Investigation into the Design of a Remote Maintenance System for Clinical Analysers

A thesis submitted to Dublin City University
for a Master's Degree in Computer Science

June, 2001

E. G. Maher, Dip. E. E., BSc. (Eng)

Control Systems and Electrical Engineering

Dublin Institute of Technology

Supervisor : Frank Duignan

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Master's Degree, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed : Eamon Maher

Eamon Maher

Date : June 2001

Acknowledgements

The author is greatly indebted to all the people who assisted with the research described in this thesis, particularly Mr. Frank Duignan who supervised the project. Dr. Jonathan Fisher, as head of the Department of Control Systems and Electrical Engineering in Kevin Street (DIT), merits recognition for gathering together the department's many talented professionals who were always eager to dispense advice, support and encouragement whilst providing ready access to necessary equipment and computer facilities.

The author is grateful to the Dublin Institute of Technology (DIT) for providing the funding which made it feasible to undertake this research

The personnel of the clinical Pathology Laboratory of Saint James's Hospital in Dublin were indispensable as a valuable source of knowledge about the medical environment and its stringent requirements. Peter Gaffney was particularly key and was always forthcoming with his extensive clinical and technical knowledge.

Support and distraction were cheerfully dispensed by the hapless souls in the Industrial Control Centre (Ciaran, Damon, Dominic, Rory, Sean) and the author's roommate (Raymond Rochford). A special mention goes to Sinead who brought happiness and light to the darkest depths of research.

Finally, the author wishes to dedicate this thesis to his family who ceaselessly provided support throughout his extended academic career.

Abstract

"Investigation into the Design of a Remote Maintenance System for Clinical Analysers"

E. G. Maher, Dip. E. E., BSc. (Eng)

This is a thesis study of the design considerations involved in interfacing medical analysers to computer systems for remote monitoring.

Most medical analysers provide a serial port connection for interfacing to a computer. Due to the limitations of this physical link, it is necessary for the interfacing computer to be within a radius of several metres. This computer can perform all the data monitoring and processing of the medical analyser. Monitoring the operational performance of a medical analyser can assist in maintenance programs. If it relays relevant data to a remote computer, a powerful remote maintenance and monitoring system can be developed when the remote monitoring computer collects data from a number of remote medical analysers.

To satisfy the demands of remote maintenance and monitoring the Transmission Control Protocol/Internet Protocol (TCP/IP) network protocol was investigated. The existing worldwide base of support for this protocol on numerous platforms and its universal addressing scheme made it the preferred choice for "openness" concerns. The relative merits of User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) were evaluated and the nature of the reliable stream service offered by TCP was selected as more appropriate to the medical environment where data loss is unacceptable.

A Client/Server architecture was investigated with a central server and remote clients which were connected to clinical analysers. The monitoring computer local to an analyser connects to the server using the TCP/IP network protocol. This Client-Server configuration is particularly suited to the distributed nature of medical laboratories where instruments are typically not centralised and the lack of availability of powerful computers makes it necessary to resort to using simple computers locally to the instrument to relay data to the more powerful server computer.

The work was carried out with the co-operation of the Central Pathology Laboratory of Saint James's hospital in Dublin.

Table of Contents

Glossary of Acronyms

The following is a list of acronyms used throughout this thesis:

AII	Advanced Instrument Interface
ANSI	American National Standards Institute
API	Application Programming Interface
CCITT	Comité Consultatif Internationale de Télégraphique et Téléphonique (International Telegraph and Telephone Consultative Committee)
CEN	Comité European de Normalisation (European Committee for Standardisation)
CTI	Computer-Telephony Integration
DARPA	Defence Advanced Research Projects Agency
DBMS	DataBase Management System
DLL	Dynamic Link Library
GP	General Practitioner
GUI	Graphical User Interface
HIS	Hospital Information System
ICMP	Internet Control Message Protocol
ICU	Intensive Care Unit
IGMP	Internet Group Message Protocol
IP	Internet Protocol
ISO	International Standards Organisation
IT	Information Technology
LAN	Local Area Network
LIS	Laboratory Information System
LTQC	Long Term Quality Control
MIC	Module Identification Code
MNP	Microcom Networking Protocol
MOM	Message-Oriented Middleware
OSI	Open Systems Interconnect
PPP	Point-to-Point Protocol
QA	Quality Assurance
RFC	Request For Comment
SLIP	Serial Line Interface Protocol
SQL	Structured Query Language
SRL	Service Request List
TAPI	Telephony API
TCP	Transport Control Protocol
TIN	Transaction Identification Number
UDP	User Datagram Protocol

1. Introduction

The medical world is one of the few remaining Information Technology-starved domains. Whereas industry and commerce have embraced the advantages of computer analysis as a competitive tool and distributed computing as an empowering technology for more efficient work methodologies, the medical world still relies predominantly on paper trails of information and documentation. The exploitation of Information Technology in the medical domain presents substantial opportunities and is receiving considerable attention from a number of sources worldwide.

Any initiatives for the adoption of Information Technology as applied to the medical domain have tended to be sporadic and on a small scale due to limited funding. To date, Information Technology has not been viewed as a critical component of the healthcare environment, and is subject to the budgetary priorities of various ministers and governments [ALLE – 91][BAKK – 88].

The perception of Information Technology as applied to the medical domain (medical infomatics) as one of high-tech indulgence limited to over-funded departments has changed to one of competitive advantage offering cost-effective benefits and is indeed now seen as a critical factor to survival. As the global population increases and the number of people who require extra medical care also increases due to ageing demographics, efficient and cost-effective delivery of healthcare services has focused attention on the costs associated with medical services.

Due to the advances have been made in all aspects of healthcare, the resulting information created (as well as the increased complexity of interacting data between the various diverse components of an integrated healthcare service) has produced an explosion in the volume of data to be collected, processed, correlated and stored. However the production of masses of data does not necessarily mean that useful information is automatically available [CONN – 80]. Any effective data management system must collect, collate and distribute data from numerous

sources in the healthcare environment. These sources are as diverse as surgical, pharmacological, laboratory, transportation, administration, billing, and combine to deliver an integrated healthcare service. They all have common elements but also numerous differences which complicate their smooth interaction from an information exchange point of view.

Soaring healthcare costs combined with the additional expense of managing such huge volumes of data have forced healthcare service providers to recognise the need to optimise their delivery of care and improve the efficiency and effectiveness of the service. Traditional paper-based manual record/data management systems are incapable of administering the volume of data which is currently utilised by healthcare systems and will balk under future loading predictions. As in many other industries, computerisation offers great potential for a cost-effective solution to the integration of the healthcare environment and its rigorous data management demands.

Some progress has been made in the application of information technology techniques to certain areas of patient care. Legacy database systems are now growing in volume with records of demographic details for the administration of patients. Medical imaging is regularly digitised and stored on some form of computer archival and retrieval system. Clinical laboratories use increasingly sophisticated instrumentation which is often interfaced with local computer systems to form part of a laboratory-wide computing environment.

However these isolated implementations of Information Technology are rarely integrated completely with other Laboratory Information Systems (LIS's) or Hospital Information Systems (HIS's) and result in hybrid data management systems which by necessity cling to the traditional documentation and reporting methods whilst struggling to embrace the waves of advancing technology.

Standardisation bodies such as the American Society for Testing and Materials (ASTM) and the European Committee for Standardisation (Comité European de Normalisation, CEN) are currently working towards developing standards for the

communication of data throughout several sections of integrated healthcare systems. However, this process is ongoing and it is expected to take several years before a definitive standard is universally accepted and even longer before it is implemented on a global scale.

There is a very clear and immediate need for opening the disparate computer-enabled medical environments and their related generated data to a wider audience. For example, if the extensive data generated in Laboratory Information Systems were to be correlated with the wealth of demographic and historical data that is available through administrative data sources, then the improved healthcare delivery would be substantial. Diagnoses would be more reliable and more rapid as environmental factors due to the patient's living situation become integrated with the physician's decision analysis.

In an attempt to realise this goal, this thesis has addressed the need to develop an architecture that facilitates the cross-integration of disparate data sources. Specifically, this thesis focuses on opening the laboratory-wide computing environment to a wider audience, as one of many interim solutions while awaiting the arrival of an embracing set of standards.

The design has utilised many existing technologies in an attempt to leverage the substantial investment in those technologies. Serial communications have been used to interface to instruments, Structured Query Language (SQL) to enable data storage and retrieval in databases, and the predominant Internet protocol, TCP/IP, to provide remote data communications. These technologies have become the de facto standards in their respective fields and have been in widespread use for numerous years. Such established and mature implementations supply robust foundations for application development without the unknowns and quirks of new cutting-edge methodologies. This is especially important in the medical domain where fault intolerance is essential.

The basic layout of this thesis follows an analysis-design-implementation format. Following upon this introductory chapter, chapter two introduces the

medical domain in general and the clinical laboratory in particular. It imparts an appreciation of the environment under analysis and highlights some of the related problems and difficulties.

The third chapter presents the specific requirements which drive the resolution of the design concept. These requirements are addressed in relation to the various categories of users who impact on the scope of this research.

The fourth chapter proposes an architecture to satisfy the preceding user requirements and is referred to as the design concept. It also discusses the components necessary to fulfil the design concept and how they interact.

The ensuing three chapters divulge the specifics of the implementation of the components. The three basic components are the interface to the instrument, the transmission of data throughout the system (particularly the storage and retrieval of data obtained from the instrument), and remote connectivity for distributed processing.

The final chapter brings all the components together again and critiques the efficacy of the total design solution with respect to satisfying the user requirements as they were described in the third chapter.

2. The Environment

2.1. Hospital Information Systems (HIS)

The goal of a Hospital Information System (HIS) is to integrate all information processing aspects of the hospital from numerous sources [BALL – 91]. However, the integration of data from such diverse sources is not a trivial matter and the resulting engineering solution is always complex.

Numerous areas of the medical domain have local data management systems. An accounts department could have a billing system integrated with a patient demographic database over a Local Area Network (LAN) within the administrative offices. Stores and purchasing departments may have an inventory and warehousing system for efficient space and stock management. Whilst laboratories may have Laboratory Information Systems (LIS) to support the processing of samples and the corresponding analytical results.

One means of achieving the above goal which is currently being studied by the medical informatics community ([HL7 – 90], [ASTM – 90], [DEMO – 92]) is through the utilisation of a computer-based Medical Record for collecting data relating to an individual patient from all medical areas. When a generic standard for the computer-based Medical Record is agreed upon, it will accommodate the meaningful exchange of medical information between healthcare service providers and their supporting agencies throughout the world [BAKK – 88][ALLE – 91]. This would result in consistent healthcare treatment given to a patient which is independent of wherever a patient may require medical services. The required medical history of the patient would be available at the point of treatment despite the possibility of the patient's primary healthcare service provider being on the opposite side of the globe.

There are several widely published protocols for the communication of data between hospital computer systems. The most common ones are Health Level 7 (HL7)[HL7 – 90], ASTM E1238-90 [ASTM – 90], and CEN ENV 1613 [DEMO

– 92]. As yet no protocol has emerged as the dominant de-facto standard or as an agreed standard and considerable effort is being exerted in those directions.

A typical HIS is shown in Figure 2.1. Such HIS's are usually composed of a vast selection of different systems, each with their own hardware and protocol dependencies. The cost of completely replacing these systems is prohibitive and so a HIS must merge the components into a coherently operating union. Different network technologies must inter-operate (twisted pair with coaxial; ethernet with token ring; TCP/IP with IPX; Novell NetWare with Microsoft Networks); operating systems must handle data and file formats from other operating systems (Windows and DOS with Dec Alpha and UNIX). Its operation tends not to be seamless and frequently is not complete. Integration compliancy requires proprietary turnkey solutions which are difficult and expensive to maintain.

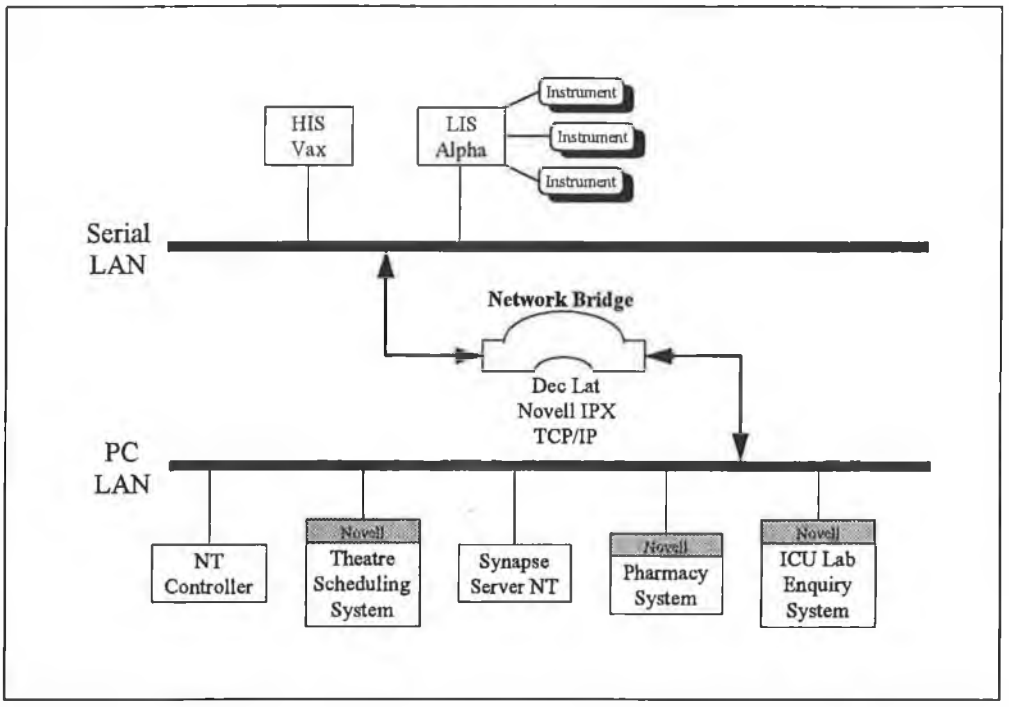


Figure 2.1 The diverse makeup of a typical Hospital Information System (HIS)

These monolithic legacy systems are based on the computing architecture model of mainframe computer systems. As such they are expensive to install and maintain, and lack the flexibility to be incorporated into the modern computing

paradigm of "downsizing" which tends to be the inevitable solution to the problem of soaring costs and competitive markets.

The component of the HIS which is pertinent to the research documented in this thesis is the LIS. In particular, clinical instruments and how they interface to the overall Information Technology (IT) structure.

2.2. Laboratory Information Systems (LIS)

Traditionally the majority of medical analysers were located within the confines of a hospital site and typically provided services for the hospital, local GP surgeries, community health clinics, and other local hospitals. It was also a regional centre for specific core competencies where specialised expertise and equipment were centralised due to financial and resource restrictions. Thus these specialised services would necessarily have been available to a very wide area.

As the coverage area for the provision of clinical analysis has grown, the volume of laboratory test requests has increased. Technological advances have also increased the variety of tests available to assist healthcare professionals. These increases in volume and variety have driven (and been driven by) the development of automated clinical analysers and this has resulted in the production of large volumes of data [KELL – 74][BENS – 80], at the rate of thousands of tests per hour.

In order to manage these large volumes of data, Laboratory Information Systems (LIS's) have been developed. Services regularly supported by a modern LIS include patient demographic and test request entry, the generation of work-lists, data capture from instruments, data manipulation, manual test entry, report validation, report generation, accounting facilities, and comprehensive database management. Sometimes these LIS's are incorporated into larger data management systems like Hospital Information Systems (HIS's). They can support the connection of several analysers to a LIS (Figure 2.2) and can even support multiple LIS's servicing one or more analysers (Figure 2.3).

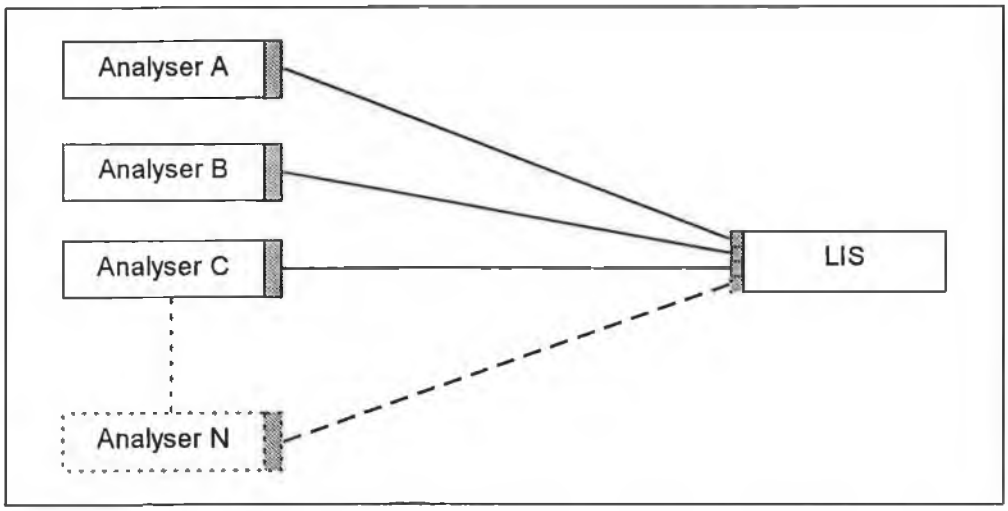


Figure 2.2 Multiple analysers connected to a Laboratory Information System (LIS)

However, the absence of a globally accepted analyser/computer interface standard necessitates the development of customised interface software solutions which are specific to each new analyser/LIS configuration. The American Standards for Testing Materials (ASTM) and the European Committee for Standardisation (CEN) are working on publishing a standard for the analyser/computer interface (ASTM 1394 and CEN/TC251 WG5), but that will take several years.

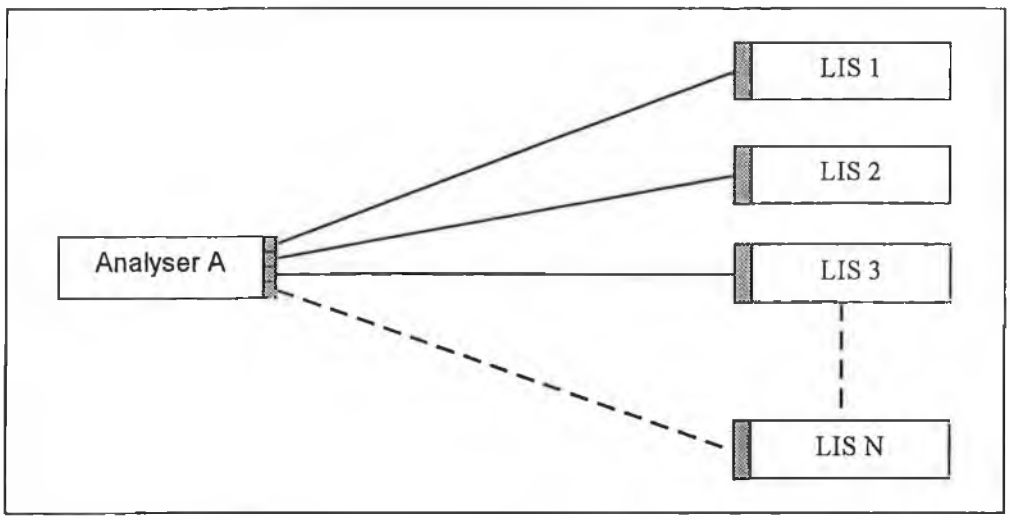


Figure 2.3 Multiple Laboratory Information Systems (LIS's) servicing one instrument

In the meantime more demands are being placed on current LIS's to process more medical requests at faster rates. Any expansion of a LIS to cope with these increasing demands tends to be labour and time intensive and further difficulties tend to occur owing to cumbersome licensing agreements (of a proprietary nature). Cost-effective solutions provided by third party developers are generally difficult if not impossible to implement due to licensing restrictions and unforeseen architectural design difficulties.

In an attempt to utilise the investment in these legacy systems, several proprietary systems have been produced in ad-hoc isolated developments which leverage the data assimilation and management features supported by the existing LIS's, while utilising new computer design concepts to fulfil specific local requirements [LEWA – 92][O'MO – 88][MOLL – 90]. The size and fragmentation of this market have resulted in isolated software solutions which in fact are solutions that adhere to few standards and exist as "stand-alone" solutions to problems shared by many laboratories (for example an order entry system for an Hitachi 717 mixed chemistry analyser [GAFF – 96]). So, whilst the problems are shared, the solutions are not turnkey and are generally difficult to transfer to other laboratory locations due to the lack of standards.

In order to assist with the management of the data administered by a LIS, it has been shown that computer-based Decision Support Systems (DSS's) can be used for data reduction and information enhancement [O'MO – 88]. With the incorporation of the knowledge of experts into DSS's it has been shown that DSS's have applications in many different areas of laboratory medicine [BENS – 80]. For example they have been shown to be of benefit in monitoring the performance of automated clinical chemistry analysers since Quality Assurance, maintenance, and fault diagnosis are areas ideally suited to the use of DSS's [GROT – 91].

Whilst many successful DSS's have been developed at a local level [SPAC – 87], they essentially tend to be third party proprietary solutions and suffer the

corresponding problems outlined above: they are specifically designed to satisfy local requirements and are often difficult to transfer to other locations and integrate with other data management systems due to the absence of an adopted data communication standards. They also tend to have an over-reliance on local personnel who were key to the implementation of the DSS, and consequently their continued performance depends on those key personnel - not a stable situation given the current employment trend of increasing mobility of employees within the medical infomatics industry. This inability to easily transfer DSS's between laboratories has limited their scope of implementation.

2.3. The Modern Medical Laboratory Environment

Just as the data administration overhead has increased throughout areas of the medical domain, clinical laboratories have had to accommodate a substantial increase in the volume of clinical services rendered.

The provision of a clinical analysis service is dependent on the ability of the laboratory to deliver accurate data within an acceptable time-frame from when it is requested. Instrument failure may have significant cost and patient health implications resulting from lost production, staff idleness and untimely sample reports. Lack of confidence in the ability of a laboratory to deliver reports on time may necessitate the healthcare service seeking a more reliable clinical analysis provider in order to deliver an acceptable healthcare service. This other clinical analysis provider would invariably be external to the hospital's sphere of direct influence and would introduce issues of regulation and Quality Assurance of the service. Additional indirect costs would be incurred due to the expensive clinical laboratory resources being under-utilised as urgent samples get sub-contracted to external laboratories. Therefore a clinical laboratory must ensure that it sustains its reputation of timely analysis by using instruments in an operational state and delivering accurate and timely reports on all sample analyses.

2.3.1. Operational Availability of Laboratory Instruments

The operational availability of an instrument is dependent on its reliability, maintainability, and staff training and experience. The reliability of an instrument is a measure of its ability to maintain the specified performance, whilst the maintainability is the ease of rectifying a failure situation [ARS – 80]. Technically competent staff may be able to sustain sufficient operational availability for an instrument which has low reliability but high maintainability, whereas less well trained staff may require an instrument with greater reliability to offset their lack of technical competence in order to deliver an equivalent level of operational availability. This would require more expensive instruments which deliver greater reliability, or an aggressive continuous training program for laboratory staff. The costs of training staff to a sufficiently high level of competency can become prohibitive if the rate of instrument replacement or acquisition is high, or if staff turnover is high.

An effective maintenance program can minimise the loss of operational availability. Maintenance is traditionally separated into the two areas of preventative and corrective maintenance. Preventative maintenance attempts to anticipate problems before they materialise by defining procedures to be performed at specific time intervals. The time intervals are calculated in accordance with component lifetimes and expected usage, and therefore can only accommodate anticipated component performance [ROBE – 80]. Corrective maintenance encompasses the procedures necessary to restore operational availability when a fault has occurred.

The costs incurred due to a maintenance program (for example downtime costs or component replacement costs) can be decreased if the preventative maintenance schedule is based on the actual status of components rather than the anticipated status. Thus component replacement costs would be incurred only when necessary rather than when dictated by a linear, inflexible preventative maintenance schedule based on predicted component lifetimes and expected usage.

The implementation of a maintenance program is defined in the Quality Assurance scheme being exercised in the laboratory. It is within the guidelines and procedures of the Quality Assurance scheme that the effectiveness of a maintenance program can be tuned.

2.3.2. Computer-Based Quality Assurance Schemes

An experienced operator may be able to intuitively assess the status of an instrument by visual inspection and by monitoring an instrument's status and report messages. However it can take a considerable amount of time and resources for an operator to become sufficiently acquainted with an instrument so that they can correctly interpret most situations that may arise and deal with them quickly. However, there also remains the possibility of the occurrence of a situation that even an experienced operator has not yet encountered, such as a low-level hardware failure for example, and may result in extended periods of inoperation.

Automating the intuitive deductions of an experienced operator is not an easy task but the benefits of such a system would include assisting inexperienced operators whenever unexpected situations arise as well as freeing resources which may be necessary to support inexperienced operators. Even experienced operators may not encounter all possible failure situations due to the complexity of modern instruments and to their improved reliability [MOLL – 90], and over time would also benefit from such a system. This is particularly relevant for a typical clinical laboratory which does not have staff dedicated to instrument maintenance and quality control duties. Instead, performing maintenance protocols and the logging of all maintenance procedures performed would thereby form only a small part of the responsibilities of clinical laboratory technicians.

This is unlike procedures in industries such as the Armed Forces, the Nuclear Energy and Aviation industries where maintenance is considered critical and strict documentation and control procedures are practised [HOLM – 92]. Computer-based systems for the control of maintenance scheduling [PETE – 89], corrective maintenance, and rapid and accurate fault diagnosis [CARR – 86][MALK – 86]

are in existence in those environments. Such comprehensive Quality Assurance schemes require the assimilation of data from numerous sources and their timely correlation and integration are only possible through the extensive use of data management facilities offered by computerisation. However these systems are targeted to the unique requirements of each industry and are therefore customised beyond transferability to the medical domain without substantial reengineering.

Whilst the exact procedures in such specialised areas are not appropriate to the requirements of the medical domain, the lessons learnt and the broad functionality of their Quality Assurance schemes can assist in the adoption of industrial-standard Quality Assurance schemes in the healthcare environment.

Maintenance scheduling based on the actual status of an instrument would be the most cost-effective approach for a maintenance program. Unnecessary preventative maintenance would be reduced because a correctly performing instrument would not require attention. As quality control methods indicate a deterioration or unexpected change in the performance of an instrument, preventative maintenance measures would be taken and, if necessary, corrective maintenance performed.

All medical laboratories have some form of manual Quality Assurance protocols to assist them in their assessment of the performance of the instrument in their care [GAFF – 96]. If the results of these Quality Assurance protocols are made available to a computer-based Decision Support System (DSS), the resultant automatic Quality Assurance scheme would be tirelessly more efficient than the equivalent manual human scheme where boredom and exhaustion can have a substantial impact on the effectiveness of the scheme.

2.4. Clinical Analysis

Clinical laboratory tests provide contributory or definitive diagnostic evidence for the discovery, confirmation, exclusion or management of disease states. While healthcare professionals request tests for many reasons, fear of litigation can be

reduced by having substantive clinical data to support a diagnosis and hence treatment. Since the accuracy of clinical data defines the validity of their use as diagnostic evidence, the quality of the data resulting from clinical analysis is of paramount importance for a clinical laboratory. The quality of the results produced and the continued operation of an instrument depends on a comprehensive Quality Assurance scheme, of which a comprehensive and effective maintenance program is an essential element.

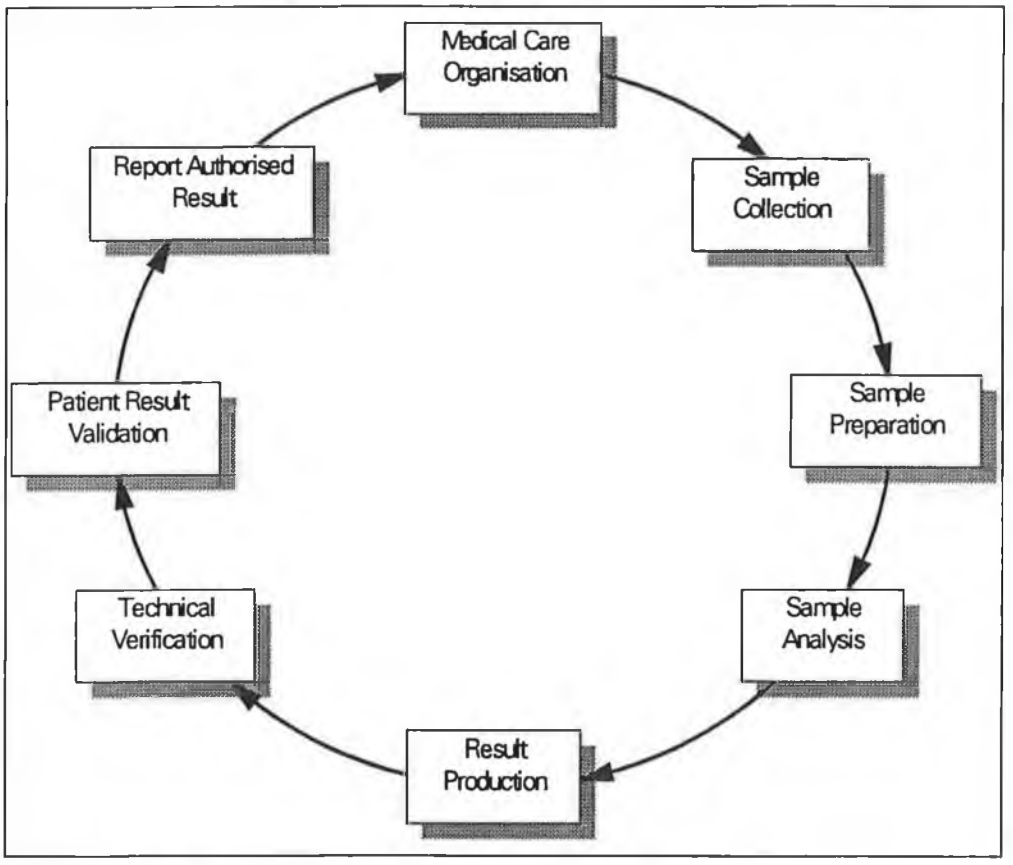


Figure 2.4 The process for clinical analysis

A complete clinical analysis iteration is a cyclic process (Figure 2.4) starting with the decision by a healthcare professional that a clinical analysis would assist the healthcare service delivery to a patient. A sample is obtained from the patient and analysed to produce a result which is reported to the healthcare professional. It can be simplified as a "brain-to-brain" process where each step in the loop is described as a generalised functional entity, as shown in Figure 2.5.

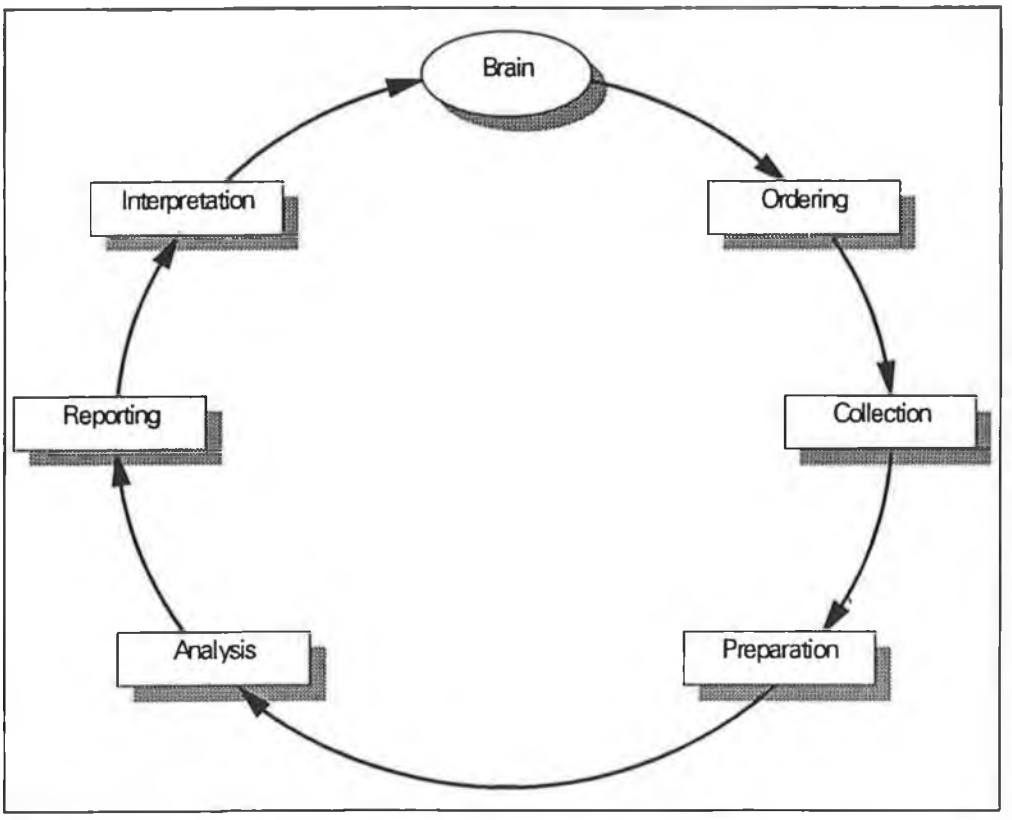


Figure 2.5 The brain-to-brain loop

Over half of all laboratory tests are used for the purpose of monitoring patients with a disease [BEEL – 87] and this is best achieved by selecting the test(s) most sensitive to change in the disease process since variations in the results for an individual are always much less than variations in the general population [WINK – 87]. Monitoring changes in consecutive results for a patient can lead to the derivation of the status or progress of the disease process. While changes in consecutive results can be due to biological variations, analytical or technical variations can also be the cause. The cause of the changes must be identified so that inconsistent results due to analytical or technical variations are not incorrectly attributed to biological variations. For large changes between consecutive results (greater than 2.8 standard deviations), it is 95% certain that they are not caused by analytic variations [GAFF – 96].

A comprehensive Quality Assurance scheme which includes technical validation will minimise inaccurate results due to analytic variations. Technical validation of results compares them with "Reference Range" values which correspond to the

spread of results found in 95% of a selected population group with a specific demographic profile. Sex, age, ethnic origin, even geographic location can influence the distribution of results for apparently healthy subjects. Because "Reference Range" values account for only 95% of the population group, statistical analysis shows that the possibility of a correct result being produced decreases with the number of tests performed (Table 2.1)[HOWA – 91]. This situation must be avoided.

Number of tests ordered	1	6	12	20	100
Probability of "normal" result	95%	75%	54%	36%	0.6%

Table 2.1 Probability of a healthy person having all test results within the "Reference Range" [HOWA – 91]

Analysis of results from testing known quality control material is used for assessing the Real Time Quality Control (RTQC) status. This enables the RTQC of patient results to be evaluated and directly indicates whether variations are attributable to analytic or technical sources.

The various Quality Assurance schemes utilised for technical validation are not effective in detecting gross errors such as mislabelled samples and data entry or transcription errors [CHAM – 86]. Gross errors can only be minimised by the provision of, and adherence to, specific operational protocols. These, by their very nature, tend to be tedious and repetitive. As fatigue or carelessness affects the human link in these protocols, the potential for gross errors increases. There is also the additional possibility of incorrectly validating a test result when it is buried under hundreds or thousands of technically valid results.

Whilst the processes utilised for evaluating the technical validity of data are typically well documented and may even be declared as Standard Operating Procedures (SOP's), it is very easy for their implementation to deviate from the defined protocols. As the volume of clinical tests being requested increases, it is necessary for the number of laboratories and their various complements of staffing levels to also increase. As more staff are required to be trained to competent and

proficient levels, the training pyramid grows deeper (Figure 2.6 Training Pyramid), with the manufacturer-defined operational protocols being spread over a wider operator base further from their original definition.

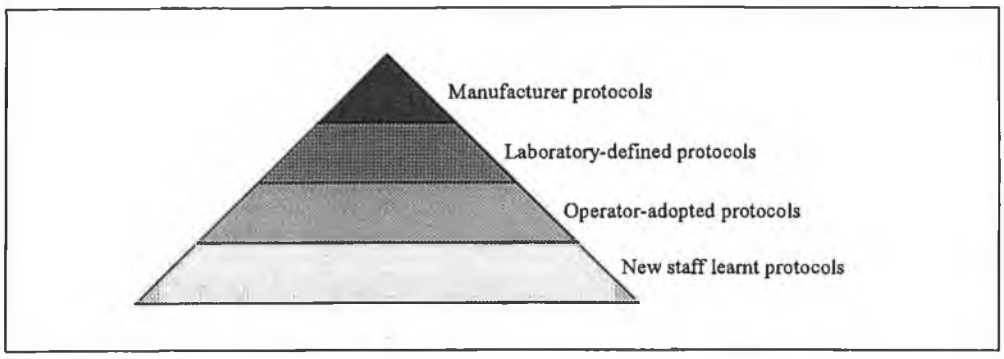


Figure 2.6 Training Pyramid

In the absence of certified training courses producing clinical instrument operators of an approximately constant competency, recently assimilated laboratory assistants are typically introduced to the SOP's by their immediate senior. This invariably results in an accumulative deviation from the operational protocols defined by the manufacturer for a specific instrument model. The complexity and diversity of modern clinical instruments being employed in laboratories compounds this problem. This can result in a degradation of the operational performance of an instrument and can impact on the definitive worth of data as diagnostic evidence.

2.5. Clinical Analysers

The vast majority of clinical analysis requires repeated iterations of relatively simple functional steps (Figure 2.7). Just as technological advances empowered increasing automation in diverse industrial processes, the clinical laboratory eventually underwent a similar revolution in sample processing and result production.

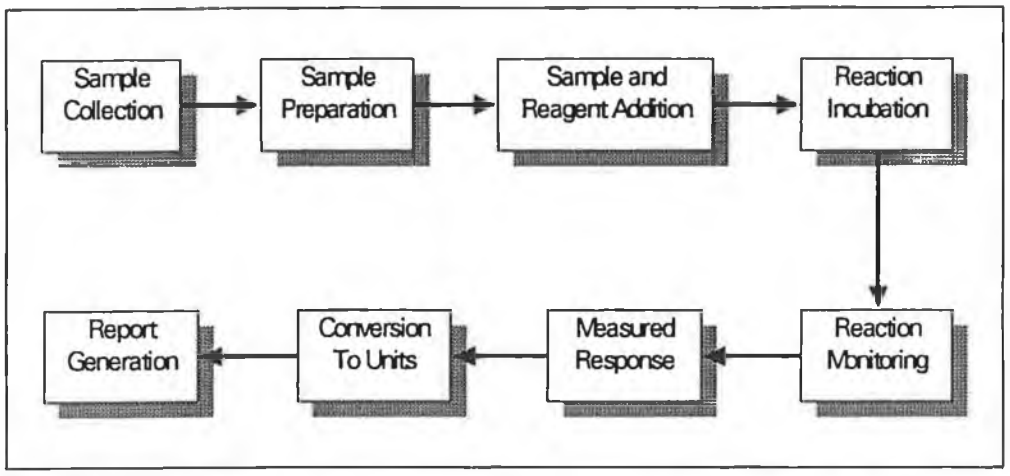


Figure 2.7 The functional steps involved in manual analysis of a sample

Initial attempts at automation concentrated on mimicking the actions required for manual methods. However, once the functional steps were identified at the process level and automation techniques applied to each function rather than to the physical activity necessary for the manual completion of the function, rapid progress was made in the development of automated clinical analysis (Figure 2.8).

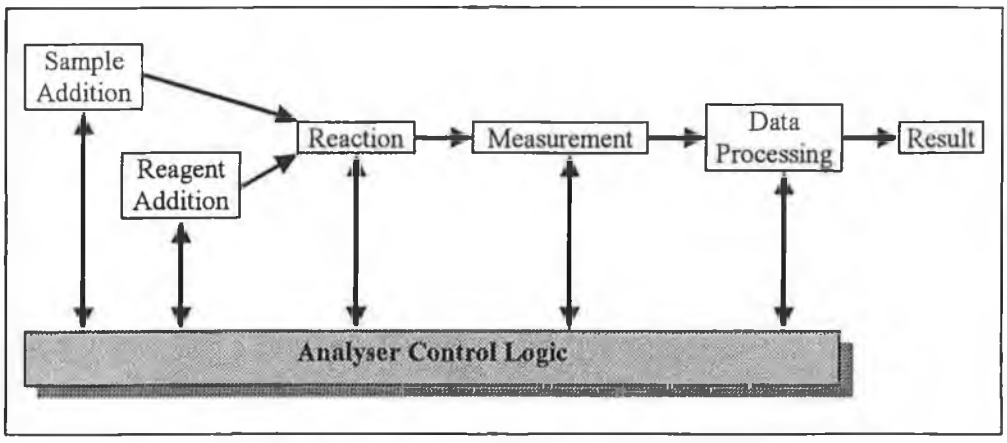


Figure 2.8 The functional steps involved in automated analysis of a sample

The basic architecture of modern analysers is based on a global controller module which monitors and regulates the operation of the various sub-systems necessary for the fulfilment of the analyser functionality specification. It detects and interprets fault patterns to produce the appropriate alarms to correctly associate errors with the responsible sub-systems. Depending on a specific

implementation in an analyser, some functional steps may be repeated to provide more comprehensive analyses. The figure illustrates the lowest common denominator in functionality.

The first viable clinical analyser performed Continuous Flow Analysis for a single analyte. The modern trend in analyser design is for discrete analysers offering a large range of tests and using Multiple Channel Random Access analysis [GAFF – 96], for example the Beckman CX7, Hitachi 717 and 747, Bayer Dax 96, and the Olympus AU5200. These are multi-functional analysers whose expense has limited their use to only the larger, well-funded medical laboratories. Continuous Flow non-discrete analysers such as the Technicon AutoAnalyser system are now obsolete.

Currently there are numerous categories of clinical analysers which satisfy specific niche market requirements depending on sample throughput and cost. They can be broadly classified according to a combination of operational principle and physical size.

It is not easy to precisely classify analysers according to operational principle since several principles may be utilised in one analyser. The International Union of Pure and Applied Chemistry have issued definitions of operational principles employed by automated analysers [IUP – 78] and the more common ones are listed below.

2.5.1. Operational Principles

Batch Analysis – a number of specimens are processed in the same analytic session or run.

Sequential Analysis – each specimen in the batch enters the analytic process one after another and each result, or set of results, emerges in the same order as the specimens entered.

Continuous Flow Analysis – each specimen in the batch passes through the same continuous stream of liquid and is subjected to the same analytical reactions as every other specimen and at the same rate.

Discrete Analysis – each specimen in the batch has its own physical and chemical space separate from every other specimen.

Single-channel Analysis – each specimen is subjected to a single process so that the result for a single analyte is produced.

Multiple-channel Analysis – each specimen is subjected to multiple analytical processes so that a set of results is obtained.

Parallel Analysis – all specimens are subjected to a series of analytical processes at the same time.

Discretionary Multiple-channel Analysis – specimens in sequence can be analysed by any one or more than one of the available processes; each sample can use any combination of the available methods.

Random Access Analysis – any specimen can be analysed by any, or all, available process, in or out of sequence with other specimens, and without regard to their initial order.

2.5.2. Physical Size

Pocket-sized analysers are designed to perform a single dedicated function such as blood glucose monitoring. Their simplicity make them ideal for use by a doctor or patient in a non-medical environment.

Desktop and **small analysers** have a low sample throughput and a limited range of tests. They are typically used in clinics and physician offices to provide confirmatory diagnosis or initiate more comprehensive investigation. An example would be the Boehringer Reflotron for measuring glucose, urea, cholesterol, etc.

Bedside analysers are essentially desktop and small analysers that are located immediately beside the patient in critical care situations such as Intensive Care Units (ICU's) and may provide communication facilties to interact with a nursing station. An example would be a Hewlett Packard's HP Care System.

Medium-sized analysers have a typical throughput of 50 – 120 samples per hour and can perform a large range of tests. They are expensive (> £50,000) and complex instruments necessitating their location in a clinical laboratory and operation by experienced laboratory technicians. They produce quite large volumes of data and may include some rudimentary data management capabilities.

Large analysers are based on the design and operating principles of medium-sized analysers but with a large range of tests and a throughput of more than 120 samples per hour. Due to the very large volumes of data they produce, they always have a dedicated data management workstation. They are very expensive (> £120,000) and used in large clinical laboratories where workload and budgetary constraints can support them.

As can be seen from the above categories, few analysers comfortably satisfy the definition of only one category. However all automated processes share common and/or logical steps, just as manual analytical processes can be viewed as procedures consisting of generic steps or functionality (refer to Figure 2.7).

2.6. Computerisation of Clinical Analysis

Traditionally laboratory reports were hand-written transcriptions of data and manually calculated results. The process of transferring instrument data to peaks on chart paper and then converting them into test results was tedious and prone to transcription errors. This process was sufficiently effective for use with the first commercially viable clinical analyser – the Technicon AutoAnalyser [COON – 58] – which was first marketed in 1957. However with the advent of multi-channel clinical analysers [KADI – 66], the task of manually producing reports was a major drawback. These automated analysers substantially increased the data being produced [SKEG – 64] by the laboratory. Whilst these analysers used pre-printed chart paper and generated an output for each result, the volume of data produced became so great that the main task of the laboratory technician was the transcription of results from worklists to report forms and log books. Thus sample handling and data management became the limiting factors in the rate of production of results.

The introduction of bar code readers improved sample handling capabilities by automating the identification of samples and the association with their corresponding entry in a worklist. This reduced the possibility of gross errors due to sample identification errors. Random Access Analysers permitted specific

individual tests to be requested from the total available on a multi-channel analyser. Thus the volume of data produced was reduced as only essential tests were requested whilst unnecessary test requests were minimised. This also focused the attention of medical professionals to request only necessary tests and eliminated irrelevant and distracting data produced by the laboratory.

The power of micro-computers to help in analysing large amounts of data for quality control purposes was recognised early [DITO – 70], and was a natural progression from the use of programmable calculators previously used for this purpose [WEST – 75]. During the late 1970's microprocessor-based computers became available that were small enough and cheap enough to be incorporated into new analysers. These computers were primarily used to control the analyser operations and assisted in the development of multi-channel Random Access Analysers. They also allowed many operator procedures such as calibration and data conversion to be automated to varying degrees.

Advances in micro-computers facilitated their use in sophisticated data analysis of quality control results involving more than one rule (multi-rule analysis) and automatic graphing facilities were incorporated into these systems [WEST – 84].

Despite technological advances to improve sample handling and reduce the volume of requests for spurious tests, the demand for the services of clinical laboratories continues to increase. Modern analysers monitor many of their functions and processes but do not integrate or interpret the data produced.

A comprehensive computer-assisted system for Quality Assurance, technical validation, fault diagnosis, maintenance scheduling and maintenance instruction will ensure minimum deviation from the recommended operating protocols as defined by the manufacturer or in SOP documentation. Computerised technical validation ensures that questionable results are isolated and brought to the attention of the operator. Thus the operator can investigate the few problem results without the distraction of dealing with the thousands of results that are technically valid.

The incorporation of the knowledge of experts into Decision Support Systems (DSS's) to produce Knowledge-Based Systems (KBS's) has been shown to be possible in many different areas of the laboratory environment [BENS – 80]. KBS's provide a level of technical competency greater than that which the majority of individuals could possibly attain in a lifetime of dedication, and permit the laboratory technicians to concentrate on the focus of their abilities. Unfortunately only instruments currently with dedicated workstations included in their basic design concept have DSS's natively built in and these systems are in the class of expensive large analysers.

The benefits of having such systems available to other smaller analysers would provide work methodology efficiencies similar to those obtained from the large integrated analytical systems with inherent DSS's, but on a wider scale due to the larger installed user base of smaller clinical analyser systems.

2.7. Decentralisation of Laboratory Services

There is a growing trend for analyser manufacturers to produce instruments specifically designed for non-laboratory sites. Most hospitals have recognised the advantages (and sometimes the necessity) of distributing clinical analysis services and consequently locate analysers in strategic sites for efficient healthcare service delivery where non-laboratory staff have to perform analyses. This has led to the increasing use of near-patient analysers in areas such as Emergency Wards and Intensive Care Units (ICU) [BURR – 90]. Such analysers are designed to be used by non-laboratory personnel.

It is well recognised that the performance of near-patient analysers is linked to the level of support provided by the central laboratory [MARK – 88]. A well-designed Quality Assurance scheme is essential for the safe and efficient use of analysers in non-laboratory sites, particularly because the instrument would be out of the immediate influence and control of the experienced laboratory staff. Instead it would be under the direct influence of inexperienced operators such as doctors and nurses, and the possibility of a catastrophic event occurring outside the

bounding timetable of a static maintenance schedule is consequently greatly increased.

Near-patient analysers therefore require regular skilled maintenance to ensure optimum performance since these casual operators would not possess the technical expertise to perform corrective maintenance whenever problems arose [RUBI – 79][LEWA – 92]. Hence these distributed instruments place the emphasis of the Quality Assurance scheme on preventative maintenance.

To evaluate the operational performance of distributed instruments remotely would alleviate wasteful time spent travelling to sites by the experienced laboratory personnel. This remote monitoring, or telematics, could be done on a scheduled batch basis, on demand whenever initiated by laboratory personnel, or in real time for optimal monitoring. Laboratory personnel could then spend the maximum amount of their time in the laboratory performing laboratory-based analyses and interpreting data sent from remote instruments.

2.8. Conclusion

The combination of advanced maintenance procedures and telematics would ensure that all instruments could be part of a "distributed laboratory". This would enable the performance of non-laboratory instruments to be monitored remotely as if the analysers were located in the laboratory instead of the necessity of a laboratory staff member explicitly travelling to the site of an instrument to perform the required Quality Assurance procedures. Thus the instrument can be included in whatever computerised Quality Assurance scheme is implemented in the central laboratory. An intelligent maintenance scheduling system which includes an early warning facility would result in the central laboratory being better able to support and control these distributed analysers. The resultant intelligent preventative maintenance scheduling would substantially increase the operational availability and accuracy of results available to the healthcare professionals using the facilities of the remote instrument.

The purpose of this thesis is to evaluate and propose the design of an architecture for the interface of medical instruments in an open environment which would support the remote maintenance of decentralised clinical laboratory services. This design concept is dependent on the requirements of identified users (refer to Section 3.3), and the concept is introduced in the fourth chapter.

3. User Requirements

As with any development project, the design phase is a critical component of the project life-cycle, and the design phase is dependent on the specifications obtained during the user requirements identification stage. An incomplete requirements specification would inevitably result in an incomplete design concept, which would yield an unsatisfactory solution. Getting a comprehensive user requirements specification is not trivial as the user's wishes often occlude the user's actual needs. Therefore, substantial time was needed to isolate realistic and attainable requirements which would definitively result in improved work methodologies.

Assistance for specifying the user requirements was obtained from a senior medical laboratory technician in St. James Hospital, Dublin [GAFF – 96]. This emphasised the focus of the design concept on satisfying the real-world requirements of a modern clinical laboratory and its associated distributed components. The categories of users who impact on the delivery of clinical analysis services were identified and their characteristic stipulations identified. Each user category interacts with the clinical laboratory in a different manner and computer assistance with these interactions would be beneficial to the delivery of an improved service.

3.1. Categories of Users

Four broad user categories were segmented according to how they interact with the operation of the clinical laboratory. The most obvious category is the collection of people who deal directly with laboratory instruments - namely the laboratory staff. This "front-line" group of operators leads to the identification of the ancillary group of operators who use clinical instruments which are remote from the laboratory. These two groups use analysers to fulfill the needs of the third group - the category of people who actually request clinical analysis of samples. The final group identified was secondary to the actual delivery of clinical analysis because they are concerned with the overall performance of the laboratory.

3.1.1. Operators of Laboratory Analysers

These operators are the skilled professionals who staff the laboratory. Their competence with the operation and maintenance of clinical instruments is a critical constituent of their professional capabilities and responsibilities. Their primary role is the creation of clinically accurate reports of patient sample analyses. They are also concerned with the monitoring of the operational performance of instruments and make extensive use of control and calibration data to supplement statistical analysis.

The automation of sample result collection and recording (be they patient, control, or calibration samples) would be of great benefit to laboratory staff. Result recording is a long, tedious process which can involve thousands of results and therefore gross transaction errors can ensue from the sheer volume of data to be processed. Automation would free laboratory staff from this process and allow them to concentrate their attention on the few anomalous results which may be produced.

3.1.2. Operators of Remote Analysers

These operators are doctors and nurses who use medical analysers in a near-patient environment, such as an Intensive Care Unit (ICU) or ambulatory EEG's. As such their primary competency would not include clinical instrument operation and maintenance. But the technical validity of the results they obtain would depend on the efficient performance of the instrument. Thus its participation in a Quality Assurance scheme is essential.

Remote connectivity between the instrument and a computer would enable the computer to immediately analyse control and calibration data recorded from the instrument, without the need for a laboratory technician to manually record the data after visiting the remote instrument. This would permit the instrument to be included in a computer-based Quality Assurance scheme that would monitor its performance using various automated statistical techniques and would thereby result in improved technically valid results.

A real-time interaction facility between the operator of the remote instrument and a laboratory technician would allow the instrument operator to benefit from the core competency of the laboratory technician. This would be particularly valuable when an unexpected circumstance occurs which is outside the limited experience of the remote instrument operator. A more knowledgeable laboratory technician would be able to assist using a real-time interaction facility without having to visit the remote instrument, thereby decreasing the downtime of the instrument.

3.1.3. Test Requesters

The group of people who request sample analysis tend to be a diverse selection of healthcare professionals which include hospital doctors, consultants, and GP's. Typically they fill out a request form and eventually receive a report to assist them in their diagnosis and treatment of patients.

If they had access to an appropriate computer, remote connectivity with a laboratory computer would enable them to directly request the appropriate tests to be performed on a sample. The request would be made available to the worklist for the corresponding analyser and the result would be recorded. This result could then be reported back to the healthcare professional via the remote computer link, thereby expediting the report process by circumventing the traditional paper-based method. The paper-based method could be retained for documentary purposes, but the computer request and report could be logged to provide a "paper trail" of the sample analysis.

3.1.4. Laboratory Administrators

This group is concerned with the productivity and quality of service delivered by the laboratory. It is composed mainly of senior laboratory staff and laboratory administrators whose responsibilities include monitoring general performance indices. They are particularly concerned with the operational availability of all

laboratory instruments to try to minimise the costs of repairs and downtime. This information can be obtained from control and calibration data. The recording of these instrument performance data and making them available through remote connectivity would facilitate their analyses. They would consequently be able to automate much of their report generation methods since all the pertinent data would be computer-based and accessible to their computers.

3.2. Sample Processing

The system for clinical analysis was described in Section 2.4. The existing system is heavily reliant on paper-based requests and reports. Once a request form and sample are received in the clinical laboratory, the sample is prepared whilst the request is manually keyed in to the Laboratory Information System (LIS) to facilitate automated test requests at the local level in the laboratory. Once the sample has been submitted to the analyser, the tests requested are entered into the analyser or it is informed by a worklist from the LIS, and sample processing commences. When the result is available it is usually transcribed to a standard duplicate report form for delivery to the requesting healthcare professional. If this process was automated it would release laboratory staff from what are essentially administrative duties, allowing them to be more productive elsewhere.

As stated in Section 2.4, the technical validity of the report is dependent on an effective Quality Assurance scheme. In the laboratory, Quality Assurance is the responsibility of the local laboratory personnel and is given a high priority. As such the statistical monitoring techniques utilised for Quality Assurance are quite advanced. The performance trends of numerous laboratory instruments are usually observed, manually recorded, and sometimes manually correlated. These trends assist in the scheduling of preventative maintenance as the performance of an instrument deteriorates, and also highlights essential corrective maintenance as technically invalid results are produced. This allows a fully integrated maintenance program to be applied throughout the whole laboratory with the aim of producing technically valid data all of the time.

However, with the modern trend in distributed clinical analysis and bed-side analysers, for example in an environment like an Intensive Care Unit (ICU), there would be primitive or possibly even no statistical monitoring techniques in place for these remote analysers. Preventative maintenance is usually only on a fixed scheduling basis instead of as the instrument performance degrades, and substantial data could be produced before it becomes obvious that corrective maintenance is necessary. Comparative correlations between the performance characteristics of remote analysers and similar laboratory-based analysers would also not be possible. This adversely impacts the operational availability of analysers that are remote from the influence of the clinical laboratory Quality Assurance scheme, and even affects the technical validity of the results produced.

3.3. General User Requirements

The concerns outlined above provide the basis for the user requirements:

1. **Record instrument performance data.** This would obviate the need for human interaction for this simple task and as a result would free up human resources and reduce the possibility of gross errors by removing the error-prone data entry phase.
2. **Record patient data.** As various tests are carried out, their results need to be associated with each corresponding patient. The administrative overhead for this task is quite substantial and data entry errors can be an unfortunate side effect.
3. **Remote connectivity between the instrument and a geographically removed computer.** This would enable remote monitoring of an instrument's performance for inclusion into a laboratory Quality Assurance scheme, as well as making patient results available at other sites, for example at the location of the initial request.
4. **Real-time interaction between the operator of the instrument and the user of a remote computer.** This is to facilitate operators at each computer to communicate with each other. This would be of benefit for the maintenance of a remote instrument by an inexperienced person whilst receiving assistance

from a more experienced person, who would typically be located in the central clinical laboratory.

3.4. Summary

It was envisaged that the user specifications would result in an application that logs data from an analyser and provides the facility for remote interaction. The primary concern was for remote access to performance data resulting from control and calibration sample data for statistical monitoring of the performance of the instrument. This would enable a laboratory clinician with expertise specific to a particular instrument to remotely monitor its performance and even to correlate its performance with other similar instruments under the direct control of the clinical laboratory without the need for physical proximity to each instrument.

As potential problems become apparent, the real-time interaction facility would enable a discussion between an experienced laboratory staff member and the local operator to take place. Thus the laboratory-based expert could inform the local operator of necessary preventative or remedial action. The interaction facility would even allow the expert to instruct the local operator in any procedures required for the action.

Just as a remote clinician would have access to control and calibration data from an instrument, a remote physician could have access to patient results. This could eliminate the need for manually-prepared report forms and reduce the delays in the delivery of reports. For a physician outside the immediate boundaries of the hospital reporting mechanism, delivery delays are substantial and any reduction in these delays would be extremely desirable.

4. The Design Solution

4.1. Foundation Guidelines

Before any design work was initiated, several strategic design decisions were made. These were concepts which would be applied to the design process in the form of a generalised guiding framework in order to assist in resolving design issues.

It was decided to align as much of the design concept as possible with "open" standards. This would imply using existing technology as much as possible where it was based on mature standards - be they de facto adopted standards or standards introduced by one of the main standards bodies such as ANSI, CCITT, or CEN. At the time of writing there is much research being carried out in the area of open standards computing (CORBA, COM, DCOM, or Java), these are in their infancy. The medical domain tends to be very conservative when considering new technology due to the data-sensitive environment and fear of litigation, so preference is given to technologies that have matured over time and have evolved into robust and reliable standards.

Mature standards have well-defined functionalities and concise interfacing requirements. With such clear interface specifications, the various functional components can be integrated in a predictable manner to ease the development process. There is also the benefit that interface specifications in existence for quite some time invariably result in the development of software components for interoperability between them (as the filters for converting DBase files into Excel files exemplify). This is of particular relevance as the design concept goes through its iterative evolution of future versions where additional functionality and new features are introduced. These "off-the-shelf" software components can thus be reused to speed up various aspects of the development process.

This functional component development approach could be considered as a form of "plug-and-play" and depended substantially on the second guiding design

decision - a modularised design concept. As the user requirements are analysed, functional domains must be identified and subdivided into coherent, self-contained units of purpose. Each unit must have a specific sphere of responsibility for delivering certain key functionalities. The methodology which a unit implements to deliver its functionality requirements must be its own concern and not dependent on any other unit. Whatever data, algorithms or procedures are required by a unit are available for use only by that unit and are therefore essentially invisible to other units. This technique of "data hiding" (or encapsulation) is based on data abstraction which is one of the central concepts of object-oriented design methodologies [MURR – 93].

The third guiding design decision was that the basic computer paradigm must be Client/Server to facilitate a distributed architecture. Rather than designing one large solution concept, a modularised approach will produce self-contained units that can be dispersed throughout a data processing environment. This data processing environment could be contained within one computer or spread over multiple computers, with the modularised units interacting in a Client/Server fashion.

The final guiding design decision was to reuse as much technology as was already available in the target environment. This decision minimises the cost and risk of the investigation as well as eliminating the interference with existing methodologies and training requirements in the clinical laboratory.

The benefits of this kind of a guiding framework are many:

1. Adopting an architecture driven by open standards focuses compliance with existing methodologies. As stated in Chapter 1, there is substantial international interest in the whole area of computerised healthcare delivery without any clear dominant standard as yet. Whenever an internationally adopted standard is implemented, an observance of an open architecture for this design concept would simplify its integration with the adopted international standard.

2. Modular development facilitates the design and implementation of any large project. As each modular unit is created, it need only focus on fulfilling its functional requirements. This can reduce complex cross-functional dependencies and focuses design on identifying coherent processes and tasks in order to segment them into well-defined units of functionality. The resultant model can be developed in a more controllable piece-wise manner which simplifies project management and testing, enabling "black box" unit testing to be extensively utilised.
3. Components that satisfy modularised architecture considerations could be removed and replaced by other components which similarly satisfy modularised architecture considerations and maintain the common interface requirements. Thus components which deliver a superior functionality service through improved reliability, increased performance, or more complex functionality could seamlessly replace existing components whilst eliminating the impact on unrelated components.
4. Attempting to maintain an "open" environment was desirable in order to support a modular engineering model and clearly defines the interaction between component modules of the solution. The interoperability of components depends upon commonality of solution definitions and communication protocols and should allow the leveraging of reusable definitions and code modules. As new design concepts are developed an open architecture environment permits them to be absorbed into the total solution concept on an individual basis without a redefinition of other unrelated components.
5. A Client/Server communication model provides the foundation for building strategic information applications through system connectivity. It permits rapid modification as data processing circumstances change [DONO – 94].

4.2. The General Components

In order to obtain clinical data from an instrument, it is proposed to use serial data communications with an instrument-to-host module. This module is responsible for all the physical and logical requirements necessary to connect to the

instrument and conduct serial data communication. The serial byte stream is parsed into its component data elements and then interpreted to identify the context of the data before being processed by other components of the design solution.

A transaction-generating module processes these data elements. This module packages the data elements into a transaction which consists of an appropriately formatted body containing the data and a header containing certain context information.

These transactions can be viewed as requests for services which get sent to service provider modules by a transaction processing module (called the Transaction Processor). The service provider modules act on the information contained within the transaction to produce a desired action such as storing the data in a database, displaying the data in a graph, or transmitting the data to a remote computer.

It is proposed to distribute these services throughout functional modules. A distributed design concept such as this requires a modular functionality definition where each functional unit provides a complete service. The functional units can then be distributed throughout the data processing environment as required. It is proposed that they would interact with each other using a 3-tier Client/Server communication model.

As with any Client/Server architecture, the elements of the architecture can reside on the same computer or can be distributed throughout several computers that can communicate via a data network. When a service is being provided by a remote computer, the Transaction Processor initiates and maintains a data link with the remote computer in order to avail of the service(s) required by the transaction.

A final additional component is necessary to satisfy the last user requirement of a near real-time interaction facility between an instrument operator and a remote user. This component permits users on different computers to type messages to each other. Once a sentence has been typed on one computer it appears on the

other computer, thereby permitting a dynamic exchange of questions and ideas. It would be particularly useful for providing technical laboratory assistance to healthcare personnel who may not have the necessary expertise – for example doctors requesting sample assays and an assistance with the interpretation of anomalous results.

4.3. Available Services

An analysis of the user requirements (Section 3.3) yield the following necessary services:

1. The primary prerequisites revolve around data processing based on storing and retrieving data. This focus on data and its associated administration encourages a design concept which harnesses data storage as its core design element. Thus databases will provide the foundations upon which data processing will revolve.

It is proposed that the service of storing data to, and retrieving data from, databases will be performed by one coherent module. This module will be responsible for the physical and logical connection to targeted databases. It will leverage the widely implemented Standard Querying Language (SQL) standard to interact with the databases. Since SQL is a mature standard [GROF – 94], all the leading Database Management System (DBMS) vendors support it. This decision aligns itself with the first guiding design decision of using mature standards wherever possible. Due to the diverse selection of DBMS's in use in the distributed environment of medicine, SQL provides the necessary flexibility to interact with existing DBMS's, such as patient demographic information or clinical records stored in LIS's. This flexibility permits the re-use of the investment in existing database technologies – the fourth guiding design decision.

2. To support the near real-time interaction facility between an instrument operator and a remote user, the service will provide interaction with the user. It is proposed that the service provider presents a graphical user interface to the user to permit text entry and text display. It generates transactions based on the

entered text that the Transaction Processor recognises as requiring transmission to a remote computer. The Transaction Processor then uses the same mechanism as the other distributed elements of the design solution to transmit to the remote computer. The remote computer will have an identical service provider module and they will interact with each other in a symmetric peer-to-peer fashion.

As additional functionality is needed, extra services can be defined and the appropriate service providers designed. These can then be added to the design solution in a scalable manner so long as they adhere to the mechanisms outlined in the chapter. An example of additional services would be ones responsible for elements of a Quality Assurance scheme. These would establish the Real Time Quality Control (RTQC) status for all patient results and perform automatic technical validation. These kinds of services would have comprehensive maintenance and fault diagnosis facilities in order to support Long Term Quality Control (LTQC) monitoring.

4.4. Analyser to Host Computer

As summarised in Section 2.5, medical instruments come in numerous different categories as described by their operational principle and/or their physical size. The larger, more complex (and hence expensive) analysers usually have a dedicated data management workstation which processes results and applies Quality Assurance schemes locally to the data produced by the analyser. However, while this kind of system provides excellent data management facilities, the cost of integrating it with other computers is usually prohibitive and so it tends to be a stand-alone system. Being isolated in this manner does not easily permit the correlation of statistical performance data with other similar analysers, except by the traditional manual methods that are time-consuming and prone to gross errors due to transcribing mistakes.

Smaller analysers do not have dedicated data management workstations. Therefore, in order to support a computerised Quality Assurance scheme, data

processing capabilities must be made available to the necessary analysers. It is proposed that this processing power be provided by a computer based locally to the analyser. This computer could be quite a modest Personal Computer (PC) such as commonly found throughout the medical domain. The PC offers a universal, scaleable, economical hardware platform with "open" interfaces suitable for connection to laboratory devices [NORG – 95]. Therefore the abundant PC is a logical choice as the most cost-effective platform for medical instrument communication.

The local computer would provide services to all analysers connected to it, which could include the necessary data management required by a computer-based Quality Assurance scheme. This computer will be called the "host" because it hosts a variety of services for the analyser to which it is connected (Figure 4.1). The data management services provided by the "host" could range from the most basic logging of sample data for permanent storage, to making worklists available for sample processing, to real time technical validation of samples, or even a comprehensive Quality Assurance scheme with an integrated maintenance scheduling module.

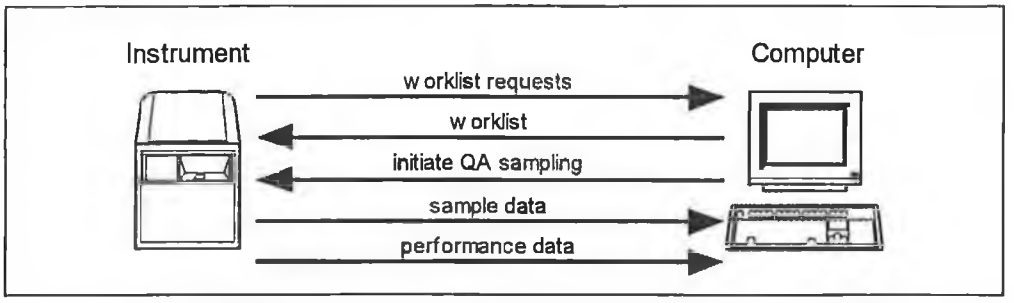


Figure 4.1 Examples of services provided to an instrument by a "host"

It is proposed to use the PC's serial port to exploit the serial data communication protocol which almost all analysers generically support. If the PC being utilised has more than one serial port, then it could act as "host" to as many analysers as it had serial ports (Figure 4.2). A "daisy-chained" polling mechanism could service each serial port in turn or else a PC with a multi-tasking operating

system such as IBM's OS/2 Warp or Microsoft's Windows would be required for the efficient hosting of multiple analysers. Since the vast majority of PC's in the test site use Windows 3.11, it was decided to leverage the multi-tasking features of this operating system. This decision follows a scaleable path as full 32-bit operating systems with proper pre-emptive multi-tasking such as Windows 95 and Windows NT become the prevalent operating systems in use in the target environment.

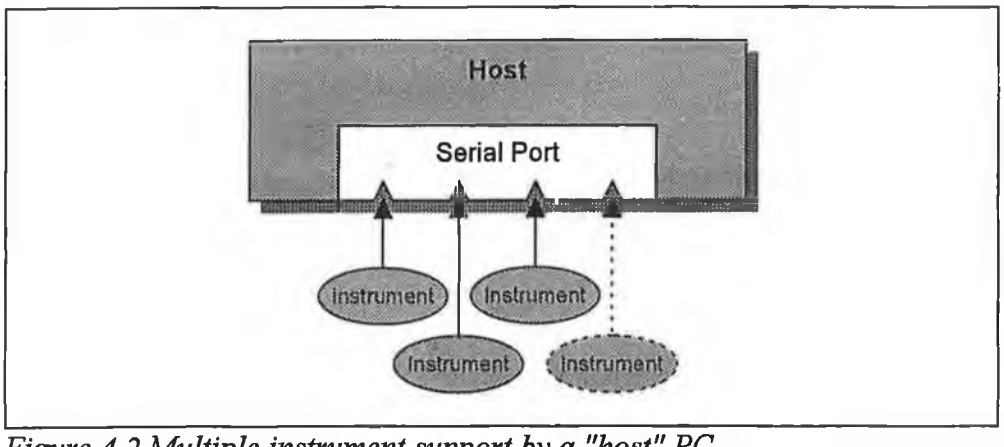


Figure 4.2 Multiple instrument support by a "host" PC

In a practical sense the module responsible for communicating with the analyser must accept and interpret all data types generated by the analyser including Quality Control data, calibration data, hardware alarms, and data alarms. The correct interpretation of these data types supplies the context information which other modules use to decide how to process the data. The design of this module is covered in more detail in Section 5.6.

4.5. Transaction-based Architecture

As mentioned earlier, it is proposed to utilise transactions to interact between the various modules contained within the architecture. Because the design concept is predominantly centred on data storage and retrieval, it logically follows to extend the "open" environment of databases to the services. It is therefore proposed to base all transaction protocols utilised for service requests on a format which is closely linked with data processing and administration. This syntax will be directly derived from the syntax used natively by almost all databases - the

Structured Query Language (SQL) - for interacting with data stored in SQL-compatible databases. By using a SQL-based transaction protocol and storing all relevant data in databases instead of proprietary data structures, an "open" architected environment is promoted.

As data becomes available from the instrument interfacing module, the transaction-generating module embeds the data in an appropriate SQL statement which is then wrapped in a transaction along with context information in the header (the exact format is described in Section 6.1).

The Transaction Processor dispatches the transaction to an appropriate service provider by based on the context information (Figure 4.3). For example, if the data is to be stored in a database, the receiving service provider is one which supports a service for interacting directly with a database. This service provider then utilises the embedded SQL statement to store the data in the database.

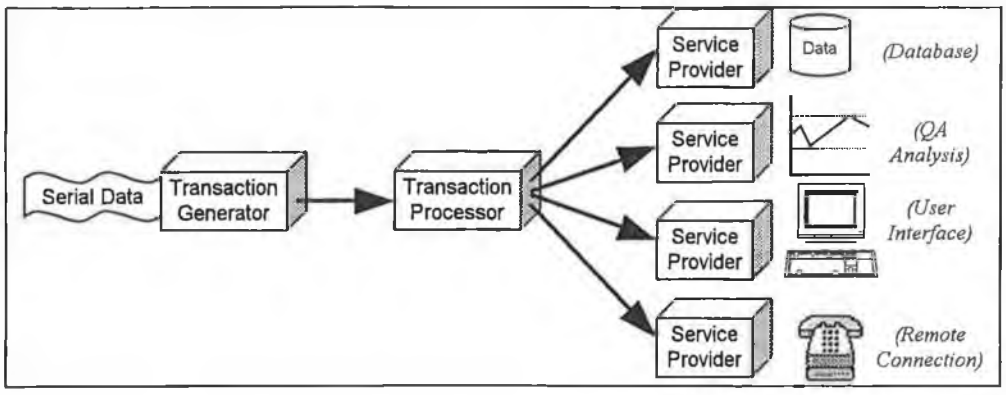


Figure 4.3 Data flow through the transaction-based architecture

As additional services are introduced, the existing format for transactions can be used with specific codes introduced to support any new data exchange information. In this way the design solution scales linearly by releveraging the underlying architecture. Only new service provider modules and new transaction generation modules would be needed to handle any new data exchange information required by these additional services.

4.6. Client/Server Architecture

The classical model of Client/Server communication uses an entity (the server) which provides services and functionality to anything that requests it. An entity that requests services from a server is referred to as a client, and it is always the client who initiates the process. A 3-tier Client/Server communication model is an extension of the classical model with an additional layer introduced between the two entities to provide extra processing and arbitration for the process (Figure 4.4). This insulating layer is commonly referred to as "middleware" and provides the necessary functionality to interface between an entity that wants data and an entity that can provide that data.

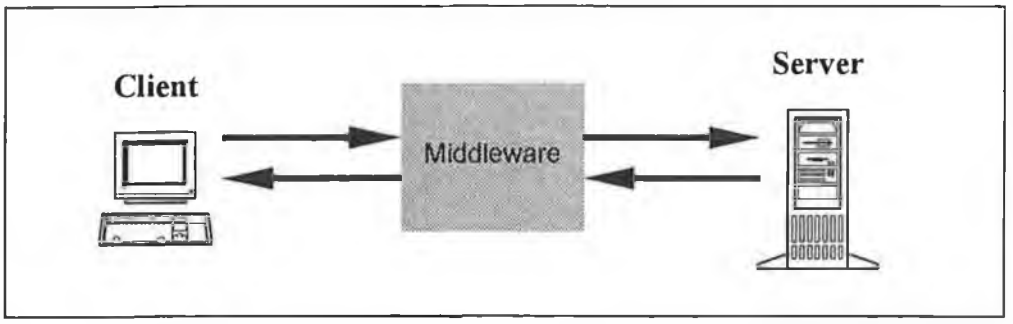


Figure 4.4 The 3-tier Client/Server communication model

The majority of services will inter-operate with a database on some level and the other components of the design concept using transactions in a 3-tier Client/Server communication environment (*Figure 4.5*). When a client module requires data processing by an available service provider, it will request the service from the service provider using clearly defined transactions (Section 6.1).

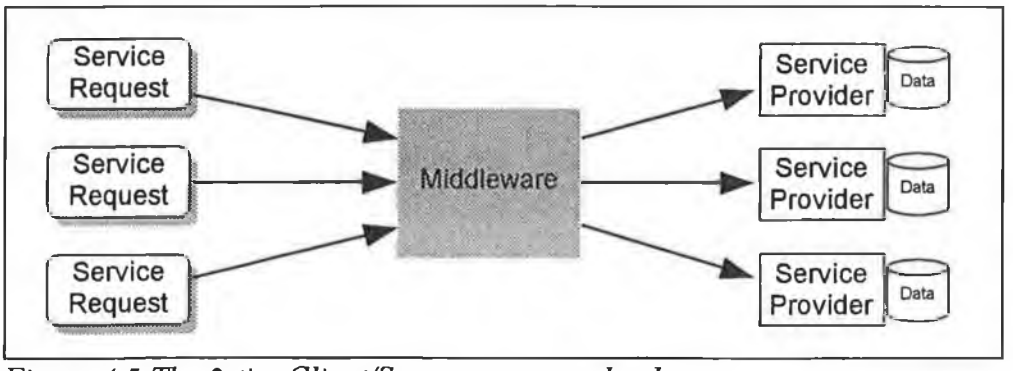


Figure 4.5 The 3-tier Client/Server access to databases

As an example of this scheme it is proposed to utilise a data storage area as a repository of all data which may be utilised to fulfil service requirements. As a client module requires pertinent data, it obtains it from the data store using a 3-tier Client/Server paradigm (Figure 4.6). The data store behaves as a Server whilst the client module generates a transaction which requests the data.

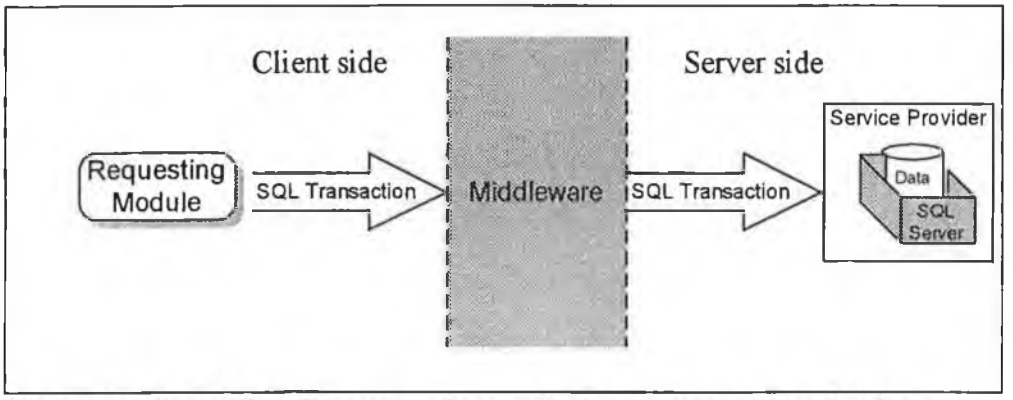


Figure 4.6 Data Flow Diagram of a service interacting with a database

This data store is a database (or collection of databases) of sufficient complexity to support efficient data storage and retrieval. The structure of the database corresponds to the nature of the data it has to store and as such needs to be custom designed. Data is obtained from analysers connected locally to the "host" and hence the database structure is dependent on the specific content of messages received from those analysers. The data is stored in the database by a service provider using the similar Client/Server transaction protocol as used by service providers retrieving data from the database.

The middleware layer performs transaction translation and processing, and ensures that the transaction is directed to the correct service provider which administers the data store. The middleware layer functionality could also be extended to provide additional services such as queuing of transactions and tracking receipt confirmations and service acknowledgements on the successful completion of the transaction.

In the proposed design concept the transaction processing module (Transaction Processor) performs as the middleware. As data is assimilated from an analyser, the interfacing module generates a transaction. As stated earlier, this transaction will be based on a SQL statement. The Transaction Processor will drive the transaction through to the appropriate service provider module, resulting in the data being stored.

4.7. Distributed Components

Remote connectivity which supports transaction processing on distributed computers would empower the services of a medical laboratory that is distributed throughout the healthcare environment. Analysers which are located outside the laboratory (for example in an Intensive Care Unit) would benefit from transaction processing through the capabilities of remote connectivity. This would permit them to participate in whatever Laboratory Information System is in place in the laboratory. The resulting conceptual model is based on hierarchical "spheres of influence" where high level entities can interact with single or multiple lower level entities (Figure 4.7). The typical communication model for this kind of concept is Client/Server, where the higher level entity behaves as a server while the lower level entity initiates transactions just as a client typically does.

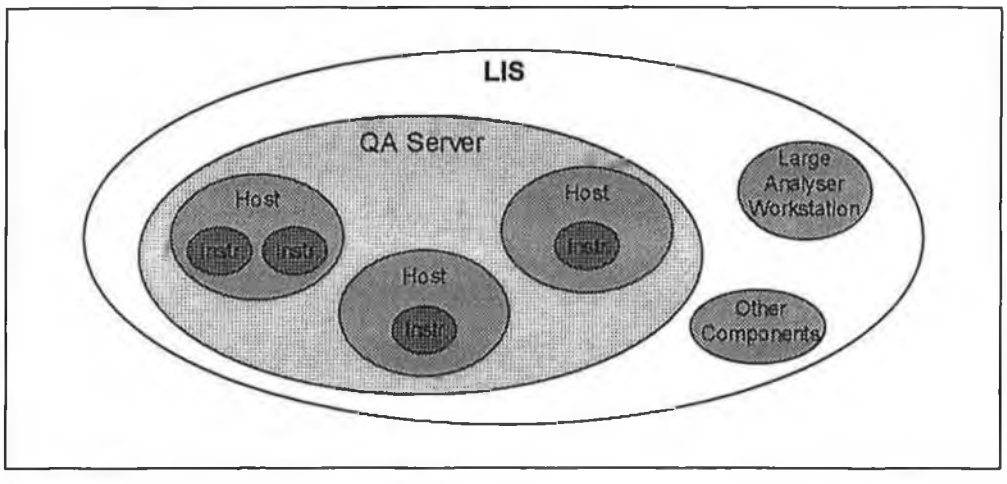


Figure 4.7 Conceptual model of the integration of laboratory components

In this kind of configuration it is proposed that a computer in the medical laboratory behaves as a centralised Quality Assurance service provider. "Host" computers connected to analysers perform transactions with this centralised "QA server", supplying it with whatever data it requires for its QA analysis, such as control and calibration data. The "host" computers of remote analysers would thus appear to interact with this centralised QA server in an identical manner as "host" computers of analysers located in the laboratory environment using SQL-based transactions.

The QA server could even proactively trigger Quality Assurance assays to be performed on a remote analyser to augment whatever data it already has available to it in order to contribute to specific Quality Assurance procedures. This would necessitate the temporary syntactical reversal of roles as Client and Server, so that the QA server could initiate a calibration run on the remote analyser and therefore behave as a client. Depending on the features available in the analyser, its "host" could act on the transaction and trigger an automatic calibration run, or else it may have to alert a local operator to perform the calibration run. Either way, the results would be reported to the QA server which could then complete its Quality Assurance analysis procedures.

The operating performance for each remote analyser could be tracked and analysed from a central location where it could benefit from the supervision and interpretation of expert laboratory personnel.

As results are obtained from an analyser, the local host computer can store them locally in databases using the previously mentioned SQL transactions. However, by collecting the data from numerous analysers distributed throughout the hospital environment, substantial statistical analysis can be performed on all the relevant information (Figure 4.8). This would require the availability of data at one processing location.

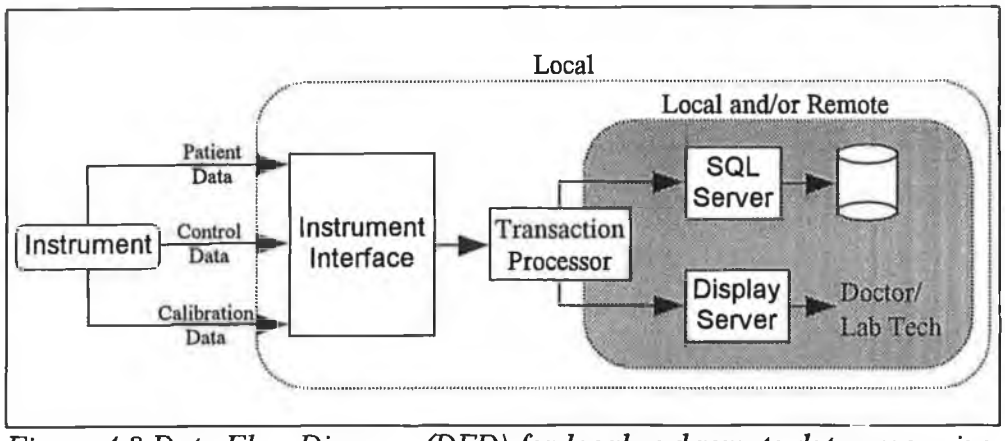


Figure 4.8 Data Flow Diagram (DFD) for local and remote data processing

If operating performance data from all analysers are available in a central location then it would facilitate their correlation with operating performance data from similar analysers. A statistical trend analysis of these data sets would indicate that it conforms to a required performance level. Deviations from this level would be immediately obvious and would trigger an alarm. The alarm would prompt a laboratory technician to perform an analysis of the reasons for the offending analyser's sub-optimal behaviour. This, for example, might result in giving an indication of the impact of a remote analyser's environment on its operating performance since it could be compared with the operating performance of similar analysers in the relatively controlled environment of the clinical laboratory. A remote analyser might indicate a tendency to overheat through usage as the day progresses. By comparison with similar analysers it might be obvious that other analysers do not overheat and further investigation is required to rectify this situation. It may subsequently be discovered that overheating in the remote analyser is due, for example, to its ventilation ducts being blocked by adjacent machinery and so an improved air flow around the analyser would solve the problem.

To transfer the data from local host computers to a centralised statistical processing computer or a remote data storage station necessitates the utilisation of a data communication protocol. The data communication protocol needs to transparently and reliably transfer data between computers. It is proposed to implement the low-level protocol upon which the widely popular Internet relies.

This protocol (commonly referred to as the TCP/IP protocol after some of its component parts) supports data exchange over numerous physical media between numerous different computer hardware and software types. It is based on a modularised architecture and has been in mature implementation for several decades – this mirrors design guidelines adopted for the design concept (Section 4.1). Its support of multiple platforms makes it particularly attractive for use in the medical domain which is composed of numerous different processing platforms.

While there are several higher level technologies utilising the TCP/IP (standard such as CORBA, Java, and WWW), these are specific to the applications which they serve. As stated at the beginning of this chapter, the primary prerequisite revolves around data processing and it was proposed to base the transaction protocol on embedded SQL statements in order to facilitate interaction with databases. Therefore the transaction architecture is specific to the application being proposed by the design concept. Introducing an alternative application-level technology purely for data communication would complicate the data communication process as well as needlessly adding an extra layer of complexity in order to simply leverage an emerging technology. Since these technologies are in their infancy, the design guideline of utilising mature standards (Section 4.1) also rules out their adoption in the design concept.

4.8. Interactive Communication Between Operators

Due to distributed nature of clinical services, several analysers may not be located within the immediate confines of the laboratory environment. The main users of such remote analysers would be medical professionals such as nurses and doctors who would not possess the skill-set equivalent to that of a laboratory technician. An interactive communication link between the remote operators and laboratory staff will provide access to such skills and was one of the user requirements specified in Chapter 3.

It is not anticipated that this service would replace voice communication, instead it is merely an additional mode of communication to complement the

traditional voice method. In certain circumstances it may even be preferable, for example when the two users are separated by very large distances and the telephony costs would be substantial, or when the concepts being discussed are sufficiently complex that seeing the words rather than hearing the words would be helpful. It would also provide a record of the discussion that can be referred to repeatedly after the discussion has ended, or as log of the resolutions decided upon during the discussion.

An e-mail-based communication link would not be adequate for urgent interactions due to the nature of electronic mail systems and their associated time lags. If the e-mail has to travel through several domain servers from the sender to the recipient, it is conceivable that it may not get forwarded from the mail queues of heavily-trafficked domain servers for several minutes. The cumulated delay could potentially be quite substantial. There is also always the possibility that the recipient may not have an active e-mail client and they would remain unaware of the pending e-mail. Since a remote computer-analyser pair would be used by several people in the remote site, if e-mail access were to be provided to all of them, there would be an administrative overhead associated with configuring and maintaining multiple profiles for the e-mail client software. Combined with this would be the training of the staff in the use of e-mail client software and the significance of switching of user profiles.

Obviously however, the proposed communication link architecture could not be securely used for sensitive communication since there is no authentication of participants. Therefore a participant could anonymously pretend to be someone else.

It is proposed that this link will utilise the underlying data communication protocol mentioned in the previous section – TCP/IP. It will take the form of a "chat" utility where each user can type text into a simple user interface and it will be displayed on the other users screen. Thus interactive discussions can take place where problems encountered at a remote analyser can be resolved with the assistance of a laboratory technician. This is in contrast to the existing method of

resolving problems where the laboratory technician would have to travel to the remote site, diagnose the problem, evaluate what materials are required, travel back to the laboratory to get the materials, return to the remote site, and then travel back to the laboratory. If the remote site was adjacent to the laboratory the time involved would not be too excessive. However if the remote site is very distant (for example the ICU in St. James's Hospital which is half a mile from the laboratory), then considerable time is wasted in travelling between the two points.

Whenever text is typed in the user interface and the Enter button hit, the text is embedded in a transaction along with appropriate context information in the header part of the transaction. The context information identifies the transaction as an interactive chat element as well as identifying the target computer for the Transaction Processor to correctly dispatch it to the recipient service provider module on the remote computer. This service provider module then displays the embedded text in the user interface for viewing and reaction by the operator.

4.9. Example

As a demonstration of how the various components interact with each other, consider the following scenario. A healthcare professional wants to analyse the effect of different steroid treatments on renal tubular acidosis caused by systemic lupus or erythematosus, as indicated by pH, CO_2 and bicarbonate concentrations in blood gas analysis. To this end several blood samples are sent to the clinical laboratory for analysis.

As this analysis is completed for each sample, the data is transferred via a serial link to the host computer where the instrument-to-host module receives and interprets it. Once it has been identified as patient data, the transaction generating module (called the Transaction Agent) can produce an appropriate SQL statement (an "INSERT INTO ...") that will result in the data being stored in the correct database. This SQL statement is then embedded in a transaction along with context information in the header part of the transaction identifying the type of data and the destination of the transaction. For the sake of this example, assume that the

database for patient data from blood gas analyses has been configured to reside on the host computer. Therefore the destination specified in the header is the local computer.

Once the transaction has been assembled by the Transaction Agent, it is dispatched to the Transaction Processor. Using the information contained within the transaction header, the Transaction Processor routes the transaction to the correct service provider – in this case one which provides connectivity to the blood gas analysis database on the local computer.

The database connectivity service provider marshals the necessary procedures to connect and communicate with the database. It extracts the embedded SQL statement and passes it through the exposed database interface to the native SQL handler/interpreter mechanism inherent in the target database. Since all the major database products contain a SQL handler/interpreter, the design decision to base the body of the transaction architect upon text-based SQL statements becomes obvious. Providing that the SQL statement is syntactically correct, the patient data is consequently stored in the database for whatever future data processing may occur (for example, report generation).

Consider now the scenario where the blood gas analyser is remotely located in the ICU and instead of processing a patient sample, a control sample is introduced. As this analysis is completed, the result of the control sample gets transferred to the host computer where the instrument-to-host module receives and interprets it as before - this time it is identified as control data. The Transaction Agent produces an appropriate SQL statement (an "INSERT INTO ...") and embeds it in a transaction along with context information in the header part of the transaction identifying the type of data and the destination of the transaction. For this scenario, assume that the database for control data from blood gas analyses has been configured to reside on a centralised QA server computer in the clinical laboratory. Therefore the destination specified in the header is the remote laboratory computer.

In order for the Transaction Processor to route the transaction to the QA server, it must initiate and maintain a data communication link between them such that the remote database connectivity service provider can receive the transaction. This is achieved using the TCP/IP communication protocol and the IP address of the remote computer. Once it receives it, the SQL statement is extracted as before and passed to the SQL handler/interpreter in the targeted database.

5. Instrument Interfacing

5.1. Introduction to Data Gathering

Computerisation of work practices in the business, manufacturing, industrial, investment and medical environments has met with great success in the area of data management [DONO – 94]. Relational databases, numerical spreadsheets, statistical analysis tools and, more recently, data warehousing methodologies have facilitated the analysis of previously incomprehensible volumes of data [HACK – 93]. As a result, business models have become more realistic, trend analysis has become more comprehensive, predictive analysis has become more accurate, and fluctuations in operational parameters of business and industrial processes can be identified and counteracted in real-time. As more and more data becomes available to data management systems, subsequent processing will produce improved yields in all the above areas.

Data management systems are not only concerned with processing and analysing data, they are also concerned with the gathering of that data. This includes the assimilation of existing data in written, printed, or digital form as well as new data as it is created. The traditional method of making data available to data management systems is through data entry using equipment varying from a keyboard and a terminal with a very primitive user interface, to a fully functional data management workstation. This manual transcription of printed data to digital form is very labour-intensive and therefore expensive. Automation of this process is highly desirable.

The conversion of existing information which is stored in a printed format to digital storage media has recently received a lot of attention due to the availability of new technologies. Libraries are among a number of institutes embracing the digitisation of printed works in order to exploit the benefits of digital storage. These benefits include:

- protection of delicate original documents by making their content digitally available,
- rapid retrieval through efficient cataloguing and indexing systems based on digital retrieval systems,
- viewing by many individuals at the same time using multiple digital copies,
- utilisation of the cheaper storage costs of digital media – the cost of storing a 300-page publication in a digital format on disk media is approximately $2 compared to $30 for traditional storage methods on shelves [LESK – 97].

The costs of digitising a standard 300-page document of printed information can range from $30 - $40 for scanning, $120 for scanning and Optical Character Recognition (OCR), and up to $600 for manually keying the original information [LESK – 97]. These methods still require some degree of human assistance and so are automated in only a limited sense. As far as published content is concerned, the majority of modern publications are natively created in digital format and thus do not require any digitisation. This limits the cost of assimilating new information to a digital format for use by a data management system in the publishing domain.

Very recently there has been a rapid trend towards integrating information from numerous different existing data management systems to produce large data sets from which information can be harvested. This results in a vast accumulation of data commonly referred to as an "information warehouse". By early 1996, over 90% of large corporations either had adopted or were planning to adopt data warehousing technology [ORFA – 96].

Richard D. Hackathorn defines a data warehouse as "a collection of data objects that have been inventoried for distribution to a business community" [HACK – 93], or more explicitly a data warehouse could be described as "an active intelligent store of data that can manage and aggregate information from many sources, distribute it where needed, and activate business policies." This data

management methodology utilises data which already exists in a digital format and easily lends itself to complete automation.

Manufacturing and industrial domains on the other hand obtain data directly from automated processes. Process automation is quite a mature area in these domains and the technology employed on their automated production lines tend to be custom designed robotic systems or more general purpose Programmable Logic Controllers (PLC's) which are particularly suited to controlling the analogue and digital signals associated with process control. These systems usually have embedded data management systems and readily support integration into automated process local area networks and factory-wide networks. The resulting integrated data management systems can provide the seamless integration of process, inventory, safety, financial, and personnel domains and results in the corresponding efficiencies due to effective inter-domain data communication. However, they are designed to satisfy the specific needs of their environment and are therefore not easily transferable to other domains.

5.2. Medical Data Management Systems

A comprehensive integrated data management system is highly desirable in the medical domain. Due to the traditionally conservative approach to embracing new technology by the medical domain, such data management systems have been slow to develop. The complex nature of the data requirements specific to the medical domain has also been a factor. For example, the validation and interpretation of clinical analysis results have a very strong correlation with a patient's demographic information, such as sex or age, and can impact on the validity of their use as diagnostic evidence [HOWA – 91]. The necessity of accessing data from diverse segments of the medical domain requires a depth of integration of various data sources not normally found in data management systems of other domains. Such a rich problem set requires an engineering solution tailored to the comprehensive needs of a medical data management system.

Currently there are several systems in use for managing data in the medical domain [BALL – 91][KENN – 94][SPAC – 87]. The majority are principally concerned with administering patient demographic information with varying degrees of integration with billing systems. There are also some legacy Laboratory Information Systems (LIS's), along with isolated data management systems involved with inventory control and data archiving. These are generally systems which have made the transition from the commercial sector without addressing the focused needs of the medical domain at the design level. Rather, they have been adapted to fulfil new requirements and as such address local needs in a proprietary and non-transferrable manner. This has been a factor in their limited integration with each other and with newer data management systems.

Existing printed medical information will need to be assimilated using techniques similar to those being employed in the publishing domain like scanning, OCR, and even manual keying. However, currently produced medical information also needs to be absorbed into the digital domain and, unlike the publishing domain, it is not automatically created and stored in a digital format. Some data-intensive operations like imaging can produce information in a digital format. Magnetic Resonance Interference (MRI) and modern X-ray machines achieve this through local hardware functionality, but this feature is not commonly exploited to its full potential due to the lack of a standard for digital image storage and the necessary infrastructure to support it.

In the clinical laboratory environment only the most sophisticated instruments have data assimilation and management features incorporated into their core design. High volume multi-channel random analysers like the Hitachi 747 are an example. These high-end instruments have a dedicated workstation which is used for test ordering and sophisticated data analysis methods for comprehensive result generation.

As medical instrument manufacturers react to the trend for integrated data management systems, they will incorporate the necessary features into an increasing number of products instead of just their high-end models. As technology

advances and becomes more cost-effective, medical instrument manufacturers will eventually incorporate data assimilation and management features into their entire product range whilst adopting a global standard for medical data processing. When that happens, medical instruments of all kinds will support the seamless integration of disparate medical data management systems.

5.3. Analyser Tests

In the normal course of operation of any instrument it's operational performance will tend to drift from its optimum due to a number of factors including age of the instrument, quality of its components, component failures, and sample variations. In order to limit the effect of this drift on the quality of result data produced for samples, it is necessary to perform additional categories of tests to monitor and supplement its efficient operation. These are outlined below.

5.3.1. Sample Tests

These are the category of normal tests that are performed on samples to produce results which are used to assist in patient diagnoses. Depending on the complexity of the analyser, a number of different channel tests can be applied to a single sample to produce multiple result sets providing information such as photometric composition and ion analysis.

These result sets are usually submitted to a technical validation process as the last authorisation step in the production of a laboratory report. This process could include delta checks to detect gross laboratory errors and sample mix-ups, and may even utilise pattern recognition techniques to detect inconsistent trends.

5.3.2. Calibration Tests

Calibration is the process of quantifying the conversion factors used to express measured values (absorbency, millivolts, etc.) as useful analytical quantities. As reagents age and analyser subsystem components wear, the measured values will

change. Calibration is required to adjust the conversion factor used in the calibration equation which converts from the raw measured values to the corresponding correct result.

The calibration test uses the measured values from a zero calibration solution and from a full calibration solution to provide the limits of the useful measurement range of the analyser. These measured limits are used internally within the analyser to generate the conversion factor for all subsequent sample analyses. Frequent variations in the conversion factor can indicate unstable reagents which rapidly decompose, or else an unstable subsystem within the analyser which would require maintenance attention. Thus calibration tests provide vital information to a Quality Assurance scheme.

5.3.3. Control Tests

Control tests compare analytical results for material of a constant composition over time. The control material is chosen to be of a similar composition as the samples being investigated and is stored in such a way as to guarantee its consistency over time (months to years).

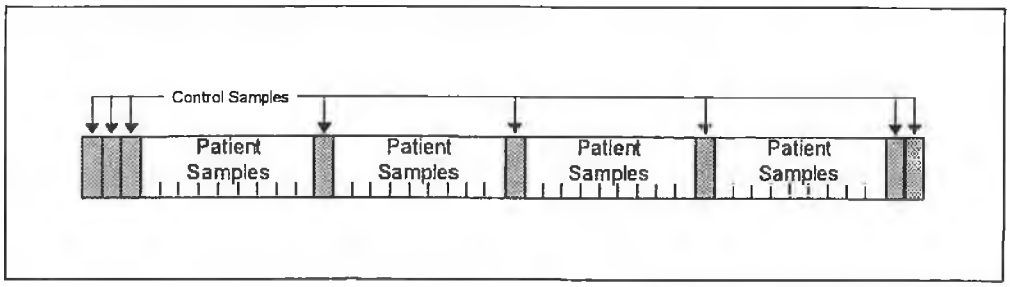


Figure 5.1 Control test locations within several runs of patient samples

The control test is performed on the material at regular intervals under exactly the same conditions as the normal samples. The results from the control test are analysed to determine the quality of the normal samples results and any increase in imprecision or bias in the analysers measurement subsystems. Typically three control materials are chosen to represent low, medium, and high concentrations

and the frequency of control tests is dependent on the anticipated fault rate. "Nested" stages of control tests involve assays being performed at an initial check stage, a monitoring stage, an end-of-run stage, and an assessment of several runs (Figure 5.1)[ARON – 84].

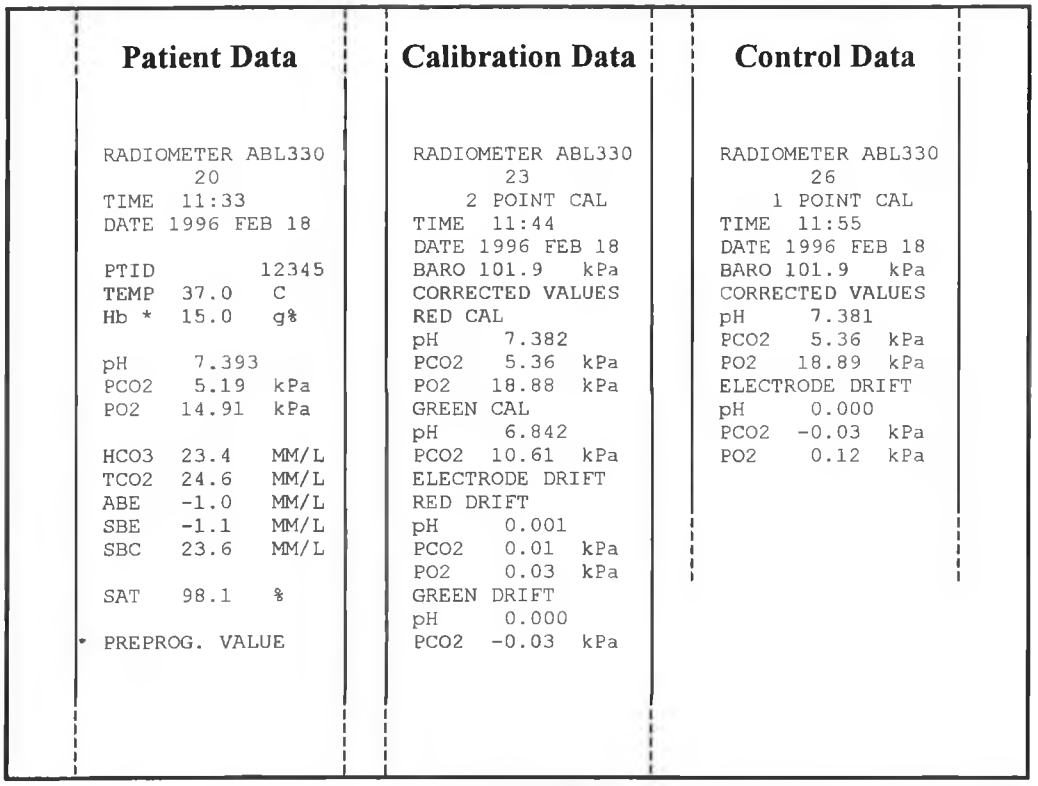
5.4. Analyser Data

Most medical analysers have a selection of tests which produce unique sets of process data. The analyser makes these sets of process data available on its output port - invariably a serial data port. Depending on the complexity of the analyser, these data could be presented in fixed length messages or variable length messages. Generally the analysers that perform multiple-channel analyses or random access analyses utilise variable length messages because the results output from the analysis vary according to the sample being processed and the tests selected for it.

Each message consists of a subset of message elements which are unique to each test. Each message subset contains informational fields such as labels and descriptors, and data fields composed of results from actual analytical assays. The exact makeup of each message depends on the message format implemented in the analyser for the different analytical tests.

5.4.1. Fixed Length Messages

This type of message format is typical of small and bedside analysers which do not possess the complexities necessary to perform different tests on different samples within the one sample set. An example is the ABL3 Blood Gas Analyser (Figure 5.2).



Patient Data

```
RADIOMETER ABL330
       20
TIME  11:33
DATE 1996 FEB 18

PTID          12345
TEMP  37.0    C
Hb *  15.0    g%

pH     7.393
PCO2   5.19   kPa
PO2   14.91   kPa

HCO3  23.4    MM/L
TCO2  24.6    MM/L
ABE   -1.0    MM/L
SBE   -1.1    MM/L
SBC   23.6    MM/L

SAT   98.1    %

* PREPROG. VALUE
```

Calibration Data

```
RADIOMETER ABL330
       23
    2 POINT CAL
TIME  11:44
DATE 1996 FEB 18
BARO 101.9   kPa
CORRECTED VALUES
RED CAL
pH     7.382
PCO2   5.36  kPa
PO2   18.88  kPa
GREEN CAL
pH     6.842
PCO2  10.61  kPa
ELECTRODE DRIFT
RED DRIFT
pH     0.001
PCO2   0.01  kPa
PO2    0.03  kPa
GREEN DRIFT
pH     0.000
PCO2  -0.03  kPa
```

Control Data

```
RADIOMETER ABL330
       26
    1 POINT CAL
TIME  11:55
DATE 1996 FEB 18
BARO 101.9   kPa
CORRECTED VALUES
pH     7.381
PCO2   5.36  kPa
PO2   18.89  kPa
ELECTRODE DRIFT
pH     0.000
PCO2  -0.03  kPa
PO2    0.12  kPa
```

Figure 5.2 Samples of process data from an Air Blood Gas Analyser (ABL330)

For each class of analytical test (patient, control, or calibration) the generated message follows a rigid definition of information, value descriptors, values, and value units. They are always in the same layout with the same number of characters for each element of a message subset. This predictability makes it easier to interface to a host computer because the interface module only needs to be sufficiently complex to process this small set of fixed length messages.

5.4.2. Variable Length Messages

This type of message format is typical of analysers that use worklists to define which tests out of a collection of tests will be performed on an individual sample. This encompasses the classes of medium and large sized analysers and an example is the Hitachi 717 Clinical Chemical Analyser from Boehringer Mannheim/Hitachi.

The message sets produced by these types of analysers vary in length depending on the selection of tests to be performed on each sample. Information,

value descriptors, values, and value units are also present in this message format, but the number of fields associated with values may increase. Some samples may only have one test result associated with their entry in the message whilst other samples may have dozens of results. The entry in the message for each sample dynamically resizes to accommodate the varying number of test result entries. Extensive use is made of delimiting characters to break the message subsets into their component fields and this feature is utilised to identify the termination of repeating fields when interfacing to a host computer.

5.5. Analyser Physical Interface

Currently almost all medical instruments have a serial port which supports the transfer of their process data using RS232C or RS485 protocols (Appendix B). Rather than invest in new instrumentation, using the process data available on this serial port as the data assimilation path will allow the leveraging of current technology to support the integration of clinical instruments into a healthcare-wide data management system. As results are produced by clinical analysers in a medical laboratory, they can be assimilated to a local computer using the serial port. Thus the current investment in technology is reused without the need to invest in new hardware interfaces ([KENN – 94]) or new instruments.

Whilst approximately 50% of the serial communication market uses the RS-232 standard, other communication interfaces like RS-422/423, RS-484, RS-485, RS-574, RS-644, and fibre-optic-based standards are also available. The RS-422/423 standard supports data transfer over distances of up to 300m at higher speeds than the RS-232 standard with improved immunity to electrical interference, and is particularly suited to a laboratory environment where it may not be possible to locate a computer close to all the instruments to which it may be connected. There is also a trend in the industry to change to CMOS transistors instead of BJT transistors, and to use 3 V instead of 5 V voltage levels. In 1996 approximately 68% of all RS-232 devices were BJT, but this is expected to drop to 52% by 2000 as CMOS is more widely adopted. Similarly 28% of RS-232 devices in 1996 used 3 V signal levels but that is expected to increase to 60% by 2000 [MANT – 97].

In order to address these trends in the serial communication market, the application development must be modular so that as different communication interfaces are adopted, the hardware and associated driver software can be replaced without impacting on the core design of the data transferring application. If it is deemed necessary to replace RS-232 connections with RS-422 connections in order to facilitate a more distributed laboratory environment, the serial port hardware can be replaced and the driver software easily upgraded without impacting on any other sub-system modules in the data assimilation application.

5.6. Analyser Software Interface

Device driver software for controlling the hardware responsible for this process of data transfer over an electrical medium and the associated hardware and error control logic is readily available. It typically permits the specification of data transfer rates, number of stop bits, number of data bits, type of parity checking, and any hardware flow control which may be utilised during the transaction. The software is a time-proven technology and is usually bundled with all operating systems.

To develop customised driver software is a relatively uncomplicated process since the serial port hardware is not too complex and the necessary functionality to control it is well documented [SERG – 89]. The software can be developed as an integral part of the data transferring application or as an add-on module. This second approach satisfies the modular approach of application development since it allows the development of the customised driver software to be carried out independently of the data transferring application. The add-on module could be a Dynamically Linked Library (DLL) or an object-based class which provide access to serial data communication through a streamlined interface.

5.6.1. Visual Basic Communications Control

For rapid assessment of the feasibility of design choices being made, it was decided to use the Communications Control available in Visual Basic 3.0. This custom control (VBX) is available with the Visual Basic application development environment and simplifies serial communications for applications by allowing the transmission and reception of data through a serial port [MICR – 95]. When the application is distributed, in order to use the Communications Control, the file MSCOMM.VBX must be copied to the Windows \SYSTEM subdirectory of the target system.

```
ctlComm.PortOpen = True   ' try opening the serial port
If (ctlComm.PortOpen = True) Then
    ' successful - try outputting a command string to the serial port
    ctlComm.Output = strCommand
End If
ctlComm.PortOpen = False  ' done, so close the serial port
```

Figure 5.3 VB code sample for interacting with a Communications Control named "ctlComm"

The Communications Control has properties and functions for interacting with it. The main ones are shown in the sample code in Figure 5.3. It has only one event – the "OnComm" event – that must be serviced. The "OnComm" event is generated whenever the value of the CommEvent property changes, indicating that either a communications event or a communications error has occurred. This is typically used for processing any data that is received by the serial port (Figure 5.4).

```
Sub ctlComm_OnComm ()
Dim strRecv As String
    Select Case ctlComm.CommEvent
    'Events
    Case MSCOMM_EV_CD          'change in the CD line
    Case MSCOMM_EV_CTS         'change in the CTS line
    Case MSCOMM_EV_DSR         'change in the DSR line
    Case MSCOMM_EV_RECEIVE     'received characters
        If (ctlComm.InBufferCount > 0) Then
            'if there were characters, read them into a string variable
            strRecv = ctlComm.Input
        End If
    'Errors
    Case MSCOMM_ER_BREAK       'A Break was received
    Case MSCOMM_ER_CDTO        'CD (RLSD) Timeout
    Case MSCOMM_ER_CTSTO       'CTS Timeout
    Case MSCOMM_ER_DSRTO       'DSR Timeout
    Case MSCOMM_ER_TXFULL      'Transmit buffer full
    End Select
End Sub
```

Figure 5.4 VB code sample for the Communications Control event

5.6.2. C Dynamic Link Library

For performance reasons and due to the fact that the custom control file `MSCOMM.VBX` must be distributed with the design solution, a C-based Dynamic Link Library (DLL) was developed. The C language provides greater control over the communication ports at a lower level than Visual Basic with less overhead. Therefore, equivalent functionality invariably executes faster when it is implemented in C compared to a Visual Basic implementation. The resultant DLL does not require any special installation process – it simply needs to be in the same directory as the executable that calls it.

From the functional prototyping achieved using the Visual Basic custom control, it was obvious that a relatively simple Application Programming Interface (API) was sufficient. The C-based DLL provides five interfaces: `Configure`, `Open`, `Close`, `Read`, and `Write` (Figure 5.5). A data structure is also defined for use in configuring and identifying the serial port.

```
extern "C" BOOL FAR PASCAL _export Configure(HWND hCaller,
struct ConfigStruct far *Config);

extern "C" BOOL FAR PASCAL _export Open(HWND hCaller,
struct ConfigStruct far *Config);

extern "C" BOOL FAR PASCAL _export Close(HWND hCaller,
struct ConfigStruct far *Config);

extern "C" WORD FAR PASCAL _export Write(HWND hCaller,
struct ConfigStruct far *Config, void far *Data, unsigned
iCount);

extern "C" WORD FAR PASCAL _export Read(HWND hCaller,
struct ConfigStruct far *Config, void far *Data, unsigned
iCount);

/* Structure of configuration data */
struct ConfigStruct
{
    char   cDeviceID[16];    /* used to identify the type/name of device */
    DWORD  dwConfigDataLen;  /* overall length of this structure */
    DCB    DCBinfo;          /* standard 53-byte Windows DCB used here */
    HWND   hCaller;          /* hWnd to the caller */
    int    iDeviceNum;       /* used to identify multiple devices */
};
```

Figure 5.5 C function definitions for a serial communications DLL

In order for these interfaces to be used by a Visual Basic application, they must be declared in a manner that Visual Basic can recognise (*Figure 5.6*). "`Alias`" is used to identify the name of the procedure in the DLL for use in the Visual Basic domain. This prevents the external procedure name conflicting with a Visual Basic reserved word, a global variable or constant, or any other procedure in the same scope. Thus the five C-based DLL interfaces are renamed to: `ConfigPort`, `OpenPort`, `ClosePort`, `ReadPort`, and `WritePort`. A data structure is also defined for use in configuring and identifying the serial port.

```
Declare Function ConfigPort Lib "RS232.dll" Alias "Configure"
(ByVal HWND As Integer, pConfig As ConfigStruct) As Integer

Declare Function OpenPort Lib "RS232.dll" Alias "Open" (ByVal
HWND As Integer, pConfig As ConfigStruct) As Integer

Declare Function ClosePort Lib "RS232.dll" Alias "Close"
(ByVal HWND As Integer, pConfig As ConfigStruct) As Integer

Declare Function ReadPort Lib "RS232.dll" Alias "Read" (ByVal
HWND As Integer, pConfig As ConfigStruct, ByVal strBuffer As
String, ByVal iBufSize As Integer) As Integer

Declare Function WritePort Lib "RS232.dll" Alias "Write"
(ByVal HWND As Integer, pConfig As ConfigStruct, ByVal
strBuffer As String, ByVal iBufSize As Integer) As Integer

Type ConfigStruct
    strDeviceID As String * 16 ' used to identify the type/name of device
    ConfigDataLen As Long      ' overall length of this structure
    DCBinfo As String * 55     ' standard 53-byte Windows DCB structure
    hCaller As Integer         ' hWnd to the caller
    iDeviceNum As Integer      ' used to identify multiple devices
End Type
```

Figure 5.6 VB declarations for calling functions from a DLL API

A simple Visual Basic code sample in Figure 5.7 shows how the interfaces of the C-based DLL can be invoked and used.

```
' use the DLL's configuration dialog box
If (ConfigPort(HWND, pConfig) = False) Then
   MsgBox "Error configuring the Port", 48
Else
   ' try opening the serial port
   If (OpenPort(HWND, pConfig) = False) Then
      MsgBox "Error opening the Port", 48
   Else
      ' try outputting a command string to the serial port
      bRetVal = WritePort(HWND, pConfig, strCommand,
Len(strCommand))
      ' done, so close the serial port
      If (ClosePort(HWND, pConfig) = False) Then
         MsgBox "Error closing the Port", 48
      End If
   End If
End If
```

Figure 5.7 VB code sample for invoking a serial communications DLL

5.7. Analyser Message Interpretation

Most medical instruments have a selection of tests which produce unique sets of process data. These tests can be one of two types: either sample analysis or calibration procedures to assist in the technical validation process of a Quality Assurance scheme. Most medical instruments provide serial transmission of process data as a stream of characters arranged to give fixed length message sets. Each message consists of subset message elements which are unique to each test. Whilst each message subset can contain informational fields and data fields composed of varying data, the length of each message subset is a fixed number of characters. A selection of sample data output from an ABL 330 Blood Gas Analyser is shown in Figure 5.2. This property can be exploited to identify the specific analytical procedure which generated the process data being assimilated.

As each message is transferred via the serial data connection to the PC, the analytical procedure which generated it must be identified in order that the data elements can be correctly assimilated according to the context of the generating analytical procedure.

In the absence of any adopted medical instrument messaging standard, the analytical procedure identification process is unique to each instrument since the message content is unique to the process data output by the instrument for each analytical procedure. Therefore the identification rules utilised in this process are unique to each instrument and must be generated as required. This is not an easily automated process and necessitates human interaction. Once the identification rules have been correctly defined, the assimilation of process data can be repeatedly iterated on an automated basis. If a different instrument is connected to the PC, then the identification rules need to be redefined for the message syntax unique to that instrument.

5.7.1. General Parsing Techniques

The identification rules are composed of two parts. The first part operates on a data stream to break it into discrete components which are recognisable as multiple

unit entities of message subsets. This division into units (which are usually called tokens) is known as lexical analysis. The module performing this lexical analysis can be custom-coded in any general-purpose programming language, such as C, to process the unique message content of a specific instrument. Alternatively a lexical analyser generation tool, such as Flex, could be used. Flex has its roots in the UNIX-based Lex [LESK – 75] which resulted from research in the area of Finite State Techniques [JOHN – 68] as applied to scanning and parsing theory [AAHO – 86][MORT – 93]. Flex requires a description of the input stream content (using a special-purpose language for writing lexical analysers) and produces a lexical analyser that is both fast and compact. The resultant lexical analyser is almost always faster than one which has been custom-coded due to the various algorithmic optimisations implemented by the Flex tool [LEVI – 92]. It is usually used for generating compilers.

The second component utilises the tokens generated by the lexical analyser and compares them to grammar rules. The grammar rules define the context to which the tokens are to be applied and therefore the sequence in which they arrive from the lexical analyser is critical. This process is known as parsing. As a sequence of tokens is logically matched to a specific grammar rule, a user-defined action can be carried out. This user-defined action could be a simple single code statement or a function call which passes temporary control to some other processing module. As with the lexical analyser, the parser could be custom-coded or else a parser generation tool could be used such as the UNIX-based YACC (Yet Another Compiler Compiler), or its modern counterpart Bison. Whilst the YACC-generated parser is usually not as fast as a custom-coded parser [LEVI – 92], the ease in writing and modifying the parser is invariably worth any speed loss incurred and it uses sophisticated parsing algorithms derived from the article [KRIS – 81].

Two different rule generation techniques were examined which performed real-time analytical procedure identification on the stream of process data received from the medical instrument over serial communications. Both techniques use a lexical analyser to perform a lexical analysis on the data input stream and break it up into a sequence of tokens. This sequence of tokens is then compared against a

collection of grammar rules in a parser. Each grammar rule set is specific to an analytical procedure, thereby enabling its identification. The difference between the two techniques is based on how the lexical analysis and grammar rules are generated for each instrument being interfaced to produce a lexical analyser and parser pair.

5.7.2. Advanced Instrument Interface (A.I.I.)

The first technique utilised software developed for the OpenLabs project, which was an EU Advanced Informatics in Medicine (AIM) research initiative to study the area of medical informatics [OPEN – 95]. The software generated the lexical analyser and parser using a set of tools in a graphical environment (Figure 5.8).

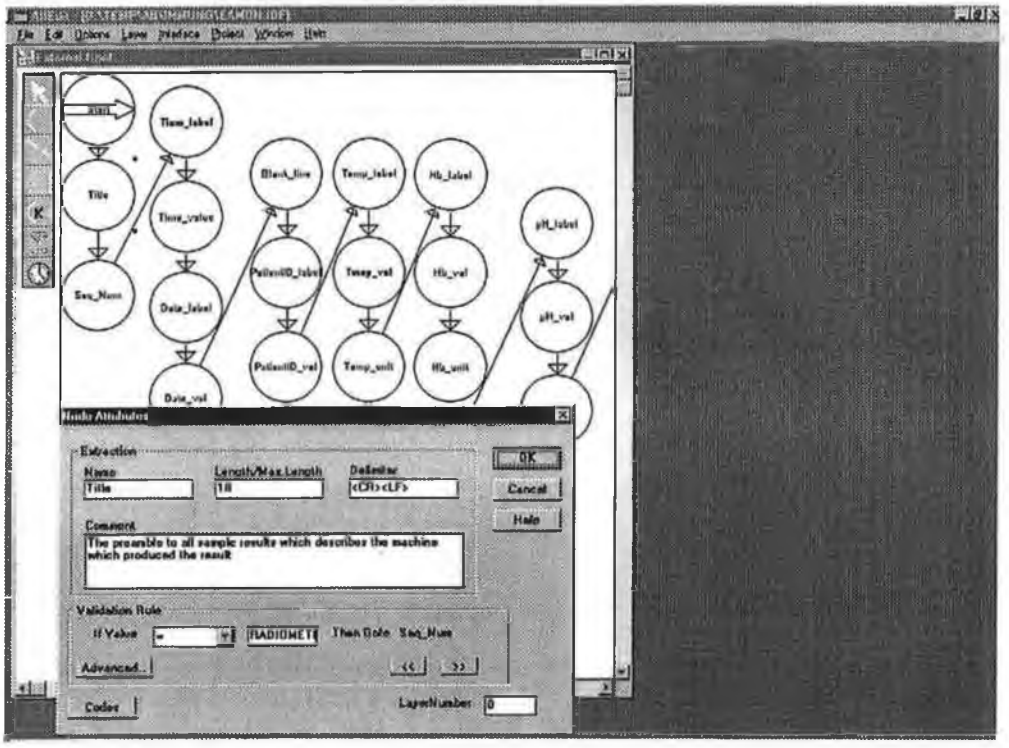


Figure 5.8 Advanced Instrument Interface (AII) development environment

Tokens were represented in a simple graphical manner and their relationships with each other were represented as lines connecting different tokens in specific orders to produce the grammar rule set specific to different analytical procedures

for each instrument. Each token has attributes associated with it such as an identifying label, its length, delimiter character(s), and validation rule(s) for grammatical integrity. These attributes dictate its interaction with other tokens in the grammar analysis.

The software has several plug-in interfaces for connectivity to various data sources and data sinks. These include RS-232 for serial communication, TCP/IP for network communication, and Dynamic Data Exchange (DDE) for DDE-compliant inter-application communication.

The graphical-based definition of tokens enables rapid and easy development of a lexical analyser and parser pair. Specification of token attributes and additional grammatical rules is achieved with a few dialog boxes. The architecture permits the design of new plug-in interface modules to satisfy unique requirements as they are encountered, resulting in an extensible development environment. Integrated debugging features display messages indicating the status of parsing as the grammar rule sets are navigated. This enables rapid prototype development. The resultant run-time interface application can support multiple data streams from multiple sources going to multiple destinations all independently of one another. However there is a performance penalty for the flexibility of this run-time interface application. The processing overhead can result in sluggish performance for data transfer rates above baud rates of 9600. Whilst most medical instruments have a range of data transfer rates, they usually do not have serial communication capabilities at those higher data transfer rates, so this system would be sufficient to interface an instrument to a computer. If network communication is also required, then this may introduce too many time delays to make it any more realistic than a prototype development.

5.7.3. Static Parser Generator

The second technique was custom designed for this research using Visual Basic to exploit its powerful user interface creation abilities and rapid development and ratification cycles (Figure 5.9). It was designed to exploit the typical format of

serial data from simple medical analysers in the low-end of the market. Most of these analysers output a relatively uncomplicated stream of data sets which do not require complex processing before assimilation. These streams tend to be static in structure with fixed field sizes and no dynamically generated content. For example, the content of a calibration data set always contains the same message subsets. Since low-end analysers are more prolific in the medical environment than the more expensive high-end analysers, it was decided to target their serial data communication templates for design guidelines when developing this parser generator.

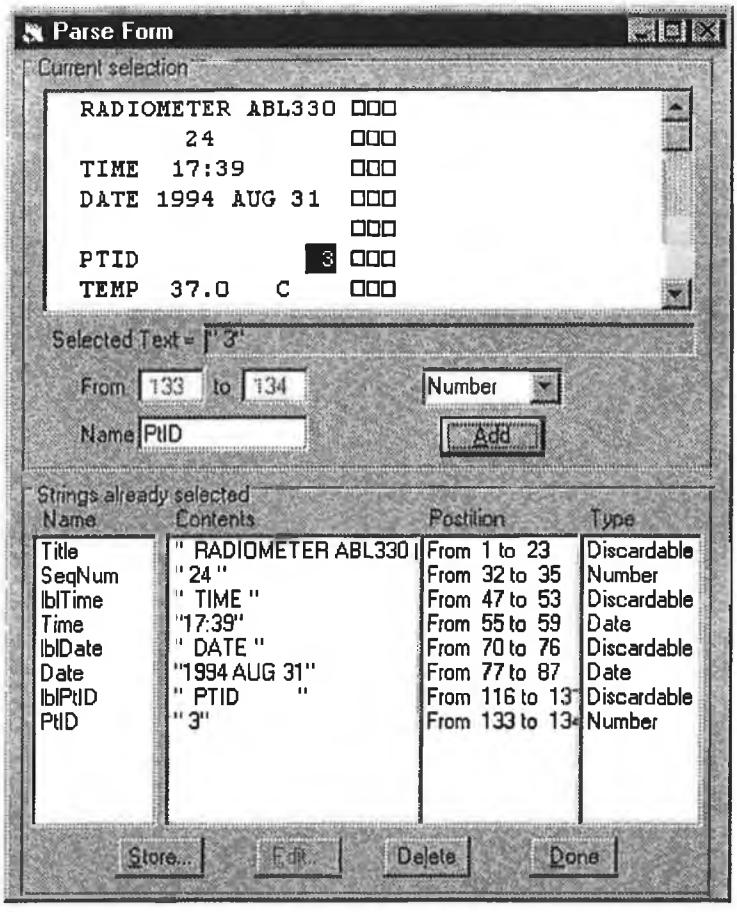


Figure 5.9 Parser being "trained" for message sets from an ABL 330

The model adopted was vaguely inspired by the methodologies used in neural networks where the net was "trained" with known inputs to give an improved probability of a correct output when presented with an unknown input. Thus the parser was "trained" with a selection of typical analytical procedure message sets

to generate the rule sets for the lexical analyser and parser. These rule sets were utilised by the parser to identify unknown analytical procedure message sets.

Training is performed by specifying the operational mode as being "training". The instrument then transmits sample messages for each of the operational procedures whose data are required to be assimilated. The messages are transmitted to the host computer over the serial communication link as separate data streams. Each data stream is extracted from the serial communication module to reproduce the original message. The message is then presented to the user permitting the identification of the discrete units of the message subsets within the message by a simple point-and-click interface. The message subsets are given appropriate attributes to convert them to tokens for lexical analysis and to define their location within the grammatical structure of the message. Due to the simple format of the messages from the targeted low-end analysers, the location within the grammatical structure is simply the location in the stream of data. Hence the location attribute specifies the character count up to the start character of the message subset and the number of characters within it.

Once the "training" has been completed for all the analytical procedures of interest, the operational mode is changed to "parse". As serial data is received from the connected medical instrument, the message data is extracted and passed to the lexical analyser and parser pair for processing. These modules attempt to match the message data to a grammar rule by iterating through all the pertinent grammar rules available to it. When a match has been found the analytical procedure which created the message data has been identified so that the subset information can be used in the correct context. Thus, for example, result data from patient sample analyses are made available to Technical Validation techniques whilst result data from control sample analyses are made available to Quality Assurance procedures.

5.7.4. Adopted Module

Since most of the clinical analysers available in the test environment are of low-end variety, they generate data sets of fixed length and predictable content.

Therefore, it was decided to utilise the "static parser generator" as the analyser message interpretation module. It provides the necessary flexibility and ease of use to suit the user requirements dictated by the clinical laboratory environment.

Both systems follow the same process. As the lexical analysis identifies required message elements, they are paired with their assigned name in a multi-dimensional array. Any data that is attributed a "discardable" type is ignored for further processing. The resultant two-dimensional array has a name and value pair for each message element.

During configuration of the lexical rules, the name assigned to a message element is critical. Since the data will ultimately be stored in a database, the message element name corresponds to the field name of the table to which it will be stored. Thus, they must match exactly. The two-dimensional array is passed to its associated Transaction Agent in the Instrument Interface module. There it is used to generate a SQL statement that will result in it being stored in a database. The message element names are used as the "field" parameters whilst the data values are used as the "value" parameters in an INSERT statement (see Section 7.1 for a discussion of SQL and Appendix D for the specific SQL syntax). For example, "`INSERT INTO PatientData (SeqNum, Time, Date, PtID, Temp, Hb) VALUES ('20', '11:33', '1996 Feb 18', '12345', '37.0', '15.0')`" (data taken from Figure 5.2). If the message element names do not concisely match the names of fields in the table of the target database, the SQL statement will fail.

At configuration time, the successful operation depends upon accurately specifying the message element names. If errors become apparent at execution time, the "static parser generator" facilitates correcting the name assignments with minimal difficulty. In fact, it is not necessary to repeat the "training" process since the rule set can be edited and updated as required. Once the necessary modifications have been completed, the rule set just needs to be saved before returning to the "parse" mode of operation.

6. Transaction Architecture

A design concept based on a Client/Server communication model inherently utilises discrete parcels of data to carry out units of work. Each unit of work is referred to as a transaction, which is essentially a negotiation between a Client and a Server using a prearranged syntax to deliver some functionality. The syntax could be composed of specific predefined requests which are constant and follow a rigid pre-defined format, or they could be more complex permitting the dynamic generation of requests which may never have been encountered before, but which must be correctly interpreted according to their context.

The syntax used for the transactions must be defined in accordance with the units of work to be performed. In the context of this design concept the units of work are service procedures. In order to invoke a service procedure a transaction must be dispatched to the service provider via the Transaction Processor. The transaction requires the appropriate data formatting when it is transmitted in order that the service provider can correctly interpret the service request and utilise the appropriate procedure to deliver the correct service (Figure 6.1).

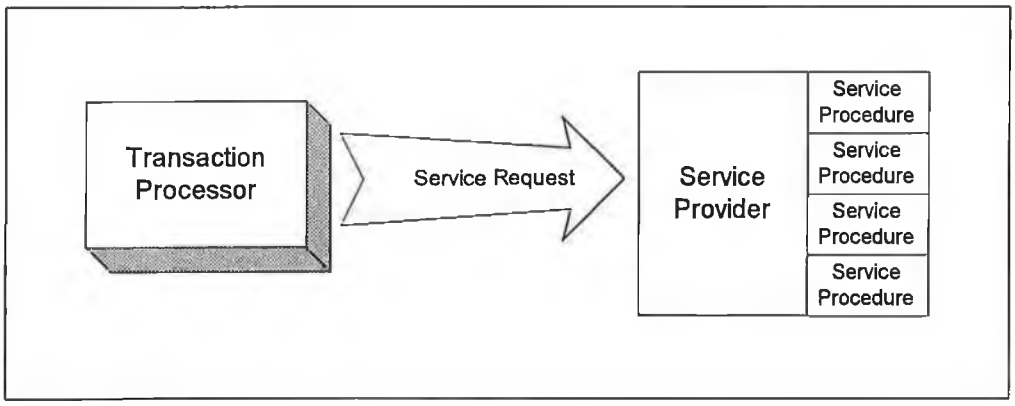


Figure 6.1 The infrastructure for transaction processing

For the proposed 3-tier Client/Server communication paradigm a middleware layer of functionality will be provided by the Transaction Processor. The Transaction Processor will route all the transactions from client modules to the

appropriate service providers and provide additional processing and administration of transactions.

When client modules are initialised (for example Instrument Interface modules), they register with the Transaction Processor by providing identifying information in the form of some brief user-entered text. This text is used for user-interface and logging purposes. The Transaction Processor consequently supplies each client module with a unique identifier code appropriate to their function. Each unique identifier code (Module Identification Code, MIC) issued to a client module at registration is composed of a functional code (see Table 6.1) and a number. The number is incremented by the Transaction Processor for each newly registered module of the same functional type. So a "host" with 3 medical analysers connected to it would have 3 instrument interface modules registered with its Transaction Processor each having MIC's of INS1, INS2, and INS3. This scheme of centralising the registration process reduces the arbitration problems as new client modules are initiated at run time and simplifies the tracking and routing of transactions between clients and servers.

Function Code	Description
INS	Instrument interface module
GUI	Graphical User Interface module
DBS	Database interfacing module
CHT	Chat facility
TUT	Tutorial facility

Table 6.1 Function codes for module registration

Each client module is responsible for identifying all transactions it generates by incrementing a counter and using the counter as a Transaction Identification Number (TIN). With the MIC combined with the TIN, the Transaction Processor can uniquely identify all transactions originating from its locally connected client modules.

In order to define the necessary transactions and their components, the service domains were analysed and their component individual services procedures identified. The service domains revolve around two general areas: administering a transactional link and exchanging data over that link.

Administering a transactional link involves:

- opening and closing it
- negotiating identities
- establishing privileges
- obtaining additional information
- enquiring the status

Exchanging data includes a variety of data from a number of services:

- data storage
- data retrieval
- interactive chat
- tutorial instruction

Since the focus of the design concept is data processing based on storing and retrieving data, the majority of transactions will be concerned with exchanging data of the first two types. The last two data types are included to permit the exploitation of expert knowledge by the operator of an analyser which is located remotely from the medical laboratory.

6.1. Transaction Structure

It is proposed to construct transactions that are composed of two sections – a header and a body. The header section encapsulates three types of fields – a "command" field, an occasional "qualifier" field and several "administrative" fields, whilst the body section encapsulates a "message" field (Figure 6.2).

The "command" field specifies the service procedure being requested (as outlined earlier and described in more detail later), while the "qualifier" field provides additional information to instruct how, or on what, to perform the service procedure. Obviously some service procedures will not need any qualifiers in order for a service provider to completely deliver the requested service. Typically, service procedures located on remote computers would be the most frequent users of this "qualifier" field where it would contain the IP address of the remote computer. The "administrative" fields contain information to uniquely identify which client module originated the transaction (the MIC) and the transaction itself (the TIN), the time it was requested, the overall length of the transaction, and a delimiter.

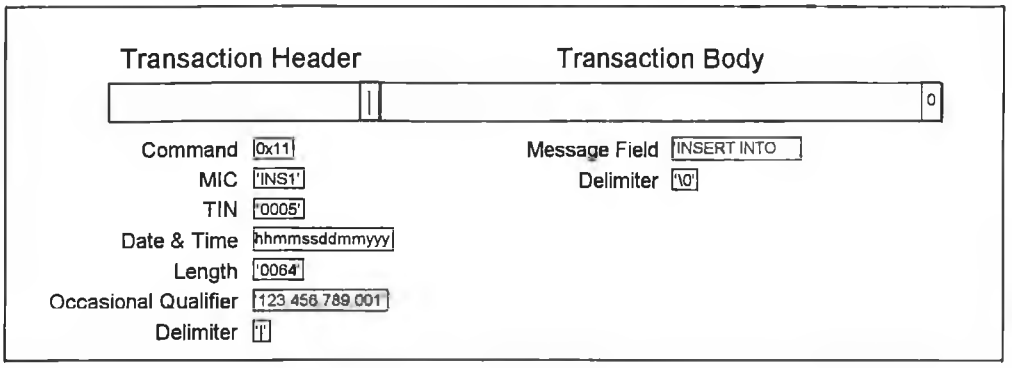


Figure 6.2 The component fields of a transaction

The "message" field contains the raw data which will be used by the service provider in the service procedure and can vary from potentially no characters to several thousand, depending on the type of service being requested. As identified in the user requirements (Chapter 3), the majority of transactions will be database centric. Therefore the "message" field would typically contain the text-based SQL statement necessary to fulfil the required database activity.

The header contains a fixed-length (2 bytes) hexadecimal number representing the transaction type command code (Appendix D). In the next 8 bytes are the fixed-length MIC (4 bytes) and TIN (4 bytes), used for uniquely identifying the originating client module. A 14-byte character representation of the date and time stamp follows, which has the format "hhmmssddmmyyyy". Thus 14:52:10 on the

2nd of April in 1992 appears as "14521002041992". The next field is also fixed-length (4 bytes) and contains a hexadecimal number representing the total count of bytes in the complete transaction including the header. Therefore the maximum theoretical length is 0xFFFF = 65,535 bytes. Following this is an optional field (called a "qualifier"), the content of which depends on the type of transaction (see Appendix D). In order to identify when the header is complete and the body of the transaction is starting, a delimiter is used as the final field and is defined as the "|"character. This is necessary because the optional qualifier field has a variable length which can be zero or greater than zero. The body of the transaction also has a delimiter that it is defined as the NULL character (0x00) because the message is typically text (formatted as a SQL statement), and NULL is widely accepted as a terminator for text.

Transactions are based on a request-response pairing. For every transaction request there should be a corresponding transaction response. When a client module requires a service, it generates a transaction with its MIC and the TIN appended together to uniquely identify the transaction. This is placed in the designated field of the header along with the command code and other necessary information to provide context (Appendix D). This transaction is dispatched to the Transaction Processor and routed through to the service provider. The response from the service provider contains the same unique identification as the request (the original combination of MIC and TIN), as well as the requesting command code placed in the "qualifier" field, and a command code indicates the success or failure of the transaction. This is sent back to the client module via the Transaction Processor. The unique identification of each request-response transaction pair allows the Transaction Processor to match the response to each individual request and permits the tracking of incomplete transactions. Since all transactions contain a time-stamp, matching the response to the request also gives an indication of the time taken to complete transactions. Excessively slow completion of transactions would therefore be highlighted and corrective action could be taken. The transaction response could contain a simple acknowledgement code, a substantial data set, or a transaction failure based on the output of the transaction service procedure.

6.2. Transaction Processor

The Transaction Processor provides the middleware functionality in the adopted 3-tier Client/Server communication model. Its purpose is to provide coherence to the transaction-based messaging system by centralising the control and administration of transactions. A functional representation of the internal components is shown in Figure 6.3 and they are explained below.

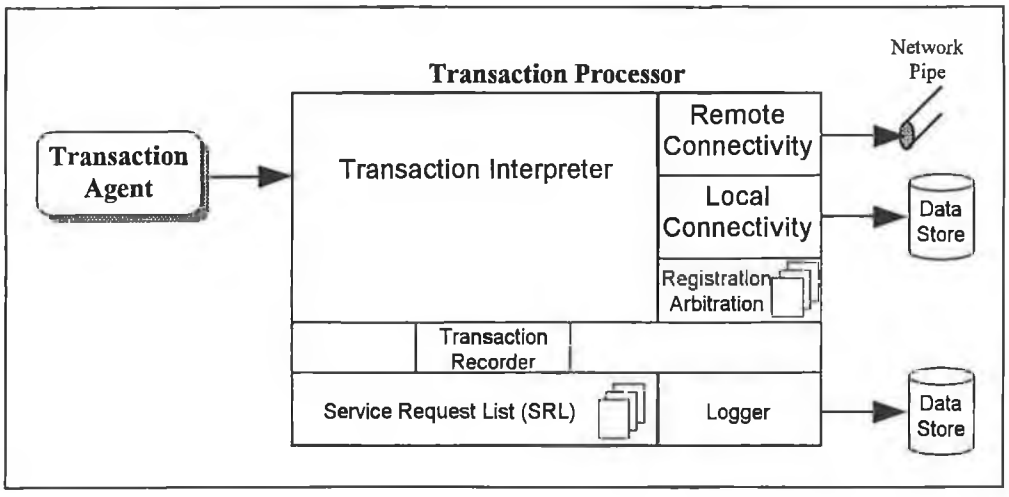


Figure 6.3 Transaction Processor internals

Each "host" has its own Transaction Processor to arbitrate over internal transactions. Each module (client modules and service provider modules) on the "host" must register with the Transaction Processor so that it knows how to route transactions. The Transaction Processor is able to redirect every transaction to the correct service provider based on the command code and any qualifiers associated with it.

For example, when an instrument interface module needs to store data in a database, its Transaction Agent generates a transaction which the Transaction Processor routes to the database connectivity service provider. This routing is based on the command code in the transaction header and the fact that the database connectivity service provider module has previously registered with the

Transaction Provider. Even a remote database can be referenced with this mechanism since the transaction header command code would indicate a remote database operation and the "qualifier" field would specify the IP address of the remote database server.

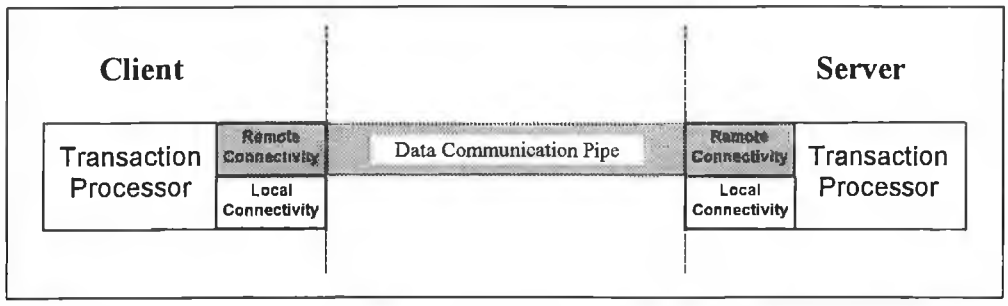


Figure 6.4 Extending the Transaction Processor to a remote server

For transactions to a remote computer (for example a centralised QA server), the Transaction Processor conceptually extends over the data communication medium to include a counterpart Transaction Processor on the server, producing one expansive virtual Transaction Processor (Figure 6.4). Therefore the "host" Transaction Processor establishes a data communication link with the remote Transaction Processor, and passes the transaction to the remote Transaction Processor. This remote Transaction Processor then arbitrates and dispatches the transaction to the required service provider as if it had been generated locally. Response transactions from the service provider are passed to the "host" Transaction Processor by the remote Transaction Processor via the data communication link and routed to the requesting client module. Isolating the data communication functionality in this manner results in transparent transaction processing whether the service provider is located on the local computer or on a remote computer.

Whenever it receives a service request, the Transaction Processor appends an entry to the end of a "Service Request List" (SRL). The SRL entry is composed of two sections – a request section and a response section – to mimic the request-response pairing of transactions (Figure 6.5).

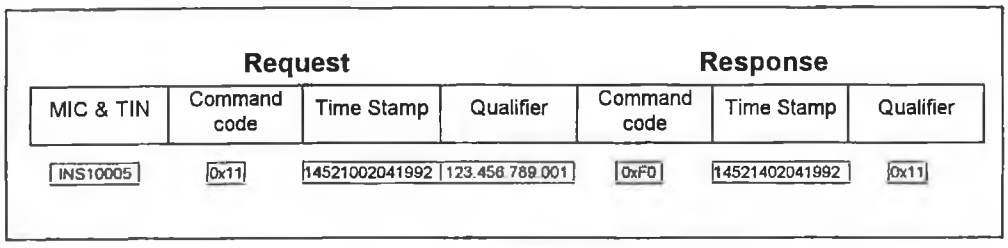


Figure 6.5 The elements of a Service Request List (SRL) entry

The request section contains the unique MIC and TIN, the transaction command code, the time-stamp, and the "qualifier" field (if any) of the transaction. These four elements are obtained from information contained within the header of the transaction (Appendix C). The response section contains the transaction command code, the time-stamp, and the "qualifier" field (if any) of the response transaction. It is filled out using information contained within the header of the response transaction when it is received from the service provider. The time-stamps permit the identification of time-intensive transactions or sluggish transaction processing, whilst an unmatched time-stamp could be used to identify incomplete transaction and provide a time-out to the requesting client module.

As each request-response transaction pair is matched up, it indicates a completed transaction, successful or otherwise. At this point the entry is removed from the SRL and can be either discarded or else logged to a report file (Figure 6.6). The logging information could be an exact duplicate of the SRL entry or else a verbose translation of the SRL entry, depending on configuration data supplied at start-up. Thus the time stamps and transaction codes would be translated by the logging module within the Transaction Processor from definitions in a look-up table and the MIC would be replaced with information supplied at the time of the module's registration.

Default formatting of SRL output to log file

```
INS10005  0x02      14520002041992      123.456.789.001      0xF0      14520902041992      0x02
INS10005  0x11      14521002041992      123.456.789.001      0xF0      14521402041992      0x11
INS10005  0x11      14521502041992      123.456.789.001      0xF0      14521902041992      0x11
INS10005  0x11      14522002041992      123.456.789.001      0xF0      14522402041992      0x11
INS10005  0x11      14522502041992      123.456.789.001      0xF0      14522902041992      0x11
INS10005  0x07      14523002041992      123.456.789.001      0xF1      14523102041992      0x07
```

Verbose formatting of SRL output to log file

```
 lood Gas Ana. in Room 34
OPEN_DB_REMOTE       14:52:00 02-Apr-1992          123.456.789.001
          ACK        14:52:09 02-Apr-1992          OPEN_DB_REMOTE
Blood Gas Ana. in Room 34
DB_STORE             14:52:10 02-Apr-1992          123.456.789.001
          ACK        14:52:14 02-Apr-1992          DB_STORE
Blood Gas Ana. in Room 34
DB_STORE             14:52:15 02-Apr-1992          123.456.789.001
          ACK        14:52:19 02-Apr-1992          DB_STORE
Blood Gas Ana. in Room 34
DB_STORE             14:52:20 02-Apr-1992          123.456.789.001
          ACK        14:52:24 02-Apr-1992          DB_STORE
 lood Gas Ana. in Room 34
DB_STORE             14:52:25 02-Apr-1992          123.456.789.001
          ACK        14:52:29 02-Apr-1992          DB_STORE
 lood Gas Ana. in Room 34
CLOSE_LINK           14:52:30 02-Apr-1992          123.456.789.001
          ACK        14:52:31 02-Apr-1992          CLOSE_LINK
```

Figure 6.6 Sample outputs to the Service Request List (SRL) log file

An example of the flow of transactions between a client module and a service provider is shown in Figure 6.7. In this example, the client module is requesting a database to be opened for some pending database activity. The request transaction and its acknowledgement transaction are shown, as is the syntax for the interaction with the SRL.

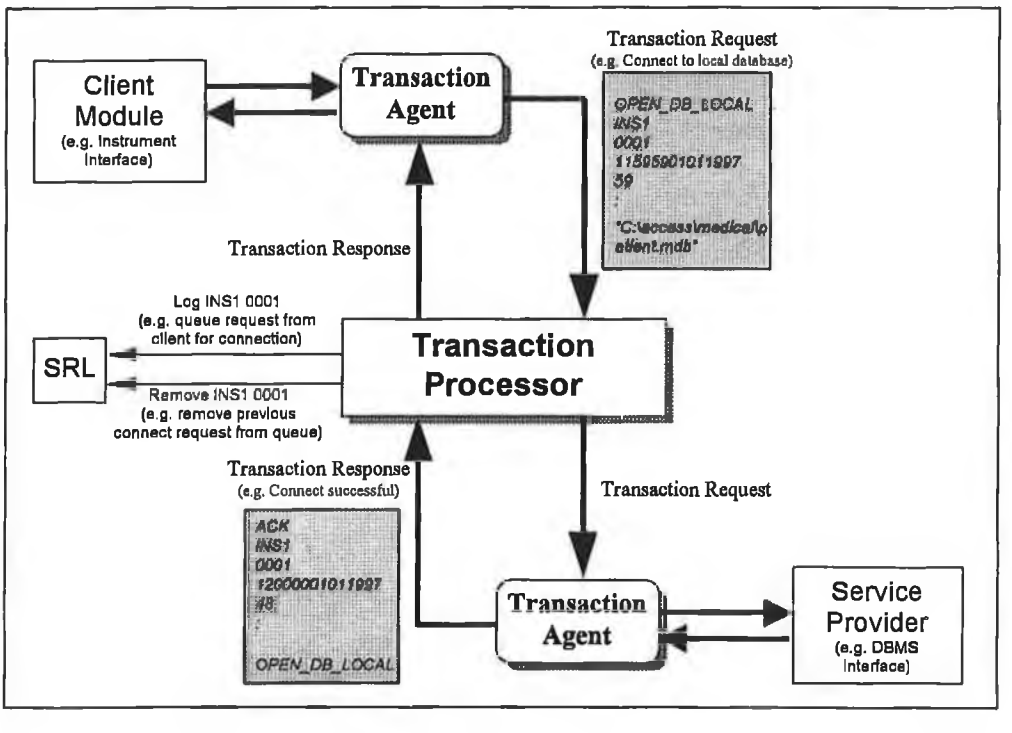


Figure 6.7 Example Transaction flow for opening a database

6.3. Service Procedures

As stated above, service procedures belong to one of two generalised domains. The procedures belonging to the "Administrating the Link" domain are necessary for establishing the transaction link and maintaining it in order to support the procedures of the second domain of "Exchanging Data". The procedures are listed in their entirety in Appendix D.

The addition of new message types (e.g. binary X-ray images) will not affect the transaction structure. This is because they will simply be additional types with their own unique message content and will therefore not require a redefinition of the transaction structure. As new functionality is added to the transaction brokering system implemented within the Transaction Processor it may fall under the category of an existing service procedure class and therefore also scale linearly. However, it is recognised that some new functionalities may require an extension to the functions available within the Transaction Processor.

6.3.1. Opening and Closing the Link

All data exchanges must be initiated before they are committed and subsequently completed. This class of services enable the Transaction Processor to identify the appropriate service provider and ensure that it is in a stable state and is prepared to deliver the requested service. The type of data that is exchanged over the transactional link defines the exact content of the opening and closing service syntax, and these are listed below:

Local database activity (OPEN_DB_LOCAL)	Data being exchanged will be stored or retrieved from a local database
Remote database activity (OPEN_DB_REMOTE)	Data being exchanged will be stored or retrieved from a remote database
Interactive chat (OPEN_CHAT)	Data will be exchanged in a real-time peer-to-peer link between operators
Tutorial information (OPEN_TUTOR)	Data being exchanged will request tutorial information such as a specific tutorial, or a list of archived informational topics

The last three initiate transaction links with remote computers and must therefore include the appropriate information to establish a data communication session. Since the underlying remote communication protocol is TCP/IP, the IP address of the target computer must be given in the "qualifier" field of the header section of the transaction to represent the destination.

If the service provider is capable of facilitating the service request then it sends an acknowledgement with the transaction command code echoed in the "qualifier" field of the transaction header. This allows the Transaction Processor to ratify the acknowledgement with the request. If the two do not match then the Transaction Processor closes down the link and sends a failed transaction message to the requesting client module.

To close a transaction link, either party can issue a close command. There are two types of close commands – one graceful and one an emergency.

Graceful shutdown (CLOSE_LINK)	Communication link shutdown request
Emergency shutdown (TERMINATE)	Emergency communication link shutdown

The graceful one (CLOSE_LINK) requires an acknowledgement from the other party in the transaction which echoes the CLOSE_LINK command in its "qualifier" field. The other close command (TERMINATE) indicates that an emergency state has occurred where it is not possible to wait for the acknowledgement associated with a graceful shut down. When a TERMINATE command is received, the other party must recover as best it can.

6.3.2. Privileges and Security Issues

In a distributed laboratory environment there would be numerous analysers capable of participating in the Laboratory Information System (LIS). It is desirable to uniquely identify each analyser and its "host" computer, as well as whatever

"server" computers may be connected to the LIS to provide data management services. This would allow nodes on the LIS to express the types of services they are capable of delivering. Thus an analyser would reply to this type of transaction with its identity, its location, any other designated parameters necessary for identifying it in the LIS, and a list of the analytical assays it is capable of performing. Similarly, a server would respond with designated information and the class of services it is capable of delivering, for example patient or sample identification, Long Term Quality Control (LTQC) of operational performance, or technical validation of samples.

Identification (WHO_ARE_YOU)	Identification request that also requires supported services

The range of services, which must necessarily be included in this type of informational exchange, can be quite extensive. Whilst each service could be uniquely identified using a proprietary coding mechanism unique to the laboratory, there is substantial continuing research in the area of standard medical nomenclature [BOAR – 94][CAMP – 94][COTE – 73] as well as published standards (SNOMED, etc.). One of these standards should be used to define the analytical services for this type of informational exchange.

In the data-sensitive medical environment, it is imperative that security issues are addressed. Only services with the necessary privileges will be allowed access to data. This is an attempt to eliminate unauthorised use and abuse of critical medical information. It also enables auditing procedures to track data and error sources.

6.3.3. Obtaining Additional Information

This category of services is included to provide future informational features as they are required. For example, as audit trails are ratified it may become necessary to explicitly log data transactions or provide digital signatures. This information could be made available through transactions based on this category.

6.3.4. Enquiring the Status

This simple service procedure allows clients and servers to obtain information about the status of another machine. It could be used as the precursor to the delivery of a critical transaction in order to increase its chances of successfully completing.

Status enquiry (WHAT_STATUS)	Request for the status of a computer

The command (WHAT_STATUS) does not require any message component and the acknowledgement command (STATUS) has a message which indicates its status. The defined statuses are listed below:

Available	Can receive and process transactions
Busy	Currently performing a service
Off-line	Services are not currently available
No answer	Server inoperative or data communication problem

If, for example, a host is asked its status, it could be ready for transaction processing (and therefore give its acknowledgement status as "available"). Alternatively it could be involved with an analyser assisting in performing a sample analysis (and give its acknowledgement status as "busy").

6.3.5. Data Storage and Retrieval

This category of service procedures contains elements that are to be transferred to and from databases. It is anticipated that these procedures will account for the majority of transaction traffic in the final implementation of the design concept. In order to minimise interface complications the transaction will use a syntax which is most easily recognised by the vast majority of databases - Standard Querying Language (SQL) [GROF – 94] - as the contents of the message element.

Whilst this standard is very mature with comprehensive support throughout the DBMS vendors, the standard permits some implementation-specific interpretations and has resulted in some variance from vendor to vendor. Consequently, not all feature sets are directly transferable between vendors systems. To avoid these incompatibilities transactions will be based on the simple sub-set of SQL elements which are in the category of "entry level SQL" [DATE – 93] and are supported by the vast majority of DBMS's (covered in Appendix D).

Assuming suitable permissions have been authenticated (by the "open" transactions used for establishing the link – Sections 6.3.1 and 6.3.2), these SQL statements will store data to existing databases and extract required data from existing databases. No database structures will be altered or generated due to any transaction and all databases are expected to have been previously defined and created. It is the responsibility of the module that generates the transaction (the Transaction Agent) to ensure the accuracy of any SQL statement which it uses in a transaction. The outcome of the SQL statement will be returned to the requesting Transaction Agent, thus it is also the responsibility of the requesting Transaction Agent to handle any subsequent errors due to the failure of SQL statements.

These criteria will distribute some of the processing load to the client modules and will subsequently limit the overhead on the database server, thereby allowing the support of more transactions than would otherwise be possible. This is of particular importance in a distributed environment where several client computers may be trying to simultaneously interact with one central server. As the overhead to deal with a single transaction on the server increases, the volume of transactions that can be serviced consequently reduces. Keeping this overhead to a minimum therefore increases the number of clients a single server can support. This kind of scalability issue becomes a practical concern as increasing numbers of instruments are connected to a central server [BENS – 80][MARK – 88].

Data storage (DB_STORE)	Data will be stored to a SQL-compatible database using an "INSERT INTO ..." statement
Data retrieval (DB_EXTRACT)	Data will be retrieved from a SQL-compatible database using a "SELECT FROM ..." statement

It is desirable that data will be stored to databases whenever it is generated. Typically, this data originates from an instrument connected to a "host" after it has completed a sample analysis. The results of the test(s) are assimilated by the instrument interface module and made available to its Transaction Agent. The Transaction Agent generates a service request to store the results in a predefined database. It does this by an appropriately formatted SQL statement (see Appendix D), and dispatching it to the Transaction Processor for further processing. The Transaction Processor delivers the service request to the appropriate service provider (in this case, a SQL-servicing mechanism) which stores the data and delivers an acknowledgement transaction to indicate the successful completion of the service request (Figure 6.8).

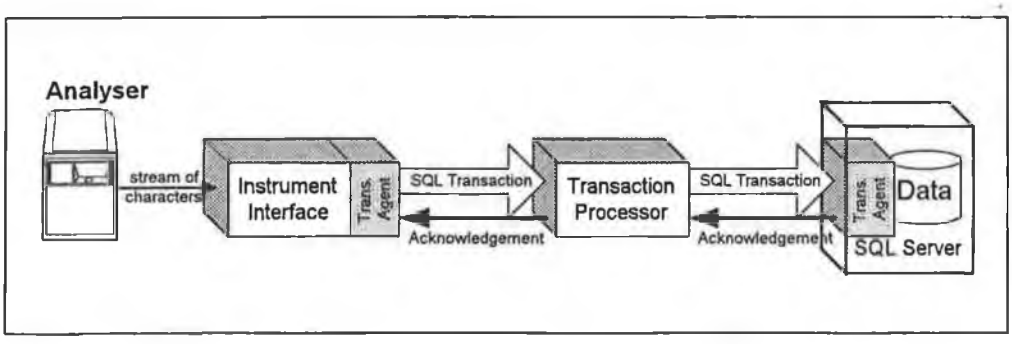


Figure 6.8 Transaction flow for instrument data

Data is retrieved from databases whenever it is required for contributory analysis. This could be QA analysis (for example patient demographic information for technical validation or LTQC of the operational performance of the instrument), or a medical professional using the results to assist in the diagnosis and treatment of a patient. In each case the requesting module (analysis tool or user interface display) issues the requirements to its Transaction Agent which

generates the appropriate service request based on a SQL statement. The Transaction Agent dispatches the transaction to the Transaction Processor which in turn delivers it to the appropriate service provider. The transaction acknowledgement contains the result(s) of the SQL statement (Figure 6.9).

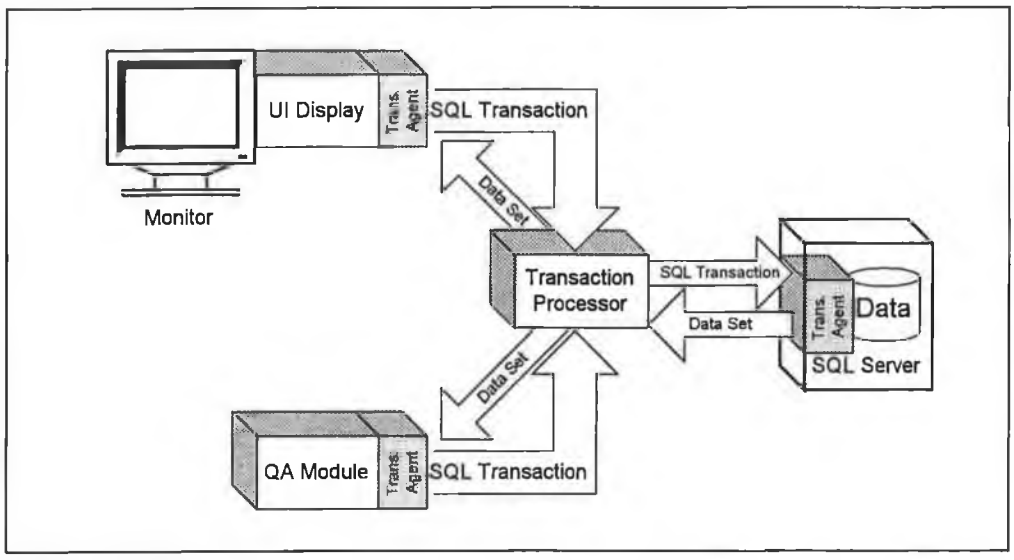


Figure 6.9 Transaction flow for data requests

If the database is remote, the transaction uses a different transaction qualifier which includes the IP address of the remote computer in the qualifier fields. The SQL statement would be identical to a SQL statement which would be used to interact with a local database. This simplifies the SQL generation to a single transparent format for local and remote database activity.

6.3.6. Interactive Chat

This type of service uses message elements which are character-based to provide a service for the real-time interaction between two operators. It is specifically suited to an analyser that is located remotely from the medical laboratory, for example in an Intensive Care Unit (ICU). Therefore, the operators of such an analyser would be healthcare professionals whose core competencies would not primarily be focused on analyser techniques and operations. Personnel with those skills would be based in the laboratory. In order for the efficient

operation of the analyser, sometimes it would be desirable for the experience of laboratory personnel to be available to the remote operators of the analyser.

Interactive communication (CHAT_SEND)	Character-based messages will be transferred to a remote computer

This is achieved through the use of a simple interface (Figure 6.10) which allows a user to type a message and it appears on the other user's screen. This facility could be used by a remote operator to discuss difficulties encountered through use of the analyser with a member of the laboratory staff. As stated in Section 4.8, it is not anticipated that this service would replace traditional communication methods, instead it is merely an additional mode of communication to complement traditional methods.

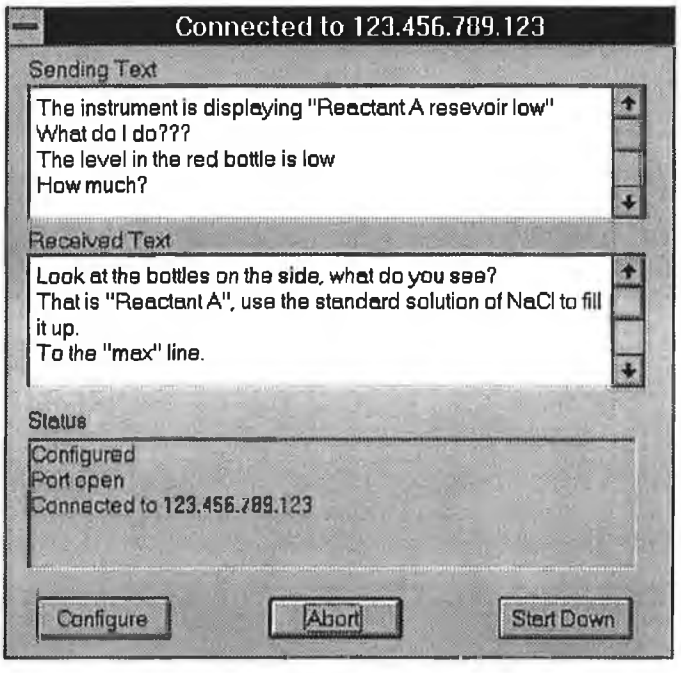


Figure 6.10 Interactive chat user interface screen

In order to deliver varying degrees of real-time interaction, the length of the data being transmitted in every transaction can be tuned. While the data transmission capacity of the network and the volume of network traffic will be the

main data transfer factors, the size of the data packets and their corresponding frequency will also impact on performance. For as close to real-time an experience as possible, every key-press would be transmitted for each transaction command (CHAT_SEND).

In order to preserve network bandwidth, complete words could be transmitted with white space characters indicating the boundary of each word and triggering the transmission of the preceding word. For the most efficient use of network bandwidth, the characters typed could be buffered until a special key is pressed and then the complete message would be sent. This latter type of exchange could be used for distinct question-and-answer sessions where interactivity could be sacrificed for the benefit of network traffic and complete sentences or even paragraphs could be sent with a single transaction.

6.3.7. Tutorial Instruction

This service procedure could be used to augment whatever assistance can be given remotely by an experienced laboratory staff member using the interactive chat service. The tutorial could take the form of a simple text file, a formatted help file, or a multimedia presentation composed of (conceivably) video and audio elements, and could for example describe in detail the procedure for carrying out a calibration test on the analyser, or performing some maintenance task.

A library of previously prepared tutorial material would be stored in a central archive repository. A service could generate a transaction to request a list of available tutorial material and then a different transaction would be generated to request specific material to be transmitted to the remote operator and assist in whatever task was currently being undertaken.

Tutorial instruction (TUTOR_REQUEST)	Tutorial request that delivers a specific tutorial or a list of available tutorials depending on the qualifier

All tutorial requests use the same command (TUTOR_REQUEST) with different qualifiers to indicate what is being requested – a list or a specific tutorial. If a specific tutorial is requested, its name is given in the message field. Service provider responses also use a common command (TUTOR_SEND) with the qualifier field indicating whether it is a list or an actual tutorial. If it is an actual tutorial, the qualifier field specifies the format used for the tutorial (text, audio, etc.). The message field contains the data. A list of tutorials will be a delimited sequence of text listing tutorial entries. An individual tutorial will be a binary file and could be several megabytes in size (fragmentation of such a large data packet is taken care of by the data communication layers).

The whole area of multimedia and interactive help facilities is a vast area of human-computer interaction and is receiving considerable research interest. Whilst the facility is provided in the design concept, it is outside the scope of this analysis and the specific implementation issues are not examined.

7. Database Interface

As stated in Section 4.2, the user requirements revolve around data processing based on storing and retrieving data. This data processing closely couples the design concept with databases and interacting with them. Whilst there are numerous different DataBase Management Systems (DBMS) available, all with their own specific interface requirements, the vast majority support Structured Query Language (SQL). Depending on their degree of compliance with the various generations of the SQL standard (see later Section 7.1), each DBMS provides a SQL interpreter for interacting with the structure of the data and the data itself. Leveraging this widely supported database language simplifies the design considerations for interfacing with databases. Due to the diverse selection of DBMS's in use in the distributed medical environment, SQL provides the necessary flexibility to interact with these existing DBMS's.

In the implemented design concept there are two types of SQL statements utilised by service providers to handle data processing requirements. The service providers store data and retrieve data. The syntax of each type being utilised is supported in the "entry level SQL" conformance definition [ORFA – 96] in order to be successfully executed on the majority of the mainstream DBMS's.

More complex SQL constructs such as JOIN, REVOKE, or bit strings and translations are not explicitly implemented, however that does not exclude their use by future service providers. This is because the transactions service providers generate are SQL statements and can therefore be executed on the majority of the mainstream DBMS's (so long as they are syntactically correct). Therefore, if a data processing module was added which required "intermediate level" or "full SQL" functionalities, it would be its responsibility to generate the appropriate SQL statement, embed it in a transaction, and send it to the Transaction Processor. As with all other transactions, the Transaction Processor would forward it to the service provider – in this case the database connectivity module. Depending on the specific vendor solution being manipulated at that time, the transaction would

either pass or fail. Since conformance is not guaranteed at those levels, success is dependent on a transaction-by-transaction basis. Of course, as more and more vendor solutions embrace the most modern SQL standard, the probability of success for these more complex SQL statements will consequently increase.

7.1. Introduction to SQL

SQL started out as a convenient tool in the development of relational databases. After several revisions, an official standard was adopted by the American National Standards Institute (ANSI) as ANSI standard X3.135 in 1986, and by the International Standards Organisation (ISO) as an ISO standard in 1987. This standard, slightly revised and expanded in 1989, is usually called the "SQL-89" or "SQL1" standard. The standard was ratified in 1992 to ISO SQL-92 or "SQL2" to suggest a staged approach to conformance - entry, intermediate, and full [ORFA – 96]. The most recent version ("SQL3") was drafted in 1997 but will take some time to be implemented by DataBase Management System (DBMS) vendors.

The key generic features of SQL include:

- high-level command structure making it easy to learn and quick to implement
- it is a complete database language which supports database administration, database creation, data manipulation, data sharing, and security concerns
- dynamic data definition allowing database structures to be changed and expanded dynamically
- Client/Server architecture for implementing applications in a distributed environment
- hardware and vendor independence due to the official standardisation of SQL and the adoption by the leading DBMS vendors

Relational databases are composed of self-contained elements called "tables", which hold the actual data. Each entry in a table is referred to as a "row" and is composed of any number of units of data which are called "attributes". The

information contained within a database can be retrieved from any number of tables and filtered to provide only the pertinent data by executing "queries" which specify rules for data to satisfy.

7.2. Transferring SQL Statements

Being able to communicate desired instructions to a DBMS is not the only consideration when interacting with a database. The underlying mechanism for connecting and issuing these instructions must also be addressed.

Every DBMS has its own API for exposing its functionality to other applications (for example Oracle Objects for OLE, OO4O), but an ever-increasing number are supporting the Open Database Connectivity (ODBC). While this has not yet been ratified as a formal standard, it has been adopted to such a degree that it is rapidly becoming the de-facto standard for database connectivity. In this manner, ODBC acts as an insulating layer whereby the application is isolated from the proprietary interface requirements of different DBMS's (Figure 7.1).

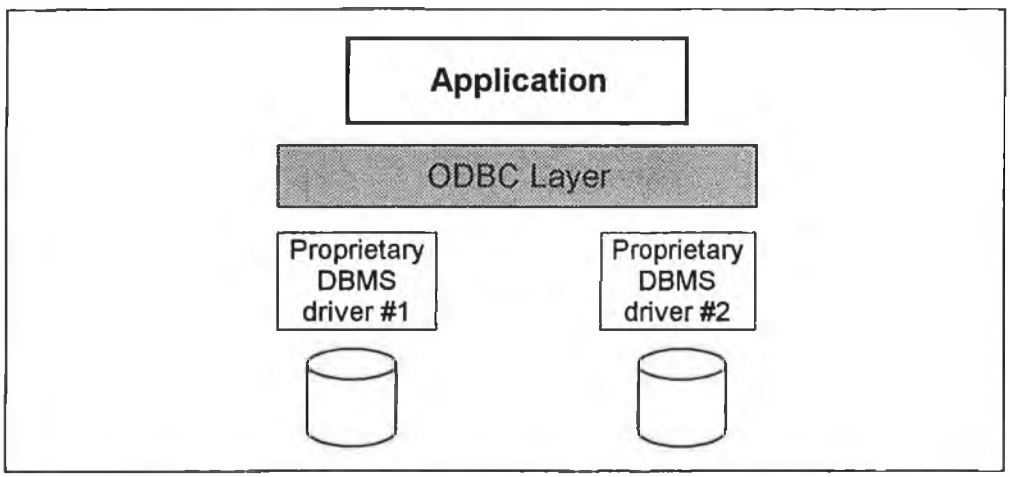


Figure 7.1 ODBC interfaces to various Database Management Systems

It is proposed to utilise ODBC as the database communication mechanism for the design implementation. This necessitates that the database exposes its ODBC interface to the hosting computer system by the specification of a Data Source

Name (DSN) to uniquely identify it. All DBMS's that support ODBC must have specific driver software installed on the computer system. This is typically the case when the DBMS is originally installed; otherwise, the driver software must be installed to permit ODBC with the DBMS.

The transaction link is initiated with an opening transaction where the header is composed of a command component indicating an impending data storage service request (OPEN_DB_LOCAL or OPEN_DB_REMOTE). The message component indicates the database to be affected. Additional database information is specified in an initialisation file which is configurable by an operator with the necessary privileges. This file assists the database interface module in identifying the correct database and table.

The database interface module attempts to establish a connection with the DBMS using ODBC (see later Section 7.3). If this is successful, the requesting module issues a transaction containing a SQL statement (in the message component) in order to achieve a specific database operation: to store the data to the database, the command code DB_STORE is used, whereas to retrieve data from the database the command code DB_EXTRACT is used. Each of these transactions has a message component that contains the appropriate SQL statement to fulfil the required operation.

The Transaction Agent of the database interface module extracts the SQL statement and passes it through the established (ODBC) connection to the DBMS (Figure 7.2). The DBMS's SQL interpreter then attempts to process the SQL statement. Depending on the success of the SQL statement, the service provider returns a pass or fail acknowledgement to the requesting module. A success acknowledgement when retrieving data will contain a dataset fulfilling the specified conditions. A failure acknowledgement will be passed up to the requesting module and it is its responsibility to handle such an occurrence.

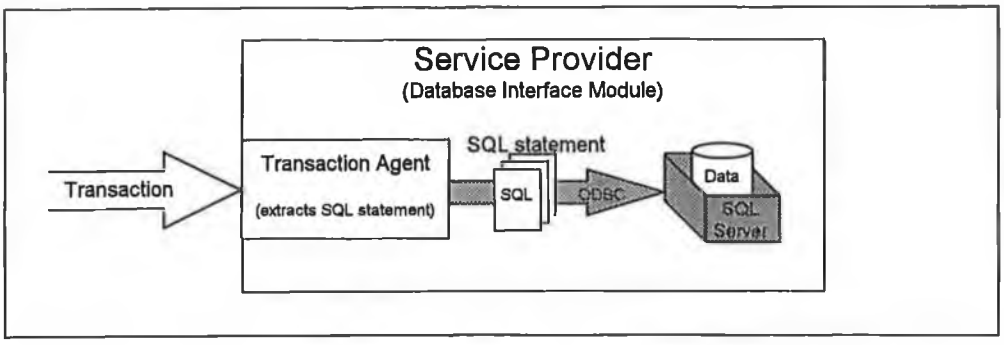


Figure 7.2 Processing a transaction by a database interface module

A failure will cause a default action to occur which attempts to store the data on the "hosts" hard disk using the previously generated SQL statement. Ignoring the possibility of a failure in the file system of the local host, this default action will prevent any data from being irretrievably lost. In an incorrectly configured system where databases, tables, or fields are not accurately and properly declared, failures will happen very frequently. Therefore, the default action could result in a substantial amount of data being stored in the local hard disk if the situation is not rectified quickly. The logging features of the Transaction Processor will highlight which transactions are failing and will assist in correcting the problem(s).

7.3. Database Software Interface

7.3.1. Visual Basic Data Control

For rapid assessment of the feasibility of design choices being made, it was decided to utilise the Data Control available in Visual Basic 3.0. This control is available with the Visual Basic application development environment and simplifies database operations. It uses the Microsoft Access database engine for its local data access functionality, and Open Database Connectivity (ODBC) for remote data access functionality [MICR – 95] which is supported by the vast majority of DBMS's. Thus, the Data Control provides a simplified method of connecting to the vast majority of database management systems commercially available.

When the application is distributed, in order to use the Data Control, the files `MSAJT110.DLL`, `MSAES110.DLL` and `ODBC.DLL` must be copied to the Windows \SYSTEM subdirectory of the target system.

The Data Control has several properties and functions for interacting with it. The "`DatabaseName`" property specifies the exact database, whilst the "`Connect`" property specifies the connection information for different types of databases (see Table 7.1). The "`RecordSource`" property specifies the source of the records accessible through the Data Control. It can directly reference a table, a SQL statement or a "QueryDef". The use of these properties is demonstrated in the sample code in Figure 7.3.

Database Format	DatabaseName	Connect
Microsoft Acecss	drive:\path\filename	(none)
dBASE III	drive:\path	"dBASE III"
dBASE IV	drive:\path	"dBASE IV"
Paradox	drive:\path	"Paradox"
Btrieve	drive:\path	"Btrieve"
FoxPro 2.0	drive:\path	"FoxPro 2.0"
FoxPro 2.5	drive:\path	"FoxPro 2.0"
ODBC	data source name or an empty string ("")	"ODBC; DSN = server; DATABASE = defaultdatabase; UID = user; PWD = password;"

Table 7.1 Properties for a VB Data Control to support various database types

```
ctlData.DatabaseName = "PatientData.mdb"
ctlData.Connect = ""      ' using MS Access, so leave blank
ctlData.RecordSource = "SELECT * FROM pHTable WHERE PtID = "
& iPatientNum
ctlData.Refresh
If (ctlData.Recordset.RecordCount > 0) Then
    ' successful - try extracting the values
    ipHValue = ctlData.Recordset.Fields("pH").Value
    iHbValue = ctlData.Recordset.Fields("Hb").Value
End If
```

Figure 7.3 VB code sample for interacting with a Data Control

The recordset resulting from correctly specifying these properties can be manipulated using a rich API composed of field objects and navigation operations such as "MoveFirst", "MoveNext", "MoveLast", and "EOF". These are typically used for processing any data contained within the recordset (Figure 7.4).

```
i = 0
ctlData.Recordset.MoveFirst
Do Until ctlData.Recordset.EOF
    DataArray(i, 1) = ctlData.Recordset.Fields("pH").Name
    DataArray(i, 2) = ctlData.Recordset.Fields("pH").Value
    i = i + 1
    ctlData.Recordset.MoveNext
Loop
```

Figure 7.4 VB code sample for processing recordsets with a Data Control

7.3.2. Simplified Database API

From the functional prototyping achieved using the Visual Basic control, it was obvious that a relatively simple Application Programming Interface (API) was sufficient. Thus, for performance reasons and due to the additional files that must be distributed with the design solution, it was decided to directly leverage the system-level calls for database communication. This is achieved using a VB code module that invokes certain Windows API functions while providing a simplified API to the design concept implementation. This also permitted the code module to follow a similar interface specification as the other code modules in the design concept (serial communication – Section 5.6.2).

As with the other code modules in the adopted modularised approach for the design concept, it does not require any special installation process, so the DLL's required by the Data Control in the preceding section are not necessary. However, it does rely on one file – ODBC.DLL – since it is specifically for connecting to ODBC-compatible DBMS's.

This simplified API requires the same basic categories of functionality as that provided by the C-based serial communications DLL. These five categories –

`Configure`, `Open`, `Close`, `Read`, and `Write` – are implemented as five database-centric interfaces: `ConfigODBC`, `OpenODBC`, `CloseODBC`, `ReadODBC`, and `WriteODBC`. Since they are VB functions in a VB code module, it is not possible to use the "`Alias`" keyword in the code declarations to assign a non-conflicting name as in the case when DLL's are used. Being in a code module, the functions are essentially internal to the design concept compared to an external DLL. They would therefore conflict with Visual Basic reserved words and any global variable or constant, or other procedure in the same scope with the same name. The Windows API functions which are invoked by these simplified API functions are shown in Figure 7.5.

```
Declare Function SQLAllocEnv Lib "odbc.dll" (HenvPtr As Long)
As Integer
Declare Function SQLAllocConnect Lib "odbc.dll" (ByVal HENV
As Long, HDBCPtr As Long) As Integer
Declare Function SQLConnect Lib "odbc.dll" (ByVal HDBC As
Long, ByVal DSN As String, ByVal DSNLen As Integer, ByVal UID
As String, ByVal UIDLen As Integer, ByVal AuthStr As String,
ByVal AuthStrLen As Integer) As Integer
Declare Function SQLDisconnect Lib "odbc.dll" (ByVal HDBC As
Long) As Integer
Declare Function SQLFreeConnect Lib "odbc.dll" (ByVal HDBC As
Long) As Integer
Declare Function SQLFreeEnv Lib "odbc.dll" (ByVal HENV As
Long) As Integer
Declare Function SQLAllocStmt Lib "odbc.dll" (ByVal HDBC As
Long, HSTMTPtr As Long) As Integer
Declare Function SQLFreeStmt Lib "odbc.dll" (ByVal HSTMT As
Long, ByVal fOption As Integer) As Integer
Declare Function SQLExecDirect Lib "odbc.dll" (ByVal HSTMT As
Long, ByVal SQLStr As String, ByVal SqlStrLen As Long) As
Integer
Declare Function SQLBindCol Lib "odbc.dll" (ByVal HSTMT As
Long, ByVal ColNum As Integer, ByVal ColType As Integer,
ByVal ColData As String, ByVal MaxData As Long, ActualData As
Long) As Integer
Declare Function SQLFetch Lib "odbc.dll" (ByVal HSTMT As
Long) As Integer
Declare Function SQLPrepare Lib "odbc.dll" (ByVal HSTMT As
Long, ByVal SQLStr As String, ByVal SqlStrLen As Long) As
Integer
Declare Function SQLExecute Lib "odbc.dll" (ByVal HSTMT As
Long) As Integer
Declare Function SQLError Lib "odbc.dll" (ByVal HENV As Long,
ByVal HDBC As Long, ByVal HSTMT As Long, ByVal SQLState As
String, NativeErrorCode As Long, ByVal ErrMsg As String,
ByVal ErrorMsgMax As Integer, ErrorMsgLen As Integer) As
Integer
```

Figure 7.5 VB declarations for invoked Windows API functions

For demonstration purposes, the `OpenODBC` function is listed in Figure 7.6.

```
Function OpenODBC (DSN As String, UID As String, PassWd As
String) As Integer
Dim RetCode As Integer
    DEBUGSQL = True
    If DataBaseOpen = 0 Then
        RetCode = SQLAllocEnv(HENV)
        If (RetCode = SQL_SUCCESS) Then
            RetCode = SQLAllocConnect(HENV, HDBC)
            If (RetCode = SQL_SUCCESS) Then
                RetCode = SQLConnect(HDBC, DSN, SQL_NTS,
                UID, SQL_NTS, PassWd, SQL_NTS)
                If (RetCode = SQL_SUCCESS) Or (RetCode =
                SQL_SUCCESS_WITH_INFO) Then
                    DataBaseOpen = 1
                    OpenODBC = 0
                Else
                    RetCode = SQLFreeConnect(HDBC)
                    RetCode = SQLFreeEnv(HENV)
                    OpenODBC = -1
                End If
            Else
                RetCode = SQLFreeEnv(HENV)
                OpenODBC = -1
            End If
        Else
            OpenODBC = -1
        End If
    End If
End Function
```

Figure 7.6 VB code for opening an ODBC database

A simple Visual Basic code sample in Figure 7.7 shows how the interfaces can be invoked and used.

```
' try opening the database
If (OpenODBC("TestDB", "Admin", "") = False) Then
   MsgBox "Error opening the database", 48
Else
   ' try outputting a command string to the database
   strSQL = "INSERT INTO TestTable (FName, LName) VALUES
   ('John', 'Smith')"
   bRetVal = WriteODBC(strSQL)

   ' done, so close the database
   If (CloseODBC() = False) Then
      MsgBox "Error closing the database", 48
   End If
End If
```

Figure 7.7 VB code sample for invoking simplified database functions

8. Remote Connectivity

In order to support the remote connectivity specified in the user requirements, a data communication channel is necessary. As with other areas of the design concept, this functionality will rely on existing standards which are widely implemented and in a mature state of development. This approach encourages the adoption of an open data communication environment to seamlessly integrate with other components of the design solution. The modularised approach introduced in the design chapter (Chapter 4) will facilitate the independent implementation of the data communication functionality. As competing standards are adopted in the medical domain, the data communication module can be upgraded or replaced with the appropriate replacement module so long as it conforms to the simple interface specification detailed in this chapter.

8.1. Introduction to Network Protocols

Data communication functionality is provided by software modules called "stacks" which are composed of layers of functionality. Each layer operates on the data as it passes through it from the application to the electrical components which provide the physical connection between the two communicating computers. A number of layers can work together to implement a certain protocol. Figure 8.1 shows a layered description of a network stack with various protocols inserted into their appropriate position in the layered structure. (Some of the elements will be described, but the rest are displayed for demonstration purposes.)

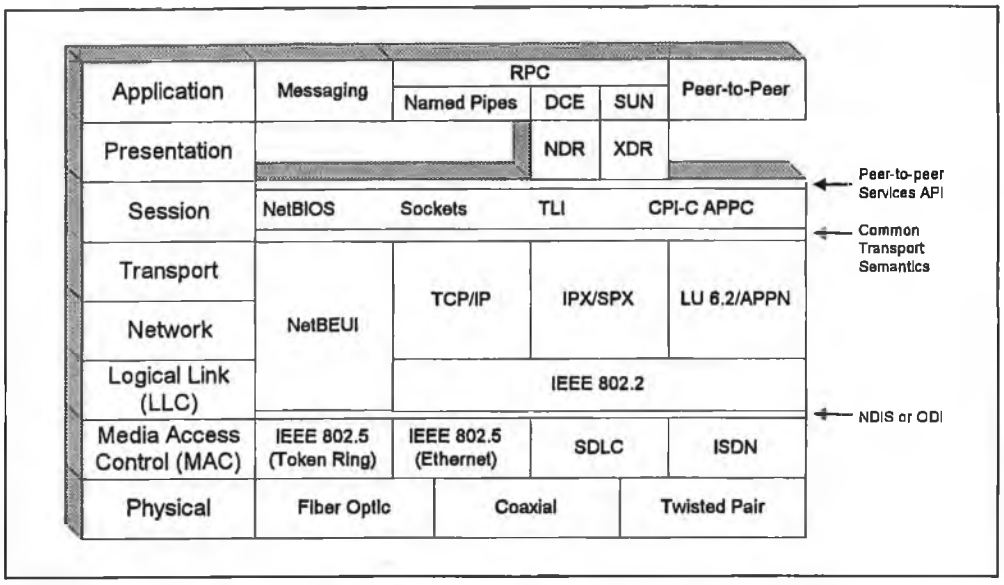


Figure 8.1 Various protocols within a stack architecture

8.1.1. Open Systems Interconnection Reference Model

The definitive layered approach to data communication was defined by International Standards Organisation (ISO) between 1977 and 1984 and is called the Open Systems Interconnect (OSI) Reference Model (ISO reference number 7498). According to Andrew Tanenbaum [TANE – 96] the layered approach adopted by this large world-wide standards group had the following ideals in mind:

- A layer should be created where a different level of abstraction is needed.
- Each layer should perform a well-defined function.
- The function of each layer should be chosen with the objective of defining internationally standardised protocols.
- The layer boundaries should be chosen to minimise the information flow across the interfaces.
- The number of layers should be large enough that distinct functions do not have to be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

The OSI model is composed of seven layers (Figure 8.2) each with well-defined functionalities, the implementation details of which are hidden from all other layers. This simplifies transferring the network stack to a new hardware platform or operating system since only the layers that are different need to be redesigned to address the target-specific issues.

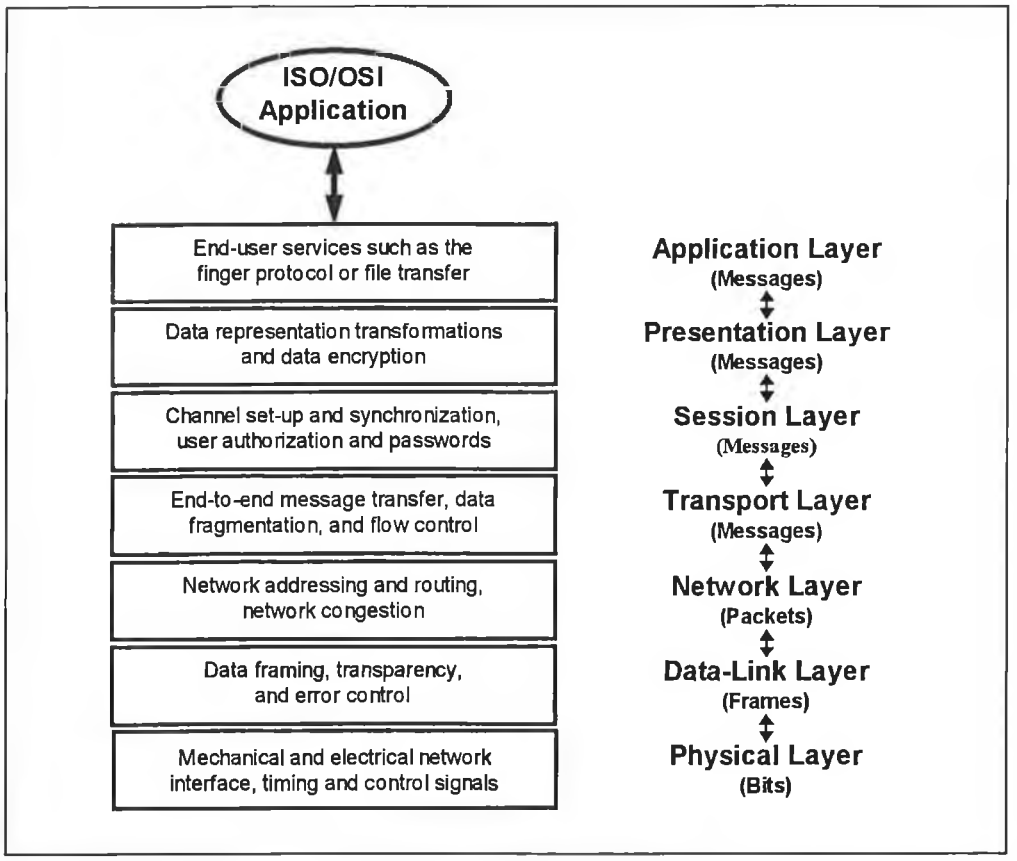


Figure 8.2 ISO/OSI seven layer network model

While OSI has provided a conceptual framework for network application developers, its presence in terms of products designed in accordance with the conceptual model has, until recently, been minimal in the commercial marketplace [BURN – 93]. Other communication protocols based on the OSI framework have achieved greater dominance, one of which is described next.

8.1.2. Internet Protocol Network Stack

The most widely implemented network stack is commonly referred to as the Internet stack, or more accurately the Transport Control Protocol/Internet Protocol (TCP/IP) stack. This layered model is similar to the OSI model, however it only consists of four discrete layers. Some layers encapsulate functionalities which are separated into component layers in the OSI Model (Figure 8.3).

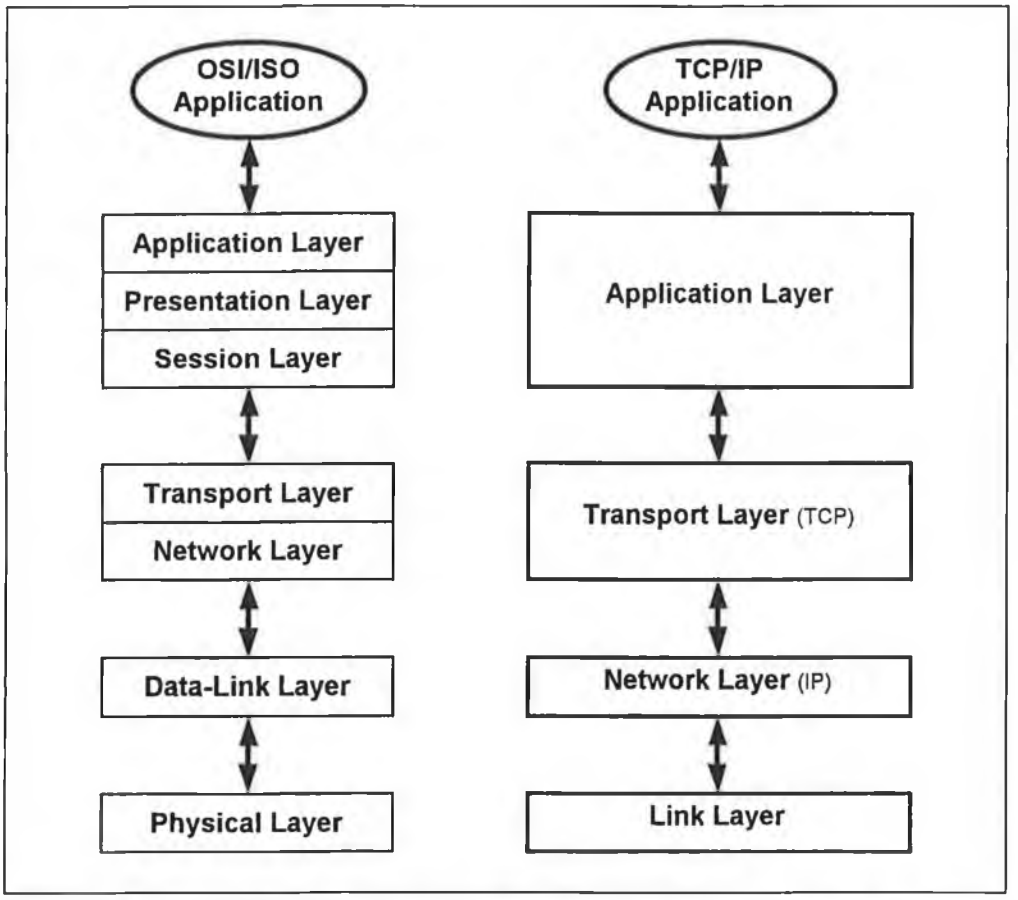


Figure 8.3 TCP/IP four layer network model compared to the OSI model

The initial development of the TCP/IP network protocol suite was undertaken as a research project funded by the Defence Advanced Research Projects Agency (DARPA) in 1969 and resulted in the experimental network called ARPANET. It initially serviced non-classified military communications, but its popularity grew when it was expanded to include educational establishments to further the research effort.

The protocol rapidly developed into a mature specification through a unique architecture methodology. Each design proposal, review and adoption was publicly available. Through the use of Request For Comment (RFC) documents it encouraged debate and input from individuals and small groups, as well as established research committees. This informal method of establishing new protocol standards was in stark contrast with the stringent requirements employed by the ISO. This RFC process was far more expeditious than the equivalent procedure utilised for defining the OSI model and consequently resulted in its quick adoption as the de facto standard for data communication across computer networks. Each RFC document addresses different aspects of the TCP/IP network suite and is publicly available. As a result anybody can make recommendations on the content of an RFC and the resulting RFC is the product of global consensus.

Some of the distinguishing features of the TCP/IP network suite are:

- Independence of network topology
- Independence of network hardware
- Independence of operating system
- Publicly available protocol standard
- Universal addressing scheme
- Powerful Client/Server framework
- Application-specific protocols

The widespread support and implementation of this network suite provides the main reason for its advocacy as the data communication protocol for supporting the remote connectivity aspect of the design concept. Numerous hardware platforms and operating systems have their own TCP/IP network stack. In the diverse environment of the medical domain, homogeneity of computer systems is not a realistic possibility, so the ready availability of a network stack for all platforms is necessary in order to integrate these dissimilar computer systems.

8.2. TCP/IP Components

The TCP/IP network protocol uses an abstract end-point for all communication services called a socket. This is very similar to the UNIX-based object descriptor which provides the UNIX operating system with a handle for interacting with various elements of the system such as printers, tape drives, and disk files. The descriptor enables input/output operations using an "open-read-write-close" process for all system interactions. However network communication requires additional functionality to the file descriptor process to support an extra step: "open-define-read-write-close". The "define" segment of the process permits the specification of a remote computer and unique process with which to participate in data communications.

Data communications based on sockets inherently use the Client/Server model: a socket on a server is configured to wait for client sockets to initiate a data link with it; then either party can read or write data on the established data link.

8.2.1. Windows Sockets API

The Windows Sockets Application Programming Interface (WinSock API) is a software interface to the TCP/IP network suite on computers which host the Microsoft Windows operating system. It is vendor-independent for any TCP/IP-compliant network suite. As explained in section Chapter 2, the predominant hardware platform available in the medical domain is the Personal Computer (PC) with Microsoft Windows as its operating system. Therefore the WinSock API is the network suite employed in the design concept.

The TCP/IP network model was initially designed for implementation on the UNIX operating system. As such it utilised the multi-tasking features of that operating system and pre-emptively multi-tasks between processes. If a function does not return immediately, the UNIX operating system pre-empts that thread of execution and allows other threads of operation to proceed. So no single function call can bring the system to a grinding halt while it waits for a response from a remote computer which may or may not even be operational.

However, under the Windows operating system, if a function does not return immediately the co-operative multi-tasking nature of Windows cannot interrupt the stalled function and it blocks further execution of other program statements and processes.

There are two ways of getting round this blocking function call. The first is to use non-blocking sockets. When a socket is created it is blocking by default, but it can be converted to non-blocking mode by using the `ioctlsocket()` function (this function is a WinSock-specific extension of the `ioctl()` function for input/output control on a file descriptor under UNIX). If a function is subsequently called which would not return immediately, an error code is returned which indicates that the function cannot immediately complete. The `select()` function is used in conjunction with the `ioctlsocket()` function to query the status of the non-blocking socket. It checks the readability, writeability, and exception status of one or more sockets. Non-blocking sockets are frequently used in Windows-based applications to prevent the calling application from locking up the system due to a blocking function call.

The second solution is to take advantage of the message-driven nature of Windows and use asynchronous function calls. Asynchronous functions include parameters which the operating system uses to send a message to report the success or failure of the asynchronous operation. When an asynchronous function is called, it tells the operating system to notify the caller when the operation has been completed and immediately continues execution to the next program statement. Thus, the calling function does not cause execution to be suspended until the operation is completed but instead permits the application to continue. When the operation has completed the operating system posts a notification message to the window of the caller which reports the success or failure of the asynchronous operation. Thus the caller must have a message-handling routine to process the outcome of the asynchronous operation. All WinSock asynchronous functions are identified by the `WSAAsync` prefix. The `WSAAsyncSelect()` function

changes a socket into non-blocking mode and informs the operating system of the combination of network operations which are to be monitored.

The difference between the two solutions is somewhat subtle. The non-blocking method returns an error immediately if the network operation cannot be completed. Whereas the asynchronous function method relinquishes control and does not return an error immediately if the network operation cannot be completed, instead it eventually times out and then returns an error.

8.2.2. Types of Data Transfers in TCP/IP

The TCP/IP network suite supports three data communication services depending on which protocol is used over the socket data link:

- User Datagram Protocol (UDP)
- Transport Control Protocol (TCP)
- Internet Control Message Protocol (ICMP)

UDP provides a connectionless datagram service where data packets are sent to an Internet address on a best-try basis. No point-to-point virtual communication link is established, therefore data packets must contain the target IP address of their destination. Thus, the sender and receiver must know each other's IP address. This mechanism results in a protocol with minimal overhead since channel management and transfer acknowledgements are not required. It is typically used in applications where data integrity is not as important as data transmission rates and processing overhead.

It is proposed to use this type of data transfer for the chat and tutorial aspects of the design concept. Having a high throughput is more important than accuracy of data. If a packet fails to arrive (for whatever reason), the overall meaning of the complete exchange (i.e. during use of the "chat" functionality) may still be intelligible. If not, then the user can request just that lost portion.

TCP provides a reliable, connection-oriented stream service where a communication channel is established and maintained throughout the complete data exchange. Once this channel has been established, data packets simply need to be sent to the channel where the underlying protocol ensures their delivery over the virtual link. Packet fragmentation, reassembly and framing are handled by the protocol. If a packet fails to arrive (for whatever reason), the protocol manages the request for retransmission invisibly from the upper layers in the communication stack. These channel management features incur the penalty of higher overheads. This mechanism of data transfer is typically used in applications where data integrity is more important than data transmission rates and processing overhead.

It is proposed to use this type of data transfer for what is anticipated to be the majority of remote communication – transactions with messages composed of SQL statements. Since it is critical that these transactions are delivered in their entirety to their destination, the reliable connection-oriented nature of TCP makes it the preferred choice over UDP.

ICMP uses low-level protocols to bypass the transport layer and communicate directly with the network layer. It is typically used for flow control, error reporting and routing manipulation. Thus, its use is more suited to network management instead of data transfer and consequently it is not used in the design concept.

8.3. Progressive Application Developments

A number of different applications were developed to investigate the design considerations associated with the WinSock API. These ranged from very basic utilities to complete networked applications and are described briefly below.

8.3.1. Test Application 1: Host Name and Service Resolution

This first WinSock application – `wsdbtest.exe` – was developed in the Microsoft Visual C++ environment. It uses the Microsoft Foundation Classes (MFC's) to draw the window frame and handle the menu messages which are used

to control the WinSock functions. It uses database look-up functions to convert between different formats for Internet addresses and Internet Services and displays the results in pop-up message boxes.

Internet addresses can be represented as dotted decimal address or dotted ASCII text name (for example "123.456.789.123" or "www.nasa.gov"). The function `gethostbyaddr()` returns the dotted ASCII text name when supplied with the dotted decimal address. The function `gethostbyname()` returns the dotted decimal address when supplied with dotted ASCII text name. The asynchronous equivalents of these functions – `WSAAsyncGetHostByAddr()` and `WSAAsyncGetHostByName()` – were also used.

Protocol ports uniquely identify a service being provided being provided by a Server such as finger or ftp. Protocol ports can be represented by their number or by a service name. The function `getservbyport()` returns the service name when supplied with the port number and `getservbyname()` returns the port number when supplied with the service name. The asynchronous equivalents of these functions – `WSAAsynGetServByPort()` and `WSAAsyncGetServByName()` were also used.

8.3.2. Test Application 2: User Datagram Protocol Client and Server

These two applications – `udpcli.exe` and `udpserv.exe` – were developed in the Microsoft Visual C++ environment. They use the Microsoft Foundation Classes (MFC's) to draw the window frame and handle the menu messages which are used to control the WinSock functions. They implement the connectionless datagram communication protocol using blocking function calls.

As described previously, both applications create a socket, however in this implementation only the Server application binds its socket to a port. The reason for this is that it was decided that communication need only be uni-directional – the Client sends datagrams to the Server and the Server receives them, but no data

flows in the opposite direction. Thus the Client sends datagrams to the Server and the Server simply receives the datagrams. Both applications display messages in the client area of its window indicating the status of the network operations and the Server also displays any messages it receives.

8.3.3. Test Application 3: Asynchronous Stream-Connected Client and Server

These two applications – `asyncli.exe` and `asynserv.exe` – were developed in the Microsoft Visual C++ environment. They use the Microsoft Foundation Classes (MFC's) to draw the window frame and handle the menu and button messages which are used to control the WinSock functions. They implement the connection-oriented communication protocol using asynchronous function calls.

As described previously, both applications create a socket but only the Server application binds its socket to a port and then listens for connections from Clients. When the Client attempts to connect to the Server, the link is established when the Server acknowledges the Client. All data transfers take place over this virtual link bi-directionally until the link is terminated. Both applications display simply messages in the client area of its window indicating the status of the network operations. Each application has an edit box control for typing messages which get transmitted one line at a time on a menu click event. Thus communication is bi-directional.

8.3.4. Test Application 4: Simplified API Implemented as a DLL

In order to speed up application development and provide a simplified programming interface to the WinSock functions, a Dynamic Link Library (DLL) was developed. This was the ultimate aim of the investigation into the WinSock API and resulted in access to TCP/IP data communication through a streamlined interface where it encapsulated network functionality within five function calls. Just as with the serial data communication module (Section 5.6.2), this kind of modular

approach to application development allows the development of network communication to be carried out independently from the rest of the application.

This C-based DLL – `aiicsock.dll` – provides all the features necessary to establish a connection-oriented link, transmit and receive data, and terminate the link. The five interfaces are: `configure()`, `open()`, `close()`, `read()`, and `write()` (Figure 8.4). A data structure is also defined for use in configuring and identifying the socket for communication.

```
extern "C" BOOL FAR PASCAL _export Configure(HWND hCaller,
struct ConfigStruct far *Config);

extern "C" BOOL FAR PASCAL _export Open(HWND hCaller, struct
ConfigStruct far *Config);

extern "C" BOOL FAR PASCAL _export Close(HWND hCaller,
struct ConfigStruct far *Config);

extern "C" WORD FAR PASCAL _export Write(HWND hCaller,
struct ConfigStruct far *Config, void far *Data, unsigned
iCount);

extern "C" WORD FAR PASCAL _export Read(HWND hCaller, struct
ConfigStruct far *Config, void far *Data, unsigned iCount);

/* Structure of IP address and port data */
typedef struct tagSockStruct
{
    char cRemoteIP[16];
    unsigned iRemoteListenPort;
    unsigned iRemoteTalkPort;
    unsigned iLocalListenPort;
    unsigned iLocalTalkPort;
} SockStruct;

/* Structure of configuration data */
struct ConfigStruct
{
    char    cDeviceID[16];      /* used to identify the type/name of device */
    DWORD   dwConfigDataLen;    /* overall length of this structure */
    SockStruct Configinfo;      /* customized structure for IP and ports */
    BOOL    bOpen;              /* open or closed */
    WORD    hConnection;        /* open or closed */
    CWinSock far* m_pWinSock;/* WinSock sub-system startup/shutdown */
};
```

Figure 8.4 C function definitions for a TCP/IP communications DLL

In order for these interfaces to be used by a Visual Basic application, they must be declared in a manner that Visual Basic can recognise (*Figure 8.5*). "`Alias`" is used to identify the name of the procedure in the DLL for use in the Visual Basic domain. This prevents the external procedure name conflicting with a Visual Basic reserved word, a global variable or constant, or any other procedure in the same scope. Thus the five C-based DLL interfaces are renamed to: `ConfigSocket`, `OpenSocket`, `CloseSocket`, `ReadSocket`, and `WriteSocket`. A VB data structure is also defined for use in configuring and identifying the socket.

```
Declare Function ConfigSocket Lib "AIICSock.dll" Alias
"Configure" (ByVal HWND As Integer, pConfig As
TCPIPConfigStruct) As Integer

Declare Function OpenSocket Lib " AIICSock.dll" Alias "Open"
(ByVal HWND As Integer, pConfig As TCPIPConfigStruct) As
Integer

Declare Function CloseSocket Lib " AIICSock.dll" Alias
"Close" (ByVal HWND As Integer, pConfig As TCPIPConfigStruct)
As Integer

Declare Function ReadSocket Lib " AIICSock.dll" Alias "Read"
(ByVal HWND As Integer, pConfig As TCPIPConfigStruct, ByVal
strBuffer As String, ByVal iBufSize As Integer) As Integer

Declare Function WriteSocket Lib " AIICSock.dll" Alias
"Write" (ByVal HWND As Integer, pConfig As TCPIPConfigStruct,
ByVal strBuffer As String, ByVal iBufSize As Integer) As
Integer

Type TCPIPConfigStruct
    strDeviceID As String * 16 ' used to identify the type/name of device
    ConfigDataLen As Long      ' overall length of this structure
    strRemoteIP As String * 16 ' IP address of remote system
    iRemoteListen As Integer   ' number of listen port for remote app
    iRemoteTalk As Integer     ' number of talk port for remote app
    iLocalListen As Integer    ' number of listen port for local server app
    iLocalTalk As Integer      ' number of talk port for local server app
    iDeviceNum As Integer      ' used to identify multiple devices
    bOpen As Boolean           ' open or closed
    hConnection As Integer     ' 16-bit connection handle
    pWinSock As Long           ' 32-bit pointer to WinSock object
```

Figure 8.5 VB declarations for calling functions from a DLL API

A simple Visual Basic code sample in Figure 8.6 shows how the interfaces of the C-based DLL can be invoked and used.

```
' use the DLL's configuration dialog box
If (ConfigSocket(HWND, TCPIPConfig) = False) Then
   MsgBox "Error configuring the socket", 48
Else
   ' try opening the socket
   If (OpenSocket(HWND, TCPIPConfig) = False) Then
      MsgBox "Error opening the socket", 48
   Else
      ' try outputting a command string to the socket
      bRetVal = WriteSocket(HWND, TCPIPConfig, strCommand,
Len(strCommand))
      ' done, so close the sockett
      If (CloseSocket(HWND, TCPIPConfig) = False) Then
         MsgBox "Error closing the socket", 48
      End If
   End If
End If
```

Figure 8.6 VB code sample for invoking a TCP/IP communications DLL

With the first reference to the DLL, it creates a socket to listen for remote Clients trying to connect. The `ConfigSocket()` function allows a local protocol port, a remote protocol port, and a remote Internet address to be specified for the communication channel. The `OpenSocket()` function then spawns a second socket for communication with the remote PC specified in the `ConfigSocket()` function. If another remote PC attempts to communicate then a third socket is spawned. Therefore, each application using the DLL has the potential to behave like a Server waiting for connection requests, and also like a Client actively requesting connections from a Server.

8.3.5. Test Application 5: Asynchronous Peer-to-Peer Communication

This application – `sock.exe` – was developed using Microsoft Visual Basic. Applications were developed for versions 3.0 and 4.0 of Visual Basic. It implements an asynchronous connection-oriented communication link by calling functions from the previously described DLL. Therefore, two computers can communicate with each other using this application running on each of them without any concerns about who is the Server and who is the Client. This is

because the DLL creates an extra socket which constantly listens for connection requests.

The application is controlled by button clicks rather than menus. It allows text to be typed into an edit box and then transmitted to the specified remote computer, which then displays the text in another edit box. Text can be transmitted in either direction since the link is symmetrical. This application laid the foundation for the development of the chat utility.

A variation was developed for transferring files. Once the full path of the file had been specified, the file gets broken into chunks of 1024 bytes, transmitted, and reassembled at the receiving computer. Numerous file types (such as text, image, and binary files) were successfully transferred using this technique. It proved the feasibility of using the same mechanism for transferring tutorial material in the form of binary files.

8.3.6. Conclusions from Progressive Application Development

All of the test applications described in this section contributed to the overall implementation of the design concept. They assisted in establishing the necessary technical details for specific areas of functionality.

For exchanging SQL-based transactions to succeed, it must be able to ensure accurate and complete delivery of transactions between modules. Stream-connected clients (Section 8.3.3) deliver reliable data communication that resends any data packets that do not correctly arrive at their destination.

By comparison, the "chat" functionality does not necessitate reliable data communications. In this capacity, it can be considered loss-tolerant. The connectionless datagram communication protocol (Section 8.3.2) attempts a "best try" to the delivery of data packets to specified destination, whilst minimising the processing overhead. This type of data communication is suitable for the peer-to-peer nature of the "chat" utility (Section 8.3.5).

By encapsulating the data communication functionality within a DLL (Section 8.3.4), the design guideline of a modularised architecture is perpetuated. Being a DLL permitted its implementation using C with the corresponding benefits in efficient and rapid execution. Exposing the functionality through a simplified API results in a programming paradigm aligned with the existing modules for serial and database communications. It also makes its adoption into the design implementation less complicated, easier to debug and simpler to black box test.

8.4. Telephony Support

Due to the distributed nature of the healthcare environment, remote connectivity may not be available using traditional physical media such as Ethernet on coaxial cable or twisted pair. This may be due to a remote location not having complete access to the hospital infrastructure (for example old buildings not included in the Local Area Network (LAN)), or geographic remoteness where the site is not on the actual grounds of the hospital (for example a GP performing house calls).

In these situations alternative physical layer media are necessary. Telephony-based networking exploits the indigenous telephone network which permeates all areas of the healthcare environment. Wireless alternatives would include satellite and cellular technologies which would support truly mobile networking. However the scope of wireless telecommunications are too extensive to be included as merely a section in this research. Only telephony data communications were investigated.

Just as the lowest layer in the TCP/IP model (Physical Layer) supports multiple network technologies (Figure 8.1), it can also be modified to handle physical connection to a telephone network. It simply involves replacing the lower network-specific layers with layers which can deal with telephony communication protocols such as Point-to-Point Protocol (PPP) or Serial Line Interface Protocol (SLIP).

When a modem is connected to a computer with the appropriate TCP/IP stack, the lower layers transparently handle the telephony issues of dialling and maintaining the data link. The modem encodes the computer-generated digital bits into analogue signals which can be transmitted over the telephone infrastructure. (A growing number of telephone networks are making digital networks available which use protocols like Integrated Services Digital Network (ISDN) and so would not require the digital-to-analogue conversion stage.)

8.4.1. Telephony Standards

The International Telegraph and Telephone Consultative Committee (CCITT) define various telephony-based data communication protocols which specify transmission speeds and processing of the data being transmitted. The defined V series of standards applies to existing switched telephone networks [HALS – 92] and is outlined in the following table.

CCITT V-Series					
2/4-wire leased circuits		2-wire switched circuits		Point-to-point	
V.23	600 or 1200 bps	V.21	300/300 bps duplex	V.35	48 kbps
V.26	1200 or 2400 bps	V.22	1200/1200 bps duplex	V.36	48-72 kbps
V.27	2400 or 4800 bps	V.22bis	2400/2400 bps duplex	V.37	96-168 kbps
V.29	4800 or 9600 bps	V.23 A	75/1200 bps duplex		
		V.29	4800/9600 half-duplex		
		V.32	4800/9600 duplex		

The protocols specify permissible processing which may be applied to data and are being expanded on a continuous basis. Error control and compression are the main data processing implemented within modems [CHEN – 91]. Error control improves data transfer by providing an error-controlled reliable connection. V.42 is an error control protocol established by the CCITT, and Microcom Networking Protocol (MNP) 4 has been adopted by the CCITT as an alternate error control protocol.

The two protocols use a sophisticated algorithm to make sure that the data received match with the data sent. V.42 (and MNP 4) copes with the telephone line impairments by filtering out the line noise and automatically retransmitting corrupted data.

The other benefit of V.42 (or MNP 4) is that it can improve throughput. Before sending the data to a remote system, a modem with V.42 (or MNP 4) assembles the data into packets and during that process it is able to reduce the size of the data. A character typically takes up 1 start bit, 8 data bits and 1 stop bit for a total of 10 bits. When two modems establish a reliable link using V.42 or MNP 4, the sending modem strips the start and stop bits (thereby subtracting 20% of the data) and sends the data to the other end. The receiving modem then reinserts the start and stop bits and passes the data to the remote computer.

Besides error control protocols, all current high-speed modems also support data compression protocols. That means the sending modem will compress the data on-the-fly and the receiving modem will decompress the data to its original form. There are two standards for data compression protocols, MNP-5 and CCITT V.42bis. Some modems also use proprietary data compression protocols [CHEN – 91].

8.4.2. Controlling Modems

Modems are configured and controlled using control strings. The most common collection of instruction strings is the Hayes AT command set and is the de facto standard for modem control. Once the port connected to the modem has been correctly configured for baud rate, stop bits, etc. (refer to Section 5.5 and Appendix B), commands can be sent to the modem. These commands can set the modem to auto-answer mode ("`ATA`") or dial a telephone number ("`ATD1234567`"). If the dialled telephone is connected to a modem which is in auto-answer mode, then the two modems negotiate transmission parameters prior to establishing the communication link. After this exchange (which can take several seconds), data can be transferred over the link just as if it was a direct network connection.

8.4.3. Application Programming Interface

The simplified API which was developed for TCP/IP communication (Section 8.3.4) was modified to support telephony communication. The main adjustments were to the configuration structure which gets passed in the `OpenSocket()` function call. Instead of containing TCP/IP-specific information (like IP addresses and port numbers), the new configuration structure contains modem-specific information (like baud rate, start & stop bits, and command strings).

As with all other elements of the design concept, the telephony component is modularised through the use of the simple API mentioned above. As different communication alternatives are exposed they can replace the module without impacting on the rest of the design concept. Other Computer-Telephony Integration (CTI) applications utilise the Microsoft Windows Telephony API (TAPI) for increased functionality and flexibility in many environments [MICR – 96]. Whilst this protocol provides many powerful features, the simplistic requirements of this prototype design make TAPI an over-engineered component. Future development could necessitate the adoption of TAPI for telephony data communication. In that case, the modular design would facilitate its implementation.

8.4.4. Telephony Scenarios

There are two basic scenarios where telephony communication is desirable. The first applies to a GP who is in his surgery or visiting a patient. If a sample had previously been taken from the patient and processed, the GP could connect to the server in the laboratory using a modem plugged into a standard telephone line. After an appropriate negotiation where the GP's identity is verified, the GP could request the patient's test results from the server. Or a collection of past results could be requested to provide the supporting data for a historical trend analysis of the patient's progress.

The second scenario is an access resolution issue. If there is telephone network in place (be it a public network or a private hospital-wide network), it may be more appropriate to use it as the communication medium. This would be the case if there was no LAN in place or if the network traffic on the LAN was substantially congested compared to telephone traffic. Thus, the availability of the telephone network could be exploited to overcome LAN-derived limitations.

Due to the considerable time involved in establishing a communication link between two modems (due to dialling, answering, and negotiation exchange) and the possibility of an engaged modem line, time-critical data is not particularly suited to this type of communication medium.

9. Conclusions

9.1. Design Implementation

The rapid prototyping possible with Microsoft Visual Basic made it the logical choice for substantial areas of the implementation of the design concept. Visual Basic proved to be a critical tool in evaluating the feasibility of concepts in the design solution due to the rapid turn-around from design to prototype. The powerful Graphical User Interface (GUI) development environment of Visual Basic made it the sensible choice for implementing all the GUI elements. In order to avail of object oriented methods, Visual Basic version 4.0 was the actual development environment employed. This provided access to class definitions and the "`Public`" and "`Private`" Visual Basic keywords for use in implementing code modules. Thus the concept of "data hiding" in an object module was feasible.

Some of the computationally intensive components were initially developed in Visual Basic for initial evaluation purposes. However these were subsequently ported to C to make use of its processing efficiency and to utilise the modularization possible with Dynamic Link Libraries (DLL's) and its faster execution at run-time.

The main problem with Visual Basic as a product-creating tool (as opposed to a prototype-creating tool) is its speed of execution. It is an interpreted language unlike compiled languages (for example C). Whilst it permits the generation of what it refers to as an "EXE", it is not a complete stand-alone binary executable. It relies on a runtime DLL (`VBrun300.dll` for Visual Basic version 3.0 and `VBrun400.dll` for Visual Basic version 4.0) to perform the run-time interpretation. It also means that the run-time DLL must be on the computer for the generated "EXE" to execute, compared to fully compiled code which can run natively on the computer, typically without any support files. Thus, all applications execute slower than equivalent applications created from compiled code (although this mechanism does execute faster than the earlier fully-interpreted Basic and Visual Basic languages). This issue has since been improved

by the latest release of Visual Basic (version 5.0) as it generates object code along with an improved runtime DLL, now referred to as a "virtual machine" (`MSVB50VM.dll`).

Applications developed in Visual Basic have the additional requirement of several runtime libraries unique to Visual Basic. Each version has its own runtime library (`VBrun300.dll`, `VBrun400.dll` or `MSVB50VM.dll`) and several support DLL's (`StdOle2.dll`, `OleAut32.dll`, and `OlePro32.dll`). These must all be appropriately available either in the same directory as the executable, in the system directory, or on the "path". Any database operations necessitate various runtime libraries (for example "`MSAJT110.DLL`", and "`MSAES110.DLL`"), as well as the ODBC runtime library ("`ODBC.DLL`"). Whilst this was not really a difficulty, it introduced an additional amount of complexity to the installation and configuration process. If any of these DLL's are absent from the installation or are incorrectly registered, then the application will fail to operate.

Current trends in the software industry favour centralising commonly used functionality and packaging them in DLL's in order for several applications to utilise the functionality (often accessing the DLL concurrently). This modular architecture facilitates the continuous enhancement of shared libraries. As improvements in quality (such as more robust exception handling) and performance (such as multi-threaded execution and lower resource requirements) are generated, they can be of immediate benefit to all the multiple applications that leverage the upgraded DLL. Another benefit is to the application development process whereby less time is consumed designing, implementing and testing commonly available functionality; instead a DLL (or other shared library mechanism) is leveraged and the development effort can remain focused on the unique requirements that the current application demands. Therefore, reliance on multiple runtime libraries is the preferred model in the modern software development industry.

9.2. Installing the Application

Delivering the implemented design solution from a development and test environment to the production environment and final users can be a task fraught with difficulties if it is not managed in a failsafe manner. It can be broken into several distinct steps that are expanded upon below:

1. Creation of the physical media
2. Deployment of the files
3. Registration of binaries
4. Initial configuration

9.2.1. Creation of the Physical Media

Standard 3.5" floppy disks were selected as the distribution media. Due to the relatively small size of the installation files only two disks were required, therefore other media with greater capacity (such as writeable CD-ROMs) would have been chosen inappropriately.

When distribution disks were created with the software solution contained on them, all the ancillary files had to be included. All DLL's for serial, TCP/IP, and database connectivity, as well as the miscellaneous runtime libraries, had to be present. These distribution disks were then used as the master copies for delivering the application into the production environment.

9.2.2. Deployment of the Files

The application consists of several files of different types. Deploying the files involves placing each categorisation of file type into the appropriate directories on the target PC. Since the target environment exclusively used the Windows operating system, the deployment process was tuned for that operating system and file structure.

The directory for the application files could be specified by the person performing the installation, although it defaulted to the general applications

directory with a sub-directory named to reflect this application. Common system files that support the application (like `VBrun400.dll`)) were placed in the system directory whilst application-specific files (like initialisation files and data files) were placed in the same directory as the main application. Files specific to an individual user (like preference files and data files) can be located either in sub-directories contained in the main application or else in their system profiles directory within the system directory. To simplify the deployment process and maintenance, it was decided to locate them in sub-directories within the main application directory.

9.2.3. Registration of Binaries

In order for any application to interact with the operating system in a seamless fashion, it is necessary to register certain binary files. This allows the operating system to manage its resources in an efficient manner, for example by reusing resources across multiple applications. For this reason, an architecture based on DLL's for frequently used components is the preferred methodology. By adding the application binary to the system path, it can permit the application to be invoked from anywhere within the file system (although it was decided not to facilitate this since the application was targeted at a very specific need and therefore should not be globally invokable).

Registration of the included DLL binaries required a single system call. Thereafter, multiple applications could share the functionality exposed by these DLL's. If the DLL was already registered with the system, then it was necessary to compare the version and decide whether to replace it with the one being installed or to reuse the one already resident on the operating system.

9.2.4. Initial Configuration

The correct operation of the system depended on an accurate and successful configuration. This was particularly true for the remote connectivity issues where IP address conflicts and errors can be especially difficult to resolve. Thus it was

advisable that the configuration process should be carried out by an installation expert who was familiar with the IP addresses of all computers hosted within the domain of the system and the full paths to database files, along with the pertinent table names in the database schema.

Mistakes in specifying these parameters would cause several services to fail. It would be preferable to have this done automatically instead of requiring skilled human intervention, but the field of network resource resolution and utilisation is a large area of study in itself and was outside the scope of this research.

9.2.5. Third-Party Installation Product

Due to the complex nature of delivering the application into the production environment, a third-party product was used. Microsoft Install Shield was chosen to simplify the creation of the installation program.

The installation program handled the deployment of the various files into their respective directories and transparently managed the registration of DLL's in a customised manner, as well as updating the computer's registry for the newly installed software. Install Shield also produced the distribution media.

The end result was complete installation package that presented the user with a professional-looking graphical user interface for installing the software that was consistent with the look and feel of other Windows-based application installation programs.

9.3. Evaluation

9.3.1. User Appraisal

As stated in Chapter 3, the user requirements were developed with the assistance of a senior medical laboratory technician in St. James Hospital, Dublin. The histology laboratory in St. James Hospital was the test environment targeted for the deployment of the implemented design concept. Due to the fact that this

laboratory was fully operational and dealing with real samples, deployment of the implemented design concept was limited. The senior medical laboratory technician acted as an "evangelist" in supporting this limited deployment in areas under his direct control. Consequently, the performance in the test environment was specifically targeted at fulfilling his needs.

Primary among these needs was accessibility to Quality Control (QC) data from numerous different analysers. A single computer was already acting as a QC server in the laboratory where QC data was manually entered before performing statistical analysis and generating reports. Therefore, it was required to automate the collection of QC data from instruments and store them in the QC database before the analysis and reporting processes occurred. Thus, patient data collection and its real-time technical validation were not required tasks in the test environment. Neither was remote access to data already stored on the QC server (since all further processing of these data was performed locally on the QC server), nor data presentation (since processing of the data was achieved using existing tools). Tutorial instructions between computers were not exploited, but the "chat" utility was found to be somewhat useful for analysers that were located outside the confines of the laboratory.

The fact that several aspects of the design concept were not exploited in the test environment does not mean that these features were useless. With deployment in a different test environment, the functionality set employed would undoubtedly be different. Because the target environment was a medical one, evaluation of performance in other test environments which would exercise different aspects of the design concept were understandably not permitted.

As part of a different research programme [ROCH – 97], the design concept was leveraged to provide an asynchronous peer-to-peer data communication link (see Section 8.3.5). The application was for remotely viewing image data (X-rays and ultrasound scans) stored on a central image database server.

9.3.2. Suitability of Architecture

As stressed in the design chapter (Chapter 4) a modularised architecture was developed using existing standards and protocols wherever possible. This approach favoured a continuous development process where functional modules were continually developed beyond their initial design specifications, or even replaced by equivalent modules.

Throughout this document there were references to several modules (Figure 9.1) which could be successfully integrated within the design concept:

- instrument interface for receiving and parsing data from a clinical analyser via a serial port (Chapter 5)
- database interface for interacting with SQL-compatible databases via ODBC (Chapter 7)
- technical validation of patient data by statistical analysis of boundary values to eliminate gross errors and highlight deviations
- Quality Assurance module to analyse the performance and maintenance requirements of instruments
- GUI representation of data in tabular form or scatter plots for use by a requesting physician (also useful for QA analysis)
- remote connectivity for connecting to isolated computers using TCP/IP (Chapter 8)
- chat utility for interaction between a remote operator and a laboratory technician (Chapter 6 and Chapter 8)
- tutorial utility for instruction in instrument operation, maintenance protocols and repair

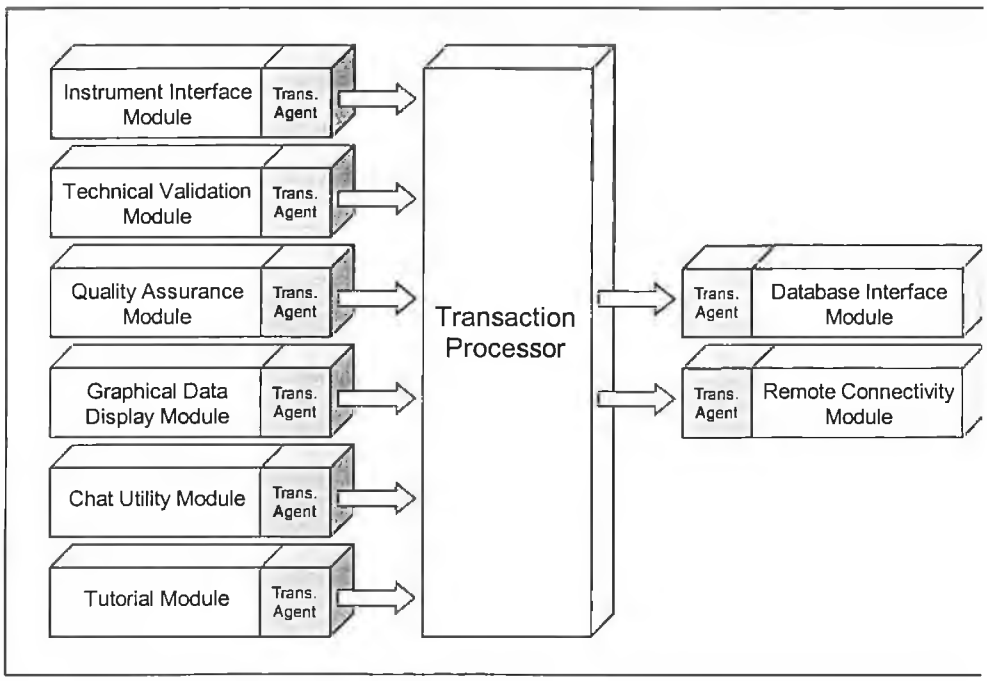


Figure 9.1 Potential modules for interaction with the Transaction Processor

Each module has its own unique implementation details (the instrument interface has serial communication and parsing components, the chat utility has user interaction components based in the Windows environment), but they all contain a Transaction Agent. This Transaction Agent permits the modules to dispatch service requests in the form of transactions via the Transaction Processor to other modules. These modules fulfill the service requests and return an acknowledgement response to the requesting modules. Thus, each module can leverage the services of other modules, thereby increasing their functionality without embedding that specific functionality within them. This kind of distributed architecture scales extremely well as new functionality is required and additional modules are introduced since existing modules do not necessarily need to be replaced or updated.

Due to the diversity of instruments in use in the medical domain, the adopted instrument interface is incapable of handling all cases. Whilst it is particularly suited to instruments that output constant length process information, variable length messages are not capable of being processed. For those types of outputs, a

custom driver or an interface module like the AII (Section 5.7.2) should be utilised.

Since almost every analyser being interfaced to a host computer requires a unique interface specification to parse its unique message format, the adopted interface module must be simple and rapid to implement. Whichever instrument interface modules are used, they must contain an intuitive user interface which builds on a user's familiarity with the standard Windows GUI. As increasingly complex instruments are interfaced, the interfacing module must likewise gain in complexity without sacrificing any of its intuitiveness.

The remote connectivity functionality was one of the primary user requirements in order to make remotely located analysers accessible to the LIS for statistical monitoring and maintenance. This was required in order to reduce the frequency of traveling to remote analysers. This remote connectivity functionality utilises the TCP/IP network protocol. This data communication protocol is used throughout the world as the enabling technology of the World Wide Web (WWW) and as such is widely supported on numerous hardware and operating system platforms. The result is an extensively adopted and robust protocol, the advantages of which are consequently inherited by the design implementation's use of it.

For regions of the healthcare domain which do not have access to a network connection, telephony data communication is supported. This was introduced to exploit the extensive private telephone network which is in operation in St. James's Hospital, Dublin, to avoid the invasive installation and additional cost of network connections wherever a clinical analyser happens to be located. The private telephone infrastructure extends to the farthest reaches of the hospital domain. All instruments are within sufficient proximity to a telephone connection point so that signal degradation due to excessively long cabling is not a problem. This communication medium can even be extended beyond the geographical boundaries of the hospital's private network. Thus, instruments connected to a host computer external to the hospital (for example in a GP's private practice) could dial up the centralised LIS server for transaction-based interactions.

Obviously this exposes the system to potential security breaches and this must be addressed.

However, telephony data communication suffers from time delays due to dialing to establish the call and then a handshaking procedure to synchronise baud rates. Therefore it may not be appropriate for some analysers that timeout if they do not receive a rapid responses to requests. Additionally, over a telephone line only one remote host computer can communicate with the database server at any one time. This will result in other remote host computers who require services from the server to be locked out from the accessing these services. A partial solution to this would be to equip the server with additional modems. This would allow a remote host computer that receives an engaged dialling tone during call set-up to try the next telephone line, and so on until it is able to connect to the server and establish a transaction link.

9.3.3. Extensibility of the Architecture

Choosing a design architecture which is database-centric provides a viable progression path. For example, consider data from clinical analysers being required for statistical monitoring of analyser performance or for the investigation of medical trends. The instrument interface module gathers the data necessary for these kinds of analyses. Thus, modules performing these new kinds of processing can extract the necessary information from the data store using simple SQL statements. These modules that require this kind of information for additional processing would use a Transaction Agent which generates the relevant SQL statements. These SQL statements are then passed to the appropriate service provider (the database interface module) via the Transaction Processor. Thus, the existing architecture is leveraged to provide new functionality.

Alternatively, applications which generate these SQL statements can be quite easily developed in Visual Basic and can connect to the SQL-compatible databases using the Open DataBase Connectivity (ODBC) protocol as supplied with the Data Control supplied with all versions of Visual Basic from version 3.0

onwards. So, whilst the architecture is not leveraged, it permits the development of stand-alone applications that can interact with data obtained using the architecture. For example, if the billing department needs to know what tests were performed on a patient's sample, an application could be developed which queries the results database. The required SQL statement would contain the patient's name as a conditional clause in the `SELECT` statement and return only the patient's tests.

Database querying applications of this type would be created separately from the main application presented in this research and would therefore be independent of whatever future developments might be implemented. These separate development paths can comfortably co-exist so long as the location information for the database(s) remains available to these types of secondary applications.

This gives an indication of the potential weakness of the design architecture: being so heavily reliant on database storage of clinical analysis data means that the databases must be very well designed and administered. A weakness in the relational model employed in the DBMS would invariably impact on the main application. It was decided that this frailty was a possibility, not a probability and as such was far outweighed by the flexibility of developing derivative applications to support information processing.

9.4. Future Developments

Advances in the software development industry occur at such a rapid pace that obsolescence can be measured in years rather than decades. The longevity of an application solution can be maximized by adopting a standards-based, modular architecture. Standards-based technology leverages currently supported methodologies that will continue to be supported for some time due to the fact that they are standards. A modular architecture facilitates the incremental updating of obsolete components without impacting the solution as a whole. Thus, as future advances introduce new ways of performing certain functionality, the module responsible for delivering that functionality would be updated in isolation.

Due to the uncertainty of future software development practices, an analysis of future developments can be split into immediately desirable enhancements and speculative possible improvements.

9.4.1. Enhancements

Multiple Analysers

Only a small selection of clinical instruments were used in the evaluation of this design solution. As described earlier (Section 2.5, "Clinical Analysers"), there is a multitude of clinical instruments in use and support for a wider set of analysers would be desirable. Since very few analysers support a standardized format for data communication, each analyzer would require its own Instrument Interface Module.

The larger analysers usually have a dedicated workstation. In order to include these types of complex instruments into this application, they would require a very specialised interface. It would be necessary to write code that would reside directly on the dedicated workstation and would be very specific to the analyser and how it operates.

Support for more diverse instruments would undoubtedly load the design implementation in new ways that would expose new areas of opportunity for improving the overall architecture.

Additional Modules

Due to the extensible nature of the design solution, modules supporting different feature sets could easily be integrated. Some of these were listed earlier (Section 9.3.2) and are reproduced here in an abbreviated format for convenience:

- technical validation of patient data
- Quality Assurance module
- GUI representation of data
- tutorial utility

Security

In the data-sensitive area of the healthcare environment, security of data is vitally important. Future developments would include several data security features. The most obvious of these is data encryption, particularly for the transmission of data over telephone connections. Public switched telephone networks are not a secure media and so appropriate measures would be necessary.

Public key encryption has received a lot of attention recently and is gaining widespread recognition as an enabling technology for commercial transactions in the Internet domain. Depending on the degree of encryption (or the number of bytes in the encryption key: 40-bit or 128-bit or the new digitally signed certificates) various levels of security can be achieved whilst affecting processing overhead. The larger the encryption key, the larger the processing overhead. A compromise would be necessary which would address the two concerns. For the relatively slow data transmission speeds of a telephone link, strong encryption could be used since processing time would be a smaller percentage of the overall transaction time. As transmission speeds increase, time spent processing an encryption algorithm becomes a more significant percentage. Therefore, the degree of encryption could be "tuned" relative to the amount of processing time required as a percentage of the overall time.

The remote connectivity module would handle encryption services. Whenever the module has to send data to a remote host, it would encrypt it using the selected encryption mechanism and then send it. The remote receiving module would then decrypt it before passing the data through to its Transaction Processor.

Authentication

For data communication within the bounds of the Hospital Information System (HIS), the issue of authentication becomes more important than data security so long as the HIS is adequately protected by firewalls from external interference. Correct identification of computers which attempt transactions with a server and

the authentication of the user driving the transaction must be obtained before critical information is divulged.

Message Server

A core element of the design solution is the Transaction Processor. This handles the correct routing of messages between different components and attempts to provide a degree of reliability by utilising a send-acknowledge architecture.

A Message Server would transparently guarantee the reliable delivery of messages between components. It would have an extremely high level of fault tolerance even through catastrophic system failures like power outages. It would achieve this through the use of a persistent data repository. Message Server products also typically provide logging and archival functionalities.

In the medical environment where data are sensitive and usually time sensitive, a guaranteed delivery subsystem would be very desirable. The impact on the design solution would be quite small since a message centric architecture was utilised. Each component would simply give a message to the Message Server agent without the need for any confirmation or acknowledgement since delivery is assured.

However these kinds of products usually have high licensing fees. This would probably prohibit their adoption under the scope of this research and its nature as an interim solution until a standard is accepted.

Localisation

As globalisation continues with the integration of widespread clinical resources, foreign countries could conceivably be included within the scope of a single healthcare provider. Thus, communication within the healthcare infrastructure may require multi-lingual support.

Localisation is the customisation of all user prompts into a language native to its deployment environment. It can be accomplished by using string tables for all text that is visible to the user. Therefore, each string entry in the table would have a corresponding entry for each language it supports. When the user interface needs to display some information to the user, it would use the corresponding table entry.

Due to the modular architecture adopted, only modules that contain user interface functionalities would require the use of localisation string tables. Thus, localisation support could be implemented on a per-module basis as the need arose.

9.4.2. Improvements

The client-server paradigm has been in use for several decades and has proven its worth as a scalable solution. This was the foundation of the architectural choices utilized in the design solution. However, there is a trend to favouring a Web-based paradigm.

A Web-based paradigm uses the Internet as the enabling communication medium. TCP/IP remains as the transport and network layers, but the application layer is considerably different. All user interface modules would be implemented using a Web browser interface with HyperText Markup Language (HTML) as the software language of the presentation layer. Interaction with the centralised server would be through HTTP requests to a web server with server-side processing to handle them.

Server-side processing is typically handled by scripts that the web server invokes based on HTTP requests. These scripts use an API called Common Gateway Interface (CGI) and can be written in Perl or C. Microsoft's Active Server Pages (ASP) are becoming increasingly popular due to their use of a version of VB Script and close integration with other Microsoft technologies like the Component Object Model (COM) and Internet Information Server (IIS).

Platform independent java provides another alternative through Java Server Pages (JSP) and Java Servlets.

Since all the resources reside on the server it liberates the design solution from client software modules. Therefore, maintenance and upgrades can be carried out in one place instead of having to follow the four-steps described above for re-installing a newer version of the design solution on every machine it was ever installed on.

Whilst this architecture reduces the processing load on the client, it consequently increases the processing load on the server. Thus more powerful hardware is required for the server than in a more distributed client-server architecture.

Another disadvantage of a Web-based paradigm relates to some of the peer-to-peer tasks currently supported by the design solution. The more automated tasks of the design solution could not be achieved using a simple web server and web browser arrangement. Since several scenarios exist where a remote machine must service requests (for example, to perform a quality assurance validation), the computer connected to the remote analyzer must have its own web server. This approach raises all the associated concerns with security and maintenance multiplied by the number of machines that host a web server and therefore does not scale terribly well.

Due to the nature of a Web-based solution, a substantial redesign of several of the modules of the design solution would be necessary. The additional software requirements and more powerful hardware would introduce a financial commitment that would have to be made at a very early stage. Therefore, the decision to implement a Web-based solution is typically part of a much larger strategic technical decision, for example as part of a global e-strategy. Within the scope of this research, a Web-based solution would not be a practical answer to the limited set of technical requirements of this research.

10. References

[AAHO – 86] A. V. Aho, Ravi Sethi, & J. D. Ullman: "Compilers, Principles, Techniques, and Tools"; Addison-Wesley, 1986.

[ALLE – 91] D. A. Aller: "Open Systems Architecture: The Multivendor Hospital Information System"; In Aller RD, Elevitch FR (eds) Clinics in Laboratory Medicine Volume 11, No. 1, pp. 73 - 81, March 1991.

[ARON – 84] T. Aronsson & T. Groth: "Nested Control Procedures for Internal Analytical Quality Control"; Scand J Clin Lab Invest, No. 44, Supplement 172, 51, 1984.

[ARS – 80] J. Arsenault & J. Roberts: "Reliability and Maintainability of Electronic Systems"; Computer Science Press, Inc., Rockville, Maryland, 1980.

[ASTM – 90] ASTM E31.11 Committee, C. Mc Donald (chairman): "Standard Specification for Transferring Clinical Observations Between Independent Computer Systems"; (Revision, E1238-90), ASTM, Philidelphia, 1990.

[BAKK – 88] R. A. Bakker, M. J. Ball, J. R. Scherrer & J. L. Williams: "Towards New Hospital Information Systems"; Elsevier Science Publications, Amsterdam, The Netherlands, 1988.

[BALL – 91] M. J. Ball, R. I. O' Desky & J. V. Douglas: "Status and Progress of Hospital Information Systems (HIS)"; International Biomedical Computing, 29, pp. 161 - 168, 1991.

[BEEL – 87] M. F. Beel & R. Sappenfield: "Medical Monitoring: What is it, How can it be Improved?"; An J Clin Pathol 87: 285 - 288, 1987.

[BENS – 80] E. S. Benson: "Initiatives Towards Efficient Decision Making and Laboratory Use"; Human Pathology Volume 11, No. 5, Sept. 1980.

[BOAR – 94] Board of Directors of the American Medical Informatics Association: "Standards for Medical Identifiers, Codes, and Messages Needed to Create an Efficient Computer-Stored Medical Record"; J Am Med Informatics Assoc., 1(1): 1-7, 1994.

[BURN – 93] J. Burns & J. Hourihan: "Simplifying LAN_WAN Integration"; Wellfleet Communications, Inc., July 1993.

[BURR – 90] M. Burrett: "Current Analytical Approaches to Measuring Blood Analytes"; Clin. Chem., (8 pt 2): 1562 - 1566, August 1990.

[CARR – 86] K. R. Carr: "Expert Systems Application for Diagnosis and Trouble-Shooting of Computer Systems and Instrumentation"; Oak Ridge National Laboratory, DE 87000219, 1986.

[CHAM – 86] A. M. Chambers, J. Elder, & D. StJ. O' Reilly: "The Blunder Rate in a Clinical Biochemistry Service"; Ann Clin Biochem, 223: 470 - 473, 1986.

[CHEN – 91] Patrick Chen: "The Joy of Telecomputing"; 1991.

[CONN – 80] D. P. Connelly & B. Steel: "Laboratory Utilisation Problems and Solutions"; Arch Pathol Lab Med 104, 59 - 62, 1980.

[COON – 58] R. Coon, L. Crowley & J. Utterback, "Automation in Small Hospital Laboratories"; paper presented at the annual meetingof the American Society of Clinical Pathologists, Nov. 5th 1958.

[COTE – 73] R. A. Côté, "Use of a Nomenclature of Medicine in a Medical Information Management System"; Bulletin of the Canadian Association of the Medical Record Librarians, Volume 5, 4, Nov. 1973.

[COUL – 94] G. Coulouris, J. Dollimore & T. Kindberg: "Distributed Systems, Concepts & Design"; Addison-Wesley, 1994. ISBN 0-20162-433 8.

[DATE – 93] C. J. Date & H. Darwen: "A Guide to the SQL Standard: A User's Guide"; Addison-Wesley, 1993. ISBN 0-201-55822-X.

[DEMO – 92] G. J. E. De Moor: "Standardisation in Healthcare Informatics and Telematics in Europe: CEN/TC 251 Activities"; Medical Informatics 17, 2:133-140, 1992.

[DITO – 70] W. Dito: "Programmable Calculations for Quality Control in Clinical Chemistry"; in Knights EM Jr. (ed): Mini-computers on the clinical laboratory; Springfield, IL, Thomas CC, pp 9-29, 1970.

[DONO – 94] J. J. Donovan: "Business Re-engineering with Information Technology"; Prentice-Hall, Inc., 1994. ISBN 0-13-311028-1.

[DUMA – 95] A. Dumas: "Programming WinSock"; Sams Publishing, 1995.

[GAFF – 96] P. Gaffney: "Medical Infomatics in a Clinical Laboratory Environment", MSc Thesis, Trinity College Dublin, 1996.

[GROF – 94] J. R. Groff & P. N. Weinberg: "LAN Times® Guide to SQL"; Mc Graw Hill, 1994. ISBN 0-07882-026 x.

[GROT – 91] T. Groth & H. Moden: "A Knowledge-Based System for Real-Time Quality Control and Fault Diagnosis of Multi-Test Analysers"; Comp Meth Program Biomed, 34: 175, 1991.

[HACK – 93] R. D. Hackathor: "Enterprise Database Connectivity"; Wiley, 1993.

[HALS – 92] F. Halsall: "Data Communications, Computer Networks and Open Systems"; Addison-Wesley. 1992.

[HL7 – 90] HL7 Committee: "The HL7 Standard for Communications of Health Care Information"; Version 2.1, Chicago, IL, July 1990.

[HOLM – 92] K. Holmberg & A. Folkeson: "Operational Reliability and Syatematic Maintenance"; Elsevier Applied Science, 1992. ISBN 1-85166-612 5.

[HOWA – 91] J. H. Howanitz & P. J. Howanitz: "Principles of Laboratory Medicine. In Laboratory Medicine - Test Selection and Interpretation"; Churchill Livingstone Inc., New York, 1991.

[IUP – 78] International Union of Pure and Applied Chemistry, Commission on Automation: "Characteristics and Attributes of Instruments Intended for Automated Analysis in Clinical Chemistry"; IUPAC Inf. Bull., No. 3, pp. 233 - 240, 1978.

[JAME – 95] K. James & K. Cope: "Internet Programming"; Jamsa Press, 1995.

[JOHN – 68] W. L. Johnson. "Automatic Generation of Efficient Lexical Analysers Using Finite State Techniques"; Communications of the ACM, Volume 11 No. 12, 1968.

[KADI – 66] A. Kadish: "Instrumentation Methods for Predictive Methods"; edited by TB Weber and J Poyer, Pittsburg, PA, Instrument Society of America, 1966.

[KELL – 74] C. R. Kelly & J. Manlin: "Ambulatory Medical Care, Quality, Determination by Diagnostic Outcome"; JAMA 227, pp. 113- 115, 1974.

[KENN – 94] Robert J. Kennelly: "New IEEE Standard (IEEE P1073) Enables Data Collection for Medical Applications"; SCAMC, 1994.

[KRIS – 81] B. B. Kristensen & O. L. Madsen: "Methods for Computing LALR(k) Lookahead"; ACM Transactions on Programming Languages And Systems, Volume 3 No. 1, January 1981.

[LESK – 75] M. E. Lesk: "Lex - A Lexical Analyser Generator"; Computer Science Technical Report No. 39, Bell Labs, Oct. 1975.

[LESK – 97] M. Lesk: "Going Digital"; American Scientific, Mar. 1997.

[LEVI – 92] John R. Levine, Tony Mason & Doug Brown: "Lex & Yacc"; O' Reilly & Associates Inc., Oct. 1992.

[LEWA – 92] K. Lewandroski, et al: "Implementation of Capillary Blood Glucose Monitoring in a Teaching Hospital and Determination of Program Requirements to Maintain Quality Testing"; Am. J. Med., 93 (4): pp. 419 - 426, Oct. 1992.

[LYNC – 93] D. C. Lynch & M. T. Rose: "Internet Systems Handbook"; Addison-Weslesy, 1993. ISBN 0-20156-741 5.

[MALK – 86] D. B. Malkoff: "Real-Time Fault Detection and Diagnosis. The Use of Learning Expert Systems to Handle Timing of Events"; Navy Personnel Research and Development Center, ADA 1746551/1, Nov. 1986.

[MANT – 97] Charles Mantel: "The Long-lived RS-232 Standard Coming to an End"; Electronic Buyers News (EBN), May 1997.

[MARK – 88] V. Marks: "Essential Consideration in the Provision of Near-Patient Testing Facilities"; Ann. Clin. Biochem., 25: pp. 220 - 225, 1988.

[MICR – 95] Microsoft Windows Help: "Custom Control Reference"; copyright 1990 – 1995, Microsoft Corporation, 4.10.1998.

[MICR – 96] Microsoft White Paper: "The Microsoft Windows Telephony Platform: Using TAPI 2.0"; Microsoft Corporation, 1996.

[MOLL – 90] D. P. Moll: "A Diagnostic Expert System"; Biomed Tech Berlin, Apr. 1990; 35 (4): 62 - 68.

[MORT – 93] "Reference Manual for MKS Lex & Yacc"; 3rd edition, Mortice Kern Systems Inc., 1993.

[MURR – 93] R. B. Murray: "C++ Strategies and Tactics"; Addison-Wesley, 1993. ISBN 0-201-56382-7.

[NORG – 95] Thomas Norgall: "Open Communication Among Medical Devices Using ISO MAP Application Protocol MMS"; CEN TC 251 WG5 N 75.

[O'MO – 88] R. O' Moore: "Decision Support Based on Laboratory Data"; Meth. Inform. Med. 27: pp. 187 - 190, 1988.

[OPEN – 95] OpenLabs (A2028) Consortium: "Specification of the Architecture for an Open Clinical Laboratory Information System Environment"; AIM Programme (A2028), ARCH-2, Jan. 1995.

[ORFA – 96] R. Orfali, D. Harkey & J. Edwards: "The Essential Client - Server Survival Guide"; Wiley, 1996.

[PETE – 89] K. E. Petersen, et al: "Reliability Analysis in Life Cycle Cost Estimation for Small Wind Turbines"; SRE Symposium, Stavanger, Norway, 1989.

[ROBE – 80] J. A. Roberts & J. E. Arsenault: "Limited Life Items"; Reliability and Maintainability of Electronic Systems, Computer Science Press Inc., Rockville, Maryland, 1980.

[ROCH – 97] R. Rochford: "Compression and Storage of Medical Images"; MSc Thesis, Dublin City University, 1997.

[RUBI – 79] P. Rubin, S. Bradbury & K. Prowse: "Comparative Study of Automatic Blood-Gas Analysers and Their Use in Analysing Arterial and Capillary Samples"; British Medical Journal 1: pp, 156 - 158, 1979.

[SERG – 89] Sergent & Schumacher: "The IBM PC from the Inside Out"; p.65.

[SKEG – 64] L. Skeggs & H. Hochstrasser: "Multiple Automatic Segment Analysis"; Clin. Chem. 10: pp. 918 - 936, 1964.

[SPAC – 87] K. A. Spacman, D. P. Connolly: "Knowledge Based Systems in Laboratory Medicine and Pathology"; Arch Pathol Lab Med, Volume 111, pp. 116 - 119, 1987.

[TANE – 96] A. S. Tanenbaum: "Computer Networks"; Prentice Hall, 1996. ISBN 0-133-49945-6.

[VASK – 93] D. Vaskevitch: "Client/Server Strategies"; IDG Books, 1993. ISBN 1-56884-064-0.

[WEST – 75] G. Westlake: "Microprocessors, Programmable Calculators and Minicomputers in the Clinical Laboratory"; in Endlander D (ed): Computers in Laboratory Medicine, New York, Academic Press, 1975.

[WEST – 84] J. Westgard & T. Groth: "Computer Systems for Implementation of Internal Quality Control Procedures"; Scand. J. Clin. Lab. Invest. 44 (suppl. 172): pp. 203 - 207, 1984.

[WINK – 87] P. Winkel, U. Adam: "Components of Biologic and Analytic Variation"; Laboratory Quality Assurance, Mc Graw-Hill Inc., 1987. ISBN 0-07-029672-3: 166-182.

Appendix A – Scenarios

A.1. Scenarios for archived data

"Draft Minutes of the 11th CEN TC 251 WG5 Meeting at Dublin Trinity Medical Centre, 8th to 10th September 1994"

(CEN/TC 251/WG5-N101)

1. EEG recording for 30 minutes. Signals sent to amplifier, A/D converter then recorded in a computer. Neurophysiologist makes a diagnosis record.

2. Ambulatory EEG. Recordings on tape recorder for 24 hours. Play-back into A/D converter and into computer. Again diagnosis record is made.

3. Sleep lab. Transducers - amplifier - A/D converter - computer recordings made for up to 24 hours. Data is archived then analysed with waveform display. Analysis provides new parameters. Report on paper. Statistics on multiple patients.

4. Epilepsy monitoring. EEG recordings for 1 - 14 days. Up to 64 channels EEG. Also video recording of patient state. Ethernet transports data from PC to UNIX where there is Big-endian/Little-endian data representation problem. Analysis produces a brain map of EEG.

5. Patients monitors, etc. feeding data into a data logger. Data flow is uni-directional. Many kinds of data but few problems of time synchronisation, etc.

6. Monitoring display - pumps, ventilators, etc. feeding uni-directionally into a larger patient monitor. Issues are time synchronisation, real-time, alert handling such that alerts on a gas monitor may be desired on the larger monitor.

7. Monitoring system with bi-directional data transfer. Added problems are real-time, alert receipt, alert acknowledge, device setting receipt, device setting changes.

8. Interoperable viewing system. Uni-directional network of monitors sending data to a central point monitor.

9. Interoperable with patient transfer. Central point or data logger needs to transfer data from bed A to bed B.

10.Third party diagnostic analyser may send diagnostic reports back to the bedside monitor.

11.Patient is moved around the hospital where continuity monitoring is a problem. Loading transport monitor data into a destination monitor introduces time-stamping as a concern.

A.2. Scenarios for Analyser Communications

CEN/TC251 WG5 CALM Modelling Activity Modelling Workout Version 6

1. Analyser sends **results** to LIS (zero, one, or more samples; qc or not)
2. LIS sends **orders** to analyser (worklist or single sample)
3. Analyser sends a **query** to LIS (query=?)
4. LIS sends a **query** to analyser (query=?)
5. Analyser sends **status** to LIS (function, production, inventory, configuration)
6. LIS sends additional **sample** information to analyser
7. LIS sends a control message to analyser (**command**)
8. LIS sends **results** to analyser
9. Analyser sends special **data** to LIS (qc, calibration, graphics, instrument)
10. LIS sends **communication check** to analyser
11. Analyser sends **communication check** to LIS
12. LIS sends microbiological orders or **batteries** to analyser

Appendix B – Serial Communications

B.1. Physical Connection

Serial ports are an industry standard using one of two connector types - 9 pin sub-D type (Figure B.1) or 25 pin sub-D type (Figure B.2). Both types require the use of only 3 wires for the most basic data transfers. One wire acts as a signal ground and the other two wires act as the signal carriers for data flowing in each direction. Therefore for data transfer in only one direction (simplex data transfer), only two wires are needed for the basic service whereas three are required for bi-directional (duplex) data transfer. Other wires are defined for hardware handshaking to produce more reliable data transfers at higher speeds over greater distances or in an asynchronous environment.

Pin	Signal	Function
1	DCD	Data Carrier Detect
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data Terminal Ready
5	Gnd	Logic ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RD	Ring Detect

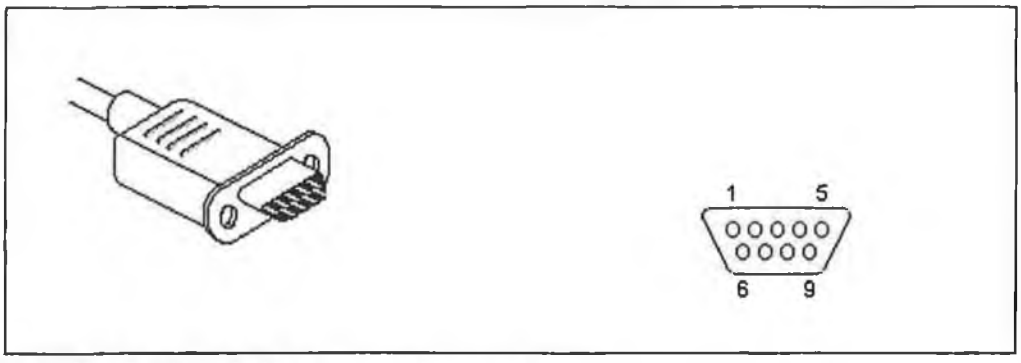


Figure B.1 9-pin sub-D type serial connector

Pin	Signal	Function	Pin	Signal	Function
1	FG	Frame Ground	14	STxD	Secondary TxD
2	TxD	Transmit Data	15	TClk	Transmit Clock
3	RxD	Receive Data	16	SRxD	Secondary RxD
4	RTS	Request To Send	17	RClk	Receive Clock
5	CTS	Clear To Send	18		(unassigned)
6	DSR	Data Send Request	19	STRS	Secondary TRS
7	Gnd	Signal Ground	20	DTR	Data Terminal Ready
8	DCD	Data Carrier Detect	21	SQ	Signal Quality
9	V+	Positive Voltage	22	RD	Ring Detect
10	V-	Negative Voltage	23	DRS	Data Rate Selector
11		(unassigned)	24	SCTE	Secondary Transmit
12	SDCD	Secondary DCD	25	BUSY	Busy
13	SCTS	Secondary CTS			

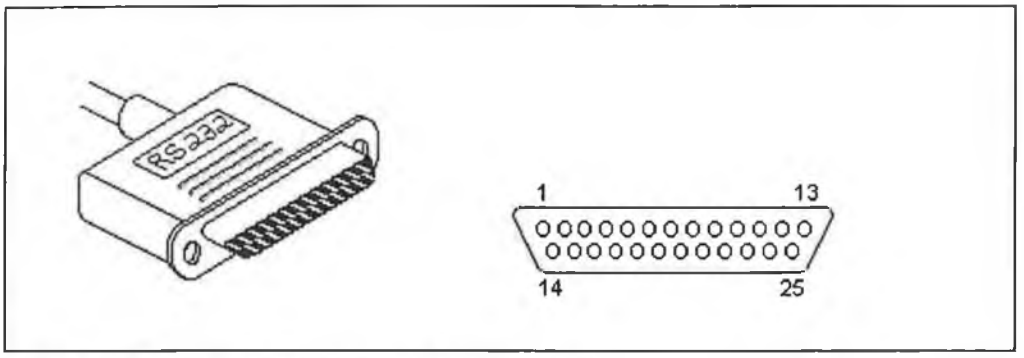


Figure B.2 25-pin sub-D type serial connector

B.2. Data Formatting

Whenever data are transmitted over a serial link between two serially-connected Data Communications Equipment (DCE) terminals they are packaged within digital bits to provide hardware-level synchronisation and error control. The format is shown in Figure B.3. The extra data bits are not seen by the receiving terminal because they are used by the hardware associated with the serial port and are removed before the data is passed up to the next level of processing.

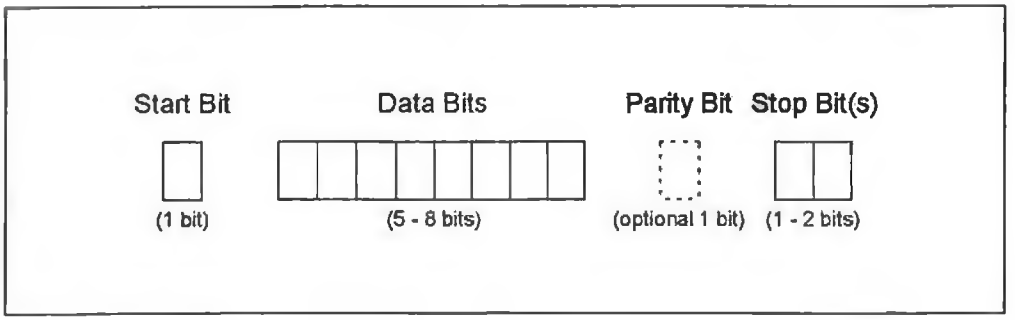


Figure B.3 Bit-by-bit breakdown of serial data format

Once the data has been correctly received by the serial port hardware and the extra communication bits removed, the application layer receives the data as a stream of sequential bytes without any semantic organisation or grouping. It is the responsibility of the top-level application program to interpret and process this stream of data.

Appendix C – Transactions

C.1. Components of the Transaction

The transaction header contains two parts - the command and its qualifiers, if any. There is sufficient information to indicate the type of transaction which is taking place and therefore enable the Transaction Processor to resolve the target of the transaction.

The elements of the header are:

Element	Description
Command	2 digit hex constant from a lookup table
MIC	3 character plus 1 number unique Module Identification Code
TIN	4 number unique Transaction Identification Number
data & time	14 characters in the form of hhmmssddmmyyyy
length	4 numbers giving the number of characters in the transaction
qualifier	occasional element for additional processing information
delimiter	to indicate the end of the header, character ':' is used

The transaction body contains as many characters as necessary for the transaction, and may even be zero. There is a NULL character ('\0') used as a delimiter of the message field to indicate the end. The transaction length in the header could be used for evaluating the size of the message, but this delimiter technique is a simpler and also guarantees that the string in the message field is always null-terminated.

Element	Description
Message	0 to ? characters
delimiter	to indicate the end of the message field, character '\0' is used

E.2. Definitions of Transaction Types

Command	Code	Qualifier(s)	Message
REGISTER	0x00	*(nothing)*	delimited text strings
ACK	0xF0	REGISTER	Module Identification Code
OPEN_DB_LOCAL	0x01	*(nothing)*	specify database (OPEN)
ACK	0xF0	OPEN_DB_LOCAL	*(nothing)*
OPEN_DB_REMOTE	0x02	IP address	specify database
ACK	0xF0	OPEN_DB_REMOTE	*(nothing)*
OPEN_CHAT	0x03	IP address	*(nothing)*
ACK	0xF0	OPEN_CHAT	*(nothing)*
OPEN_TUTOR	0x04	IP address	*(nothing)*
ACK	0xF0	OPEN_TUTOR	*(nothing)*
WHO_ARE_YOU	0x05	*(nothing)*	*(nothing)*
I_AM	0xF5	WHO_ARE_YOU	delimited strings
WHAT_STATUS	0x06	*(nothing)*	*(nothing)*
STATUS	0xF6	WHAT_STATUS	available, busy, off-line, or a time-out (no answer)
DB_STORE	0x10	*(nothing)*	"INSERT INTO..."
ACK	0xF0	DB_STORE	SQL response
DB_STORE	0x11	IP address	"INSERT INTO..."
ACK	0xF0	DB_STORE	SQL response
DB_EXTRACT	0x12	*(nothing)*	"SELECT FROM..."
ACK	0xF0	DB_EXTRACT	SQL response
DB_EXTRACT	0x13	IP address	"SELECT FROM..."
ACK	0xF0	DB_EXTRACT	SQL response
CHAT_SEND	0x14	number of chars.	character(s)
TUTOR_REQUEST	0x15	list or tutorial	*(nothing)* or tutorial
TUTOR_SEND	0xE5	number of entries or tutorial format	delimited list or tutorial file
CLOSE_LINK	0x07	*(nothing)*	*(nothing)*
ACK	0xF0	CLOSE_LINK	*(nothing)*
TERMINATE	0x08	*(nothing)*	*(nothing)*
FAIL	0xFF	*(nothing)*	the original command code

(Note: acknowledgement transactions are indented)

C.2.1. Opening a Connection

OPEN_DB_LOCAL	specify which local database
OPEN_DB_REMOTE	specify the machine IP address and database
OPEN_CHAT	specify the machine IP address
OPEN_TUTOR	specify the machine IP address

C.2.2. Negotiating Identities

WHO_ARE_YOU	own identity information plus username
I_AM	delimited string of identity information plus username

C.2.3. Establishing Privileges

ACCESS	coded level of accessibility (depends on previous identity negotiation)

C.2.4. Additional Information

(For future developments)

C.2.5. Status Enquiry

WHAT_STATUS	no qualifier
STATUS	one of the constants (available, busy, off-line, or a timeout delivering no answer)

C.2.6. Data Storage

DB_STORE	SQL statement "APPEND TO..."

C.2.7. Data Retrieval

DB_EXTRACT	SQL statement "SELECT FROM..."

C.2.8. Interactive Chat

CHAT_SEND	number of characters; character(s)

C.2.9. Tutorial Instruction

TUTOR_REQUEST	tutorial list or individual tutorial
TUTOR_SEND	whatever format the tutorial is in; the data

C.2.10. Closing a Connection

CLOSE_LINK	(wait for ACK)
TERMINATE	(abrupt without ACK)

Appendix D – Structured Query Language

D.1. Storing Data

The format of a SQL statement for storing data [DATE – 93] is:

```
INSERT INTO target [(field1[, field2[, ...]])] VALUES (value1[,
          value2[, ...])
```

where

`target`	name of the table to append records to
`field1, field2`	names of the fields to append data to
`value1, value2`	values to insert into the specific fields of the new record

Each data value is inserted into the field that corresponds to the data's position in the list: `value1` is inserted into `field1` of the new record, `value2` into `field2`, and so on. For SQL compliance values must be separated with a comma, and text fields enclosed in double quotation marks (" ").

D.2. Retrieving Data

The format of a SQL statement for retrieving data [DATE – 93] is:

```
SELECT [predicate] {*|table.*|[table.]field1[, [table.]field2[,
          ...]]} FROM tableexpression [, ...] [WHERE...]
```

where

`predicate`	one of `ALL`, `DISTINCT`, `DISTINCTROW`, of `TOP` to restrict the number of records returned (default is `ALL`)
`*`	that all fields from the specified table are selected
`table`	the name of the table containing the specific records
`field1, field2`	the name of the fields to retrieve data from
`tableexpression`	the name of the table containing the specific records
`WHERE...`	specifies a conditional clause to reduce the records returned

Appendix E – The WinSock API

E.1. Creating a Server Socket

Before the WinSock API's can be used, they must first be initialised. The `WSAStartup()` function (see later sections for complete function definitions) is used for this purpose and allows the version of the WinSock API to be verified. Various other functions can be used at this stage to get additional information such as computer Internet addresses in the form of names or dotted-decimal numbers, or services in the form of name or number.

To create the end-point of communication, the `socket()` function returns a unique socket handle on success. It is at the call to this function that the type of communication is specified – connection-oriented stream communication or connectionless datagram communication.

To receive network requests the socket handle must be associated with a specific protocol port using the `bind()` function. This function tells the socket implementation which protocol port to use for data communication, and the socket implementation informs the Transport Layer to deliver all data received for that port to the WinSock API.

E.2. Creating a Client Socket

Once the WinSock API's have been initialised as above, the end-point of communication is created using the `socket()` function. As above, it is at the call to this function that the type of communication is specified – connection-oriented stream communication or connectionless datagram communication.

In a Client/Server communication configuration, the Client initiates communication. Therefore the Client must know the protocol port at which the Server is expecting communication. However, the reverse is not necessary since the Client informs the Server which protocol port it is using. The Client socket can

be assigned a specific protocol port using the `bind()` function, but usually this is not done and the WinSock API chooses an arbitrary protocol port which is sent to the Server when communication is initiated.

E.3. Sending Data Using Streams

Sending data using streams involves establishing a connection-oriented communication channel (Figure E.1). The Server socket listens for Client sockets trying to establish a connection. This is a passive operation during which the Server does not initiate a communication channel. The Server socket is put in this listen mode by using the `listen()` function.

The Client attempts to establish a communication channel by calling the `connect()` function with all the relevant information about the Server socket's protocol port and Internet address number. When the Server receives a request for a connection on the protocol port to which it is bound, it can choose to accept the link by calling the `accept()` function. This function spawns a new socket for the communication channel and releases the Server socket for other Clients to connect to the Server. At this stage, the communication channel is established using the new socket and data can now be sent in either direction.

Data is transmitted and received by the Client and the Server using the same functions. To transmit data the functions `send()` or `write()` are used with the appropriate socket descriptor. It is not necessary to specify the Internet address of the remote computer since the communication channel has been established. The data is received using the `recv()` or `read()` functions to extract any newly arrived data from the Transport Layer.

When all data transfers are complete the `closesocket()` function terminates the communication link (Figure E.1).

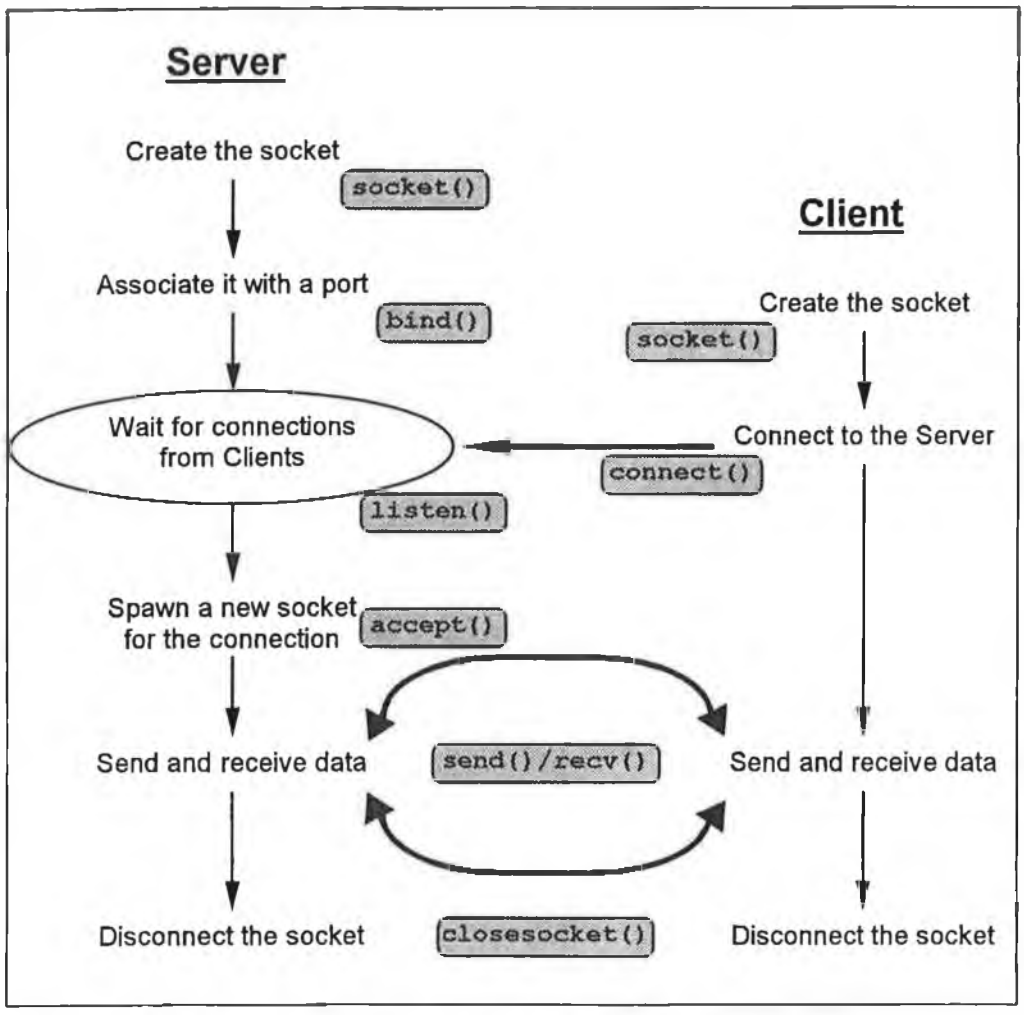


Figure E.1 Client/Server connection using streams

E.4. Sending Data Using Datagrams

Sending data using datagrams does not involve establishing a communication channel (Figure E.2). Instead the data is simply sent with an Internet address as a function parameter and a best-try attempt is made at delivering it. Therefore the sender must know the recipient's protocol port and Internet address numbers, and this will only succeed if the recipient has bound the socket to the protocol port using the `bind()` function. Hence, for two-way data transfers, both the Client and the Server must use the `bind()` function to associate the socket with a protocol port.

Once the Client and the Server have created their sockets and bound them to Protocol ports, the `sendto()` function is used to transmit a datagram to the specified recipient. The recipient then uses the `recvfrom()` function to receive the datagram. It is necessary to specify the Internet address of the remote computer since this is a connectionless communication channel.

When all data transfers are complete the `closesocket()` function terminates the communication link (Figure E.2).

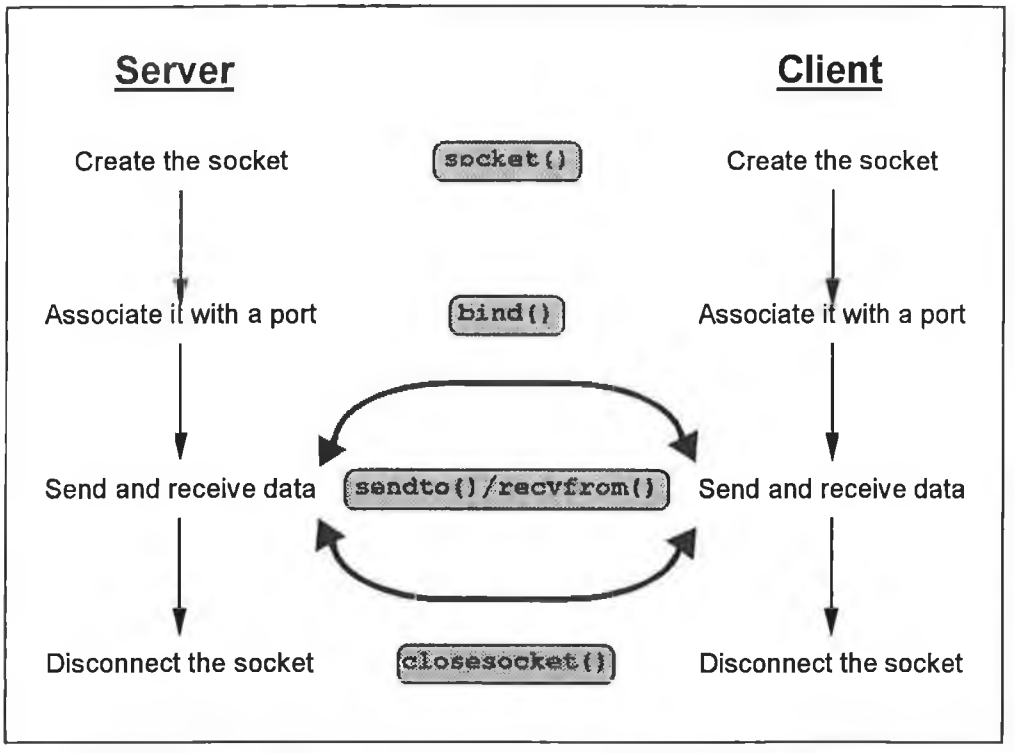


Figure E.2 Client/Server connection using datagrams

E.5. Initialisation and Error Functions

```
int PASCAL FAR WSAStartup(WORD wVersionRequired, LPWSADATA
lpWSAData);
```

Initialises the underlying Windows Sockets Dynamic Link Library (WinSock DLL) before any WinSock-specific functions are called. Permits the vendor of the particular WinSock stack to perform any necessary application-specific

initialisation. The `wVersionRequired` is the highest version of the WinSock API that the calling convention can use, allowing the confirmation that the minimum application version requirements are satisfied by the WinSock stack. `lpWSAData` is a pointer to a data structure which receives details of the Specific WinSock stack available. It contains vendor information, version information, socket information, system information, and a text description.

```
SOCKET PASCAL FAR socket(int af, int type, int protocol);
```

Creates an end-point of communication, called a socket, and specifies the type of protocol used for transferring data over the connection. `type` can be either `SOCK_STREAM` for stream oriented connections or `SOCK_DGRAM` for datagram oriented connections. On success it returns a socket descriptor which uniquely identifies the end-point of communication.

```
int PASCAL FAR closesocket(SOCKET s);
```

Terminates the application's use of a specific socket as specified by `s` whilst flushing out any remaining data to be read/written according to a previously-defined option.

```
int PASCAL FAR WSACleanup(void);
```

Terminates the application's use of the WinSock stack and causes it to release any remaining resources it was using back to the system.

```
int PASCAL FAR WSAGetLastError(void);
```

Returns a code indicating the last WinSock error which occurred and should be called immediately after a function fails to obtain information about the failure.

E.6. Socket Functions

E.6.1. Server-Specific

```
int PASCAL FAR bind(SOCKET s, const struct sockaddr FAR *addr,
int addrlen);
```

Associates the logical socket with a name as specified by the addr structure. It typically specifies a port number for server applications to listen for connections, for example port 39 for ftp connections or port 80 for WWW connections.

```
int PASCAL FAR listen(SOCKET s, int backlog);
```

Causes the socket to synchronously listen for Clients attempting to connect to initiate communication, hence it is used by a sever application.

E.6.2. Client-Specific

```
int PASCAL FAR connect(SOCKET s, , const struct sockaddr FAR *,
int addrlen);
```

Causes the socket to synchronously attempt to connect to the server specified in the addr structure and initiate communication, hence it used by a Client application. Note that the server must already be listening for connections.

```
SOCKET PASCAL FAR accept(SOCKET s, struct sockaddr FAR *, int
FAR *addrlen);
```

Accepts the initiation of communication with a Client. It stores the Client's address in the addr structure for future use and spawns a new socket on which to continue the communication. This releases the original listening socket to permit connections by more than one Client at a time. Each accepted Client gets a uniquely spawned socket for the duration of its communication. This indicates the successful negotiation of a virtual data communication circuit between a Client and a Server and supports the transmission of data using the reliable stream data transfer protocol.

```
int PASCAL FAR sendto(SOCKET s, const char FAR *buf, int len,
int flags, const struct sockaddr FAR *to, int tolen);
```

Used to synchronously send information to an address using the datagram data transfer protocol. This does not require a virtual communication circuit to be established, it blindly sends the data and assumes it gets to the specified destination.

```
int PASCAL FAR recvfrom(SOCKET s, char FAR *buf, int len, int
flags, struct sockaddr FAR *from, int fromlen);
```

Used to synchronously extract data from a socket after being sent using the datagram data transfer protocol.

```
int PASCAL FAR send(SOCKET s, const char FAR *buf, int buflen,
int flags);
```

Used to synchronously send information over a connected virtual communication circuit using the reliable stream data transfer protocol.

```
int PASCAL FAR recv(SOCKET s, char FAR *buf, int buflen, int
flags);
```

Used to synchronously extract data from a socket after being sent over a connected virtual communication circuit using the reliable stream data transfer protocol.

E.7. Format Conversion Functions

```
unsigned long PASCAL FAR inet_addr(const char FAR *cp);
```

Used to convert from the dotted-decimal IP address of the `cp` string representation for an Internet address to the internal 32-bit byte-ordered number required by other WinSock functions.

```
char FAR * PASCAL FAR inet_ntoa(struct in_addr in);
```

Used to convert from the 32-bit byte-ordered number of the `in` structure representation for an Internet address to the string-based dotted-decimal IP.

```
u_long PASCAL FAR htonl(u_long hostlong);
```

Used for platform independence. Converts the `hostlong` long integer from the host number format (little-endian for PC's) to the network format (internationally accepted as big-endian), if necessary.

```
u_short PASCAL FAR htons(u_short hostshort);
```

Used for platform independence. Converts the `hostshort` short integer from the host number format (little-endian for PC's) to the network format (internationally accepted as big-endian), if necessary.

```
u_long PASCAL FAR ntohl(u_long netlong);
```

Used for platform independence. Converts the `netlong` long integer from the network number format (internationally accepted as big-endian) to the host format (little-endian for PC's), if necessary.

```
u_short PASCAL FAR ntohs(u_short netshort);
```

Used for platform independence. Converts the `netshort` short integer from the network number format (internationally accepted as big-endian) to the host format (little-endian for PC's), if necessary.

E.8. Database Functions

```
struct hostent FAR * PASCAL FAR gethostbyaddr(const char FAR
*addr, int addrlen, int type);
```

Uses the IP address specified in the `addr` string to synchronously evaluate its corresponding text name. Retrieves the information from a locally stored lookup table or else sends the request across the network to a name server.

```
struct hostent FAR * PASCAL FAR gethostbyname(const char FAR
*name);
```

Uses the text name specified in the `name` string to synchronously evaluate its corresponding IP address. Retrieves the information from a locally stored lookup table or else sends the request across the network to a name server.

```
int PASCAL FAR gethostname(char FAR *name, int namelen);
```

Obtains the text name of the host computer and stores it in the `name` string. This text name can be used by the `gethostbyname()` function to evaluate the host computers full IP address.

```
struct servent FAR * PASCAL FAR getservbyname(const char FAR
*name, const char FAR *proto);
```

Uses the name specified in the `name` string and the protocol specified in the `proto` string to synchronously retrieve additional information about the service from a locally stored lookup table. Most commonly used services are listed in this file. This function would most frequently be used to uncover the port which must be used for a particular service.

```
struct servent FAR * PASCAL FAR getservbyport(int port, const
char FAR *proto);
```

Uses the port number specified in the `port` integer and the protocol specified in the `proto` string to synchronously retrieve additional information about the service from a locally stored lookup table. Most commonly used services are listed in this file.

E.9. Asynchronous Windows Functions

```
HANDLE PASCAL FAR WSAAsyncGetHostByAddr(HWND hWnd, u_int wMsg,
const char FAR *addr, int len, int type, char FAR *buf, int
buflen);
```

Uses the IP address specified in the `addr` string to asynchronously evaluate its corresponding text name. Retrieves the information from a locally stored lookup table or else sends the request across the network to a name server and stores it in the `buf` string buffer.

```
HANDLE PASCAL FAR WSAAsyncGetHostByName(HWND hWnd, u_int wMsg,
const char FAR *name, char FAR *buf, int buflen);
```

Uses the text name specified in the `name` string to asynchronously evaluate its corresponding IP address. Retrieves the information from a locally stored lookup table or else sends the request across the network to a name server and stores it in the `buf` string buffer.

```
HANDLE PASCAL FAR WSAAsyncGetServByName(HWND hWnd, u_int wMsg,
const char FAR *name, const char FAR *proto, char FAR *buf, int
buflen);
```

Uses the name specified in the `name` string and the protocol specified in the `proto` string to asynchronously retrieve additional information about the service

from a locally stored lookup table and stores it in the `buf` buffer. This function would most frequently be used to uncover the port which must be used to access a particular service.

```
HANDLE PASCAL FAR WSAAsyncGetServByPort(HWND hWnd, u_int wMsg,
const char FAR *proto, char FAR *buf, int buflen);
```

Uses the port number specified in the `port` integer and the protocol specified in the `proto` string to asynchronously retrieve additional information about the service from a locally stored lookup table and store it in the `buf` buffer.

```
int PASCAL FAR WSAAsyncSelect(SOCKET s, HWND hWnd, u_int wMsg,
long lEvent);
```

Used for asynchronous function calls by putting the `s` socket in a non-blocking mode of operation. Notifies the Windows messaging system of the types of socket events the socket is interested in, for example connection established or data available to be processed.