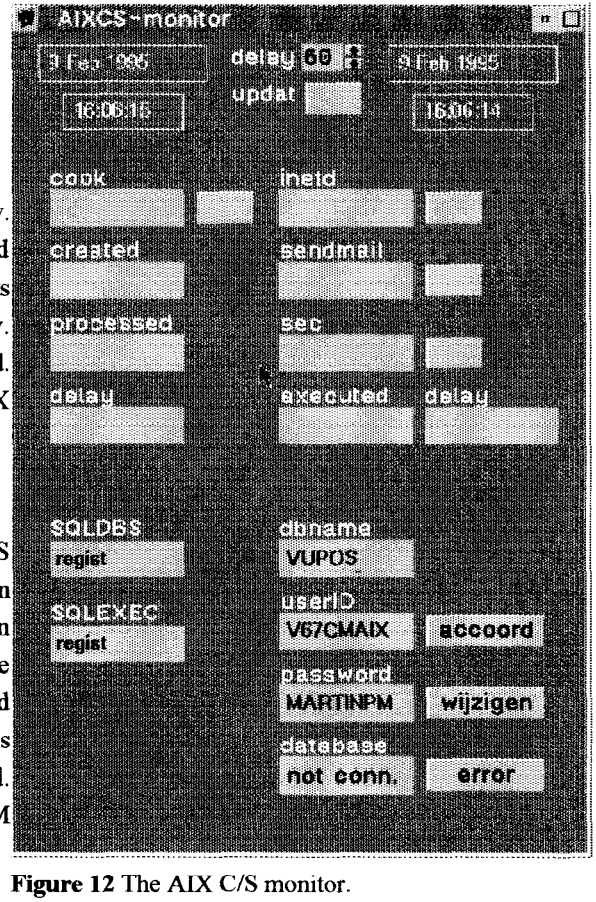

Figure 12 The AIX C/S monitor.

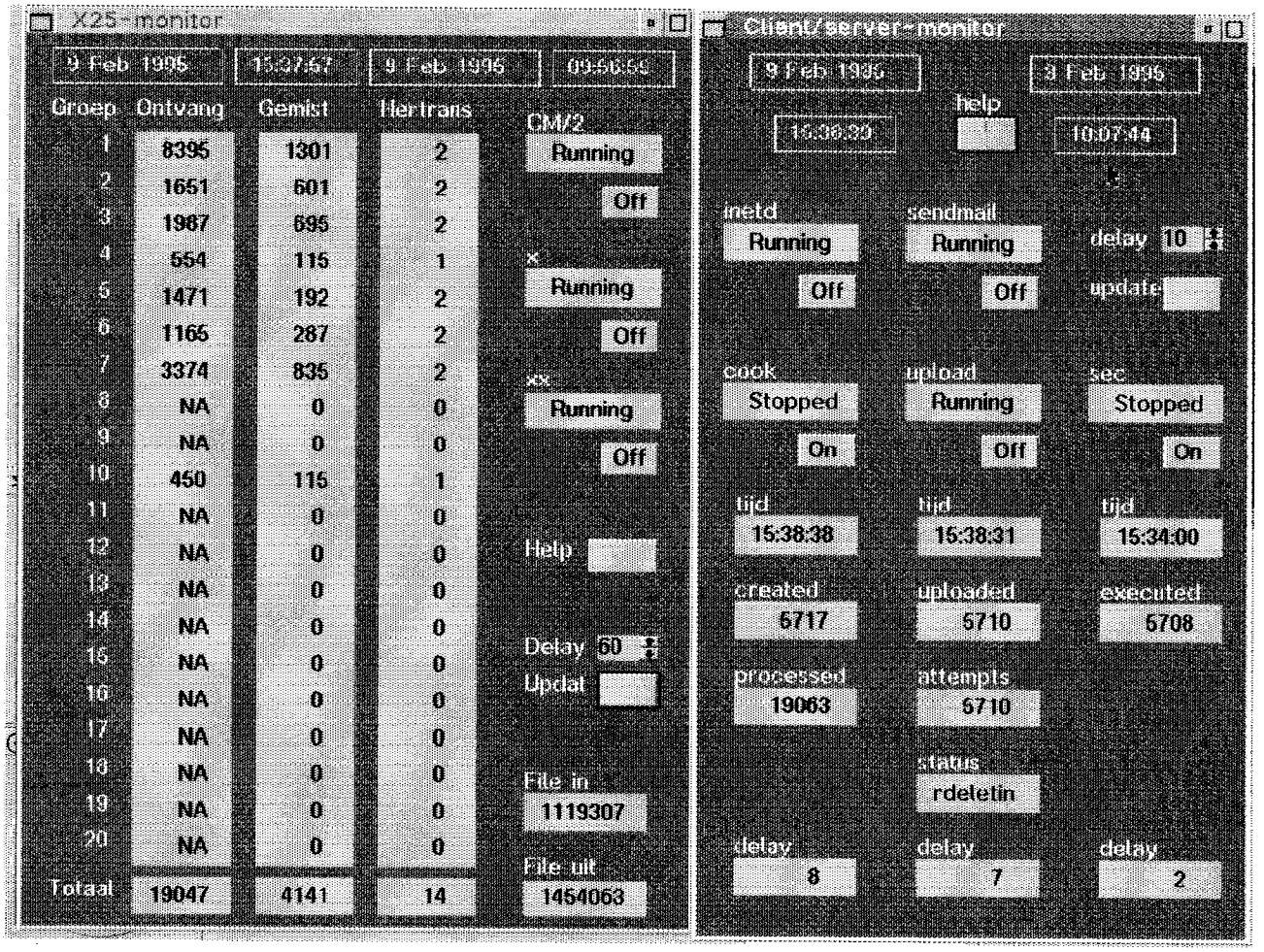

Figure 13 A full screen display of both the X.25 monitor (left) and the VM C/S monitor (right).

Finally, the difference between the packets uploaded (VM C/S) and packets executed (VM C/S) gives the third delay. In figure 13 both the X.25 monitor and VM C/S monitor are displayed.

5.5 Other graphical user-interface features

One of the issues that was ignored for a relatively long time, because the development of the C/S environments did have so many interesting research, design, implementation and usage issues that the project team almost forgot about a neat menu structure. Application of the menu structure is simple because much of the preparation for it has already been done. First, the framework for application development was designed long before. Secondly, database control and maintenance can be well-defined. Thirdly, every application program is characterized by some general functions which can easily be specified. Fourthly, the information generated can be visualized in a limited number of formats: on the computer display, on paper and on magnetic and optical media. The general menu structure to be designed should be a shell in which every application developed can have a place. Then based on user and program authorization, users can or cannot select the specific application program. The general purpose menu structure is shown in figure 14.

5.6 VX-REXX specific objects

Of the many special objects, the timer object is a very convenient one, especially if one designs and implements autonomous monitors. For control purposes, one has to have feedback frequently, and if necessary one has to maintain the C/S environment dynamically. Using timer objects one can frequently poll the environment (exception management) and retrieve information

(status files) to be displayed.

6 Future strategy of the C/S environments

6.1 Extending the matrix framework with (REXX) application programs

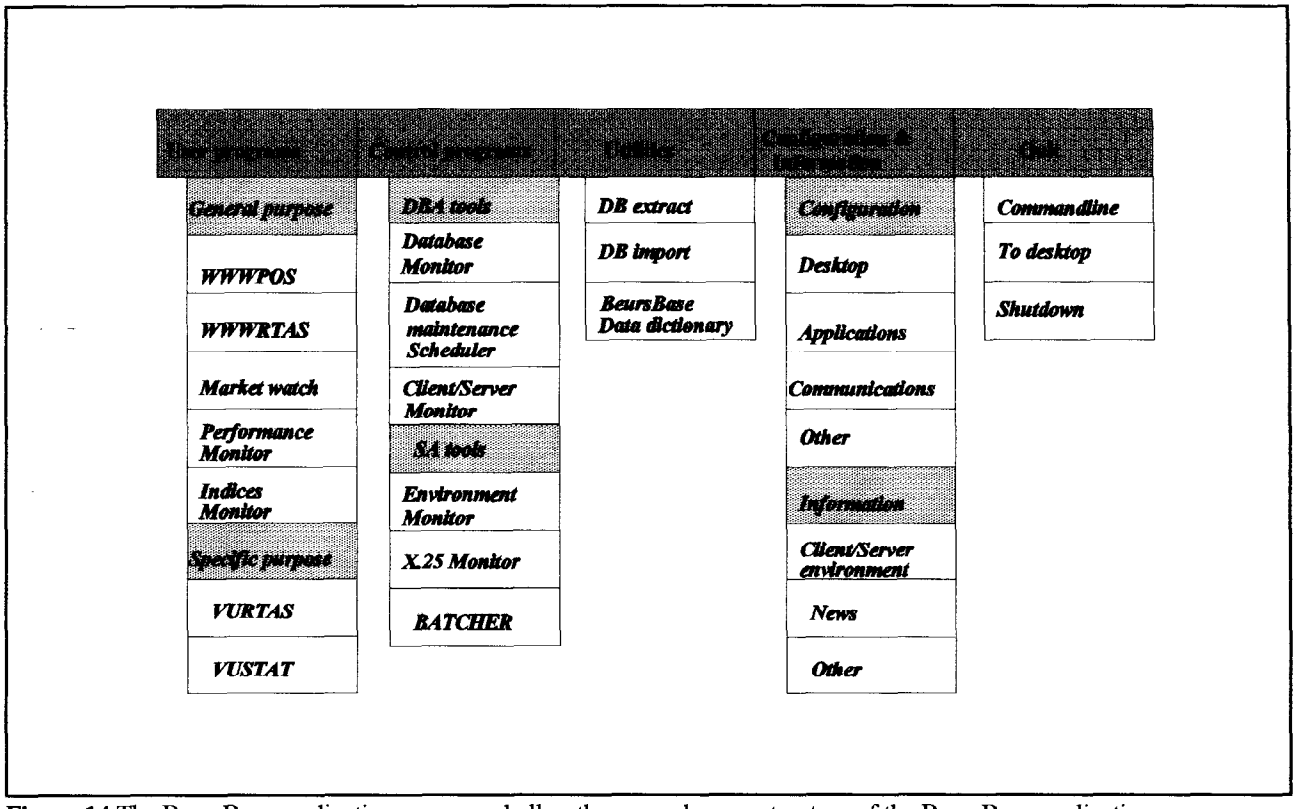

Figure 14 The BeursBase application program shell or the general menu structure of the BeursBase application programs.

VUPOS itself being ported from the mainframe to the personal computer. In contrast with the Cross Systems Product mainframe version, the personal computer version, implemented in VX-REXX is provided with a graphical interface. Compared with mainframe graphics using GDDM (CSP-ADMCHART interface), the VX-REXX graphical user interface is much more sophisticated. Though the most important advantage using a GUI painter is the significant reduction of development time. Especially for students it is very convenient to be able to generate usable output in a short time. In figure 15 an example of a simple XY-chart is shown. This XY-chart displays stock price development on December the first 1994 of the largest multinational of the Netherlands at the Amsterdam Stock Exchange known as '*Koninklijke Nederlandse Petroleum Maatschappij*' or '*Royal Dutch*'. To the general

public the multinational is known as Shell. A minimum of additional information is provided, like ASEs trading period ((9.30am to 4.30pm), the number of transactions found (159), and both maximum (188.60) and minimum (190.70) prices of the day. It still is an art and a science area to build good information displays. It is simple to expand the above example to a chart which shows, for example, the development of the portfolio of a user over a longer period of time.

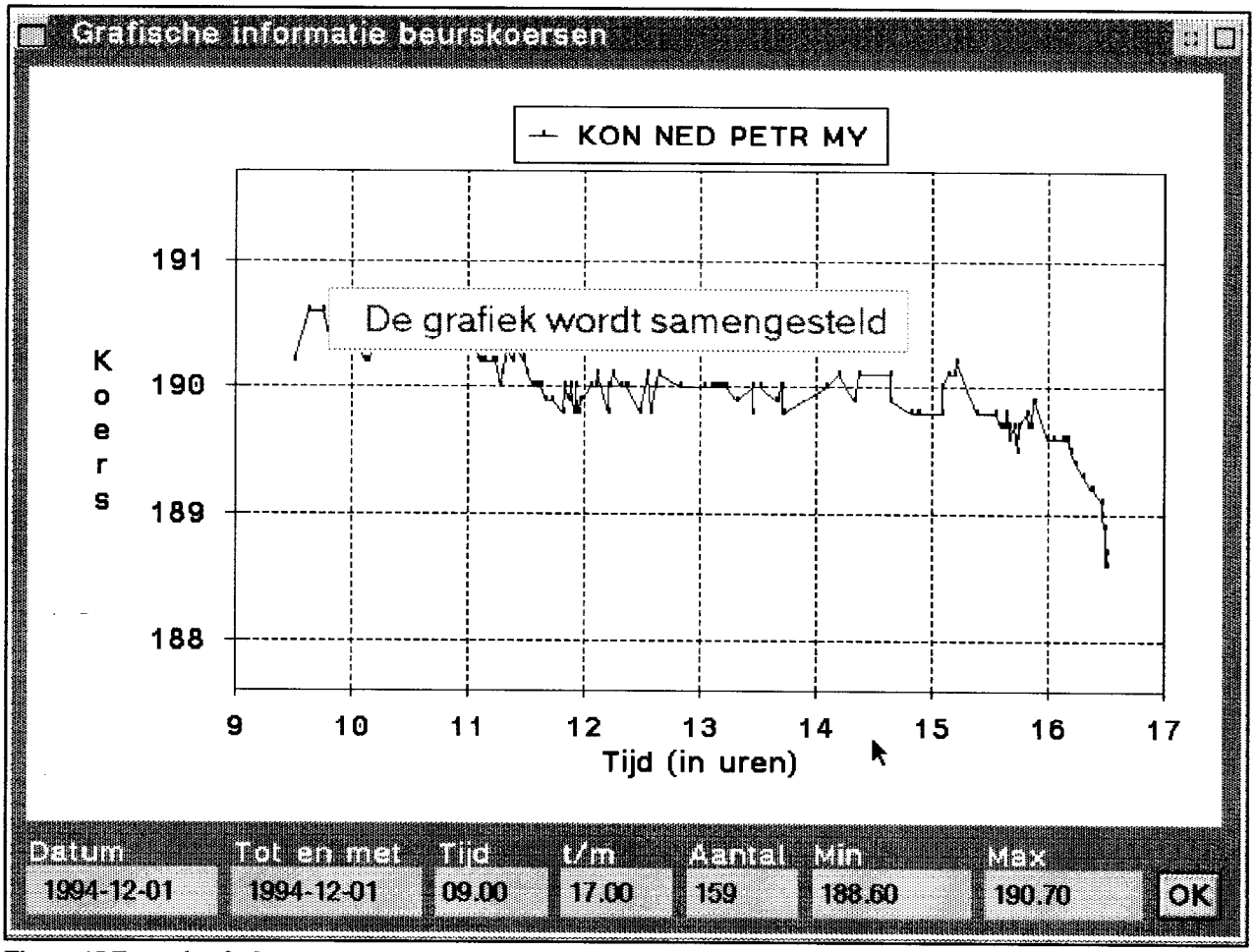

Figure 15 Example of a VUPOS screen.

6.2 Designing user (DBA) programs for database monitoring and control

As the number of users is increasing rapidly as well as the set of user application programs, the need for control and maintenance structures emerges. Though kept in mind this was set at a low priority. The following kinds of control are desired:

Authorization control: users and application programs

There are several possibilities of how to deal with multiple users and multiple programs. First, in C/S environments authorization can be placed at the user level. In this case every user obtains a userid. For maintenance purposes not very attractive, since it entails much administration (users) and a lot of authorization (programs). Secondly, as an alternative authorization can be placed at the program level. In that case every program is provided with a userid, or better, program id. Now administration

is reduced and authorization is simplified tremendously. It should be clear that in this situation program users do not have direct access to the database directly. In practice, a combination of userids and program ids is used. This requires sophisticated control. Up to now this has been done manually. An appropriate tool is under development.

Database control: database performance and tuning

BeursBase grows on a real-time basis. At the one hand Exchange data is inserted continuously, and at the other hand users and user programs add data as well. The largest tables in BeursBase contain hunderds of thousands of tuples and will rapidly grow up to tens of millions. It is not the physical size of the data that limits database operations, the gigabytes range will not be reached for years, but the performance and optimization issues. Reorganizing dbspaces, tables and indexes form a burden on database operations and database efficiency. This makes BeursBase so interesting for IS research. Therefore developments for several applications for performance measurement and database tuning have been taken.

Database maintenance

Usually database management systems are not provided with user friendly maintenance programs. Especially for BeursBase, database maintenance issues are of vital importance. Though there are lots of ideas on this topic, the have not yet been implemented in practice.

6.3 Improving the X.25 data link

A point of weakness in the BeursBase project remains the X.25 data link, because only one single communication is in use. In order to minimize the risk of data loss a secondary X.25 data link is necessary. Figure 16 shows the ideal implementation. Because ASE has two separate PTT Telecom (the national telecom operator) nodes. Therefore FEWEC can be provided with two separate X.25 links. Although the SARA link is implemented with a single line, no data will be lost when this communication line is goes down. For almost 100 percent reliable X.25 data link with the ASE, the following actions have to be taken.

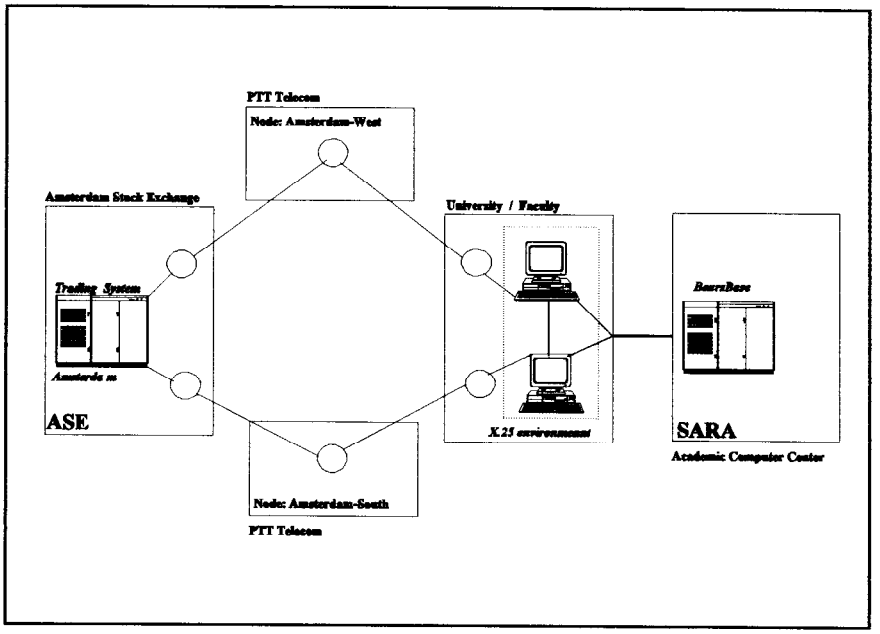

Figure 16 Two independent X.25 environments.

- Configuration of a shadow or backup X.25 connection, consisting of:
  - a second PS/2 with an X.25 Coprocessor expansion card;
  - a second ASE communication link, physically independent from the first one. This means that this X.25 link uses different nodes. At the ASE multiple independent nodes are available (Figure 16);
  - between FEWEC and the ASE, a second 19.2 Kbps PTT leased line;
  - An uninterruptable power supply (UPS) for the two X.25 PS/2s;

- A REXX-based control structure, as applied on FEWEC servers will be installed on both X.25 PS/2. This intelligent mechanism, a two PS/2s based monitoring system, should take appropriate action if one or more connections are lost;

- A hardware mechanism will be constructed to be able to cold-boot the PS/2 remotely. If the responsible system operator is not at FEWEC s/he should be able to reboot the system by a remote connection. (If one cannot telnet to the PS/2, it should be possible by connection via one of the FEWEC servers). Using a cold-boot procedure suffices, as the systems can start necessary processes automatically.

In the first half year of 1995 these investments are planned. In addition, several application programs are needed to create autonomous control between X.25 PS/2s and the FEWEC LAN servers. Two identical REXX-based programs, running at both X.25 PS/2s, have to check each other operations (file sizes and contents of the status files). If needed the C/S data link to BeursBase has to be changed, from one PS/2 to the other, without any delay and appropriate action has to be taken to get both PS/2s in operation again. Another program, residing at the FEWEC LAN server to check frequently if both X.25 PS/2 are running. Again, if some exception is found, appropriate action has to be taken. If, for some reason , one or more systems do not response, say within 15 minutes or so, FEWECs computer support staff is notified by email or screen messages.

6.4 Moving to the third and higher Client/Server levels

Eventually the AIX C/S environment has to move to the third, fourth and even fifth C/S level. Why? During the development processes of our C/S environments we encountered many performance problems. The larger tables where used by many programs and even more users concurrently. At the same time these tables are maintained (inserted, updated, deleted) on a real-time basis, resulting in many (dead) lock situations.

Looking over and over at the data, it was decided to split them into several time categories, based on their anticipated usage (figure 17). The first category, real-time (today's) data, is stored in a separate table, as this data has the highest priority. The second category, this years data, is accumulated into a second table. This data too is used very frequently (its size spans obviously maximally a year). The third category contains data from yesterday to a year before. The fourth and last category is historical data and is actualized to the end of last year. For performance reasons the last two tables are stored in dbspaces without leaving space free and records are stored by stock by timestamp. The first two tables are stored in dbspaces with small data and index buffers. Every night except for the weekend, the following batch processing is done: the today table is emptied in this year's table, also, at the end of December the last year table is emptied into the history table. A more intelligent solution would be a dynamically calculated optimal database performance with a minimum of data redundancy. Such an system area application program will be developed this year.

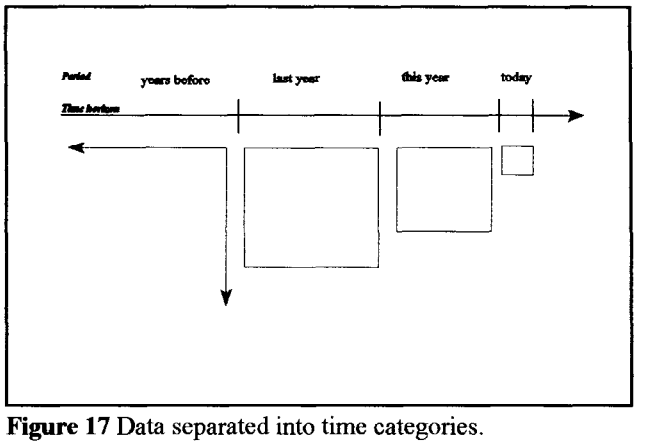

Figure 17 Data separated into time categories.

Moving to the third Client/Server level

Now for the C/S aspects one can imagine that the small real-time data table (today) used at a local database server would be significantly faster than a host database. So the next phase in the development of the AIX C/S environment is to establish a local database server for real-time data (IBMs DB2/2 for OS/2?). This will increase both application and database performance

for user applications like VUPOS tremendously. In addition, a local database server is more suitably equipped to support batch processing with the large database server as remote host.

Moving further to the fourth and fifth Client/Server levels

The third C/S level is relatively simple to reach, though should thoughtfully be implemented. User applications which need data for processing which is not available at the local server have to send requests to the remote database server. At least two problems, i.e. bottle necks are obvious: first if the amount of data is relatively large, database I/O is heavy and all data requested has to be send over the communication line which consumes much time. Secondly, local client processing is, compared to an AIX or VM host, though cheaper relatively slow. To overcome these bottlenecks, for some user applications it would be interesting to split up applications in several modules. Then these modules can be distributed across several systems, based on the CPU and database performance required. For example the portfolio simulation VUPOS or a technical analysis application has an econometrics/statistical feature which calculates several stock performance indicators over a certain time period (from actual real-time to historical data). Partially, the indicators requested are calculated at the client using the local database server (real-time) data, partially the indicators are calculated at the host using the host database server (historical) data. Finally the client program uses some algorithm to glue the partial outcomes into meaningful information. Even if the host requires real-time data (which is stored at the local database server) in total the results are generated much faster compared to the alternative in which the client does all the processing. Clearly this is another interesting area for research and education. The fourth and fifth C/S levels are within reach, though it will take at least two years before users can benefit from these ideas.

6.5 Speeding up REXX programs More extensive utilization of REXX compilers

Much can be written on the subject of REXX in terms of interpretation versus compilation. For the most part in the development of the C/S environments REXX is used by interpretation, except for some application programs which have been written in VX-REXX. At the point where speed is of primal importance, up to this moment the choice has been for programming in C. Using C as programming language implies using compiled programs. The main reasons for this choice are the flexibility of C, the clear X.25-C interface as well as the robustness and performance of C programs once they are compiled into executable code. An advantage in C too is the easiness with which multithreaded high priority programs can be developed. In addition there was no experience with the combination of X.25 and REXX or with REXX compilers which can generate fast executable code.

For manageability and maintenance reasons it would be wise to move from C to REXX. Currently research is conducted to take this step forward. Moving from C to REXX is necessary because the experience with C programming is and should be limited at FEWEC. The higher the programming, or more appropriate development, level the better. Thus one of the major concerns is to make more extensively use of REXX under the constraints that X.25 support, priority scheduling and compiling are supported and easy to implement.

6.6 Sophisticated application programs: a World Wide Web future?

Though a detailed discussion about user applications should not be included in this paper, one serious idea is worth mentioning, namely a portfolio simulation accessible through World Wide Web. As already discussed, the VUPOS portfolio simulation is being redesigned for the AIX C/S environment. This application will be used for all the user areas defined by several of the identified disciplines. For several reasons is the usage of VUPOS restricted to FEWEC members and students. First the application is relatively complex and one has to have detailed knowledge of the underlying processes. For students of economics and FEWEC members this should be no problem. If FEWEC intends to use VUPOS as a public relations tool and make it

available to others, for example high schools or other interesting populations, this will cause several difficulties. First, as said VUPOS is a complex program. Secondly, it takes several screens to manage ones portfolio. Thirdly, processing is done interactively which makes very large to huge scale application very costly and performance dramatically slow. Fourthly, a high-end OS/2 computer is highly recommended. Such systems are not so widespread in use.

Based on the rapid adoption of Internet, granted by the development of global information systems like World Wide Web, it is realistic to conclude that high schools and other interesting populations are able to connect to Internet without having to pay insurmountable costs. Thus a realistic alternative would be a stripped version of VUPOS, reduced to three to five screens, supporting batch processing once or twice every day. This simplified or stripped version of VUPOS should be provided with a World Wide Web interface. Installed at the FEWEC WWW server anyone, or if FEWEC so wishes, a selected audience will be able to use the program in, for example, a large stock investment competition. The investment results of the competition as well as the system usage itself can be input for research and education.
Anyway FEWEC is currently developing in cooperation with the ASE a WWW service which includes a real-time graphical display of the AEX, the index of both Exchanges in The Netherlands. We consider this as a first step towards an Internet based simulation, because this WWW facility will be used to research several interfaces, forms and displays for such a simulation.

6.7 Comparative application development

Once one has an AIX C/S environment as described in this paper, it is very interesting to examine and compare various development environments are available for C/S application development. Today numerous development environments available. This year we will use for example APL2/2 and VisualAge (both IBM), and CASE environments like ADW from KnowledgeWare and IEF from Texas Instruments/James Martin. Probably programming languages like C, C++, Smalltalk (via VisualAge) and COBOL will be tested too with respect to the trade-off (compared to the 4GL development environments) between an increased development time and run-time performance.

7 General conclusions and recommendations

7.1 Advantages of using REXX in a Client/Server environment

The project team of FEWEC concludes, based on the experiences so far, that the REXX programming language is, in particular in combination with the interfaces discussed in this paper, suited to develop not only C/S programs but a C/S environment itself too. Even without having much experience in developing C/S environments, implementing a REXX-based C/S was relatively simple. The project team benefitted from REXXs flexibility and portability as an AIX C/S environment was easily developed using code from the VM C/S environment.
Especially in the design and implementation stages it is advantageous to have an interpreted language in stead of having to compile code every time changes have been made to it. Finally, it has been proved that REXX user application programs can be gorgeous from the outside, efficient from within, and effectively fast in general without being to complex. This can hardly be said of programs developed in for example C or C++. Therefore in the IS curriculum of FEWEC REXX is preferred over C or C++.

7.2 Problems using REXX in a Client/Server environment

To temper the over enthusiastic mood of the project team, some problems or vague issues have to be dealt with too. Most of the problems encountered are probably heard before, but one cannot overemphasize their importance. Some REXX interfaces lack sufficient documentation, examples and example code. In the view of the project team every time the interface is used, the wheel is re-invented. Though there are for example many good REXX books available, most of them don't go beyond the introductory level. Also, voluminous manuals are not necessarily of high quality. Especially the REXX FTP, REXX SQL and REXX X.25 interfaces can gain popularity when documentation is extended.

Secondly, problems occur when one needs to share files between REXX programs in OS/2. Files in OS/2 REXX are used exclusively thus they cannot be shared among application programs. A solution was found in developing a C-based DLL function, as we found that it was possible to share the file this way. Using the C-function, OS/2 still results in DOS read errors, although it works fine. If one lacks the specific experience of writing DLLs, it is a burden to find out how things are being done. Ideally, a REXX compiler should be equipped with a tool enabling developer (student), without low-level programming, to put functions to be shared in a DLL.

Thirdly, more effort should be put in REXX benchmarking and tool evaluation. The project team encountered difficulties in choosing the 'right' REXX development tool and interface. Not usage of a REXX development was the problem, but which environment should be used. Fortunately, talking in hindsight acceptable choices were made. Fourthly, sharing information using status files is not the most sophisticated way of communication. However, no neat alternative was available. Especially the issue of file corruption and system crashes makes the usage of files in interprogram communication volatile.

7.3 Overall conclusions

During the last two years development of the VM and AIX C/S environments was successful. Both systems are operational and future plans and strategy promise to generate a massively used AIX C/S environment, high quality applications and a sound scientific environment for research, educational and public relation purposes. Though the power of REXX has already been proved in the current environment, as the AIX C/S environment moves further towards higher levels of C/S, REXXs position as an advanced C/S development language and environment shall be indisputable.

7.4 Recommendations

Finally, based on the REXX experiences, the project team comes with a few practical recommendations for developers, manufacturers and users as well. First, everyone wants better application performance. Though some other issues are of equal importance. One of the severe limitations of OS/2 REXX is that files cannot be shared among application programs. Working around the problem is not the preferred solution. In an advanced environment as OS/2, file sharing between OS/2 REXX application programs cannot be missed. Secondly, REXX still lacks some sophisticated interface toolboxes, especially for general and C/S specific monitoring. Thirdly, however fully accepted in CASE development, REXX code re-usability, re-engineering and a central repository, are far from reality. This definitely has to change if REXX is to be used in large scale application development. Fourthly, the basis for REXX-based GUIs has been set a few years ago, REXX development tools can be extended with more specific and more appropriate GUI features. Finally, as a fifth recommendation one should develop more detailed REXX programming handbooks with realistic examples.

8 References

[ASE94] Amsterdam Stock Exchange (1994), ***Amsterdam Real-Time Market Information System (ARTEMIS): user manual***, version 3.0 (in Dutch), Beursdata B.V., Amsterdam.

[Buys93] Buys, E.O (1992), ***TRANSPAS:Tthe Next Generation*** (in dutch), Graduation project, Free Univeristy, Amsterdam.

[Cowl84] Cowlishaw, M. (1984), ***The design of the REXX Language***, IBM Systems Journal, vol. 23, no. 4, pp. 326-335, IBM, New Jersey.

[Cowl90] Cowlishaw, M. (1990), ***The REXX Language***, 2nd edition, Prentice Hall International, Englewood Cliffs, New Jersey.

[Date92] Date, C.J. (1992), ***An introduction to Database Systems***, volume I, 6th edition, Addison-Wesley Publishing Company, Reading, Massachusetts.

[Davi85] Davis G.B., M.H. Olsen (1985), ***Management Information Systems***, McGraw-Hill Book Company, New York.

[Deit90] Deitel, H. (1990), ***Operating Systems***, 2nd edition, Addison-Wesley Publishing Company, Reading, Massachusetts.

[Deit92] Deitel, H., M.S. Kogan (1992), ***The design of OS/2***, Addison-Wesley Publishing Company, Reading, Massachusetts.

[Enge95] Engel, J.P. (1995), *ARTEMIS: the Amsterdam Stock Exchange and electronic information services* (in dutch), Graduation project, Vrije Universiteit, Amsterdam.

[Germ94] H. German (1994), ***The REXX handbook: BASICS, APPLICATIONS and TIPS***, Van Nostrand Reinhold, New York.

[Güse95] Güse, L.W.M. (1995), ***BeursBase project: operation's manual*** (in dutch, in preparation), Free University, Amsterdam.

[Horn90] Horne, J.C. Van (1990), ***Financial Management and Policy***, 8th edition, Prentice Hall International, Englewood Cliffs, New Jersey.

[IBM92] IBM (1992), ***VM/System Product Interpreter SQL/Data System Interface, program description/operations manual***, New York.

[IBM93] IBM (1993), ***OS/2 REXX : From Bark to Byte,*** IBM International Technical Support Centers, Boca Raton Center.

[IBM94] IBM (1994), ***Client Application Enabler/2, User's Guide*** version 1.2, IBM Canada Ltd. Laboratory: Information Development, North York, Ontario.

[Miss94a] Misseyer, M.P. (1994), ***From Stockdata to real-time Exchange Information***, in: Landelijk BIK blad, vol. 1, no. 1, pp. 25-27, Amsterdam.

[Miss95] Misseyer, M.P. et. al. (1995), ***TRANSPAS is death: Long live VUPOS and BeursBase***, Project proposal of VUPOS and BeursBase, Vrije Universiteit, Amsterdam.

[Mors94] Morsink, A.W. (1994), *Receiving, transforming, converting and adding real-time Amsterdam Stock Exchange data to BeursBase* (in dutch), Graduation project, Vrije Universiteit, Amsterdam.

[Orfa93] Orfali, R., D. Harkey (1993), ***Client/Server programming with OS/2 2.1***, 3rd edition, Van Nostrand reinhold, New York.

[Rudd94] Rudd, A.S. (1994), ***Application Development using OS/2 REXX***, Wiley-QED.

[Stal90] Stallings, W. (1990), *Business Data Communications*, MacMillan Publishing Company, New York.

[Tops94] Tops, I. (1994), *The Design and Implementation of a Real-Time Stock Exchange Simulation and Performance Monitoring System* (in dutch), Graduation project, Vrije Universiteit, Amsterdam.

[Turb95] Turban, E.F. (1995), ***Decision Support Systems and Expert Systems***, 4th edition, Prentice-Hall International, Englewood Cliffs, New Jersey.